Josep Díaz
Juhani Karhumäki
Arto Lepistö
Donald Sannella (Eds.)

LNCS 3142

Automata, Languages and Programming

31st International Colloquium, ICALP 2004
Turku, Finland, July 2004
Proceedings

Springer

Lecture Notes in Computer Science 3142

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

This page intentionally left blank

Josep Díaz Juhani Karhumäki
Arto Lepistö Donald Sannella (Eds.)

Automata, Languages and Programming

31st International Colloquium, ICALP 2004
Turku, Finland, July 12-16, 2004
Proceedings

Springer

eBook ISBN: 3-540-27836-2
Print ISBN: 3-540-22849-7

Created in the United States of America

Visit Springer's eBookstore at: http://ebooks.springerlink.com
and the Springer Global Website Online at: http://www.springeronline.com

Preface

The 31st International Colloquium on Automata, Languages, and Programming (ICALP 2004) was held from July 12 to July 16 in Turku, Finland. This volume contains all contributed papers presented at ICALP 2004, together with the invited lectures by Philippe Flajolet (INRIA), Robert Harper (Carnegie Mellon), Monika Henzinger (Google), Martin Hofmann (Munich), Alexander Razborov (Princeton and Moscow), Wojciech Rytter (Warsaw and NJIT), and Mihalis Yannakakis (Stanford).

ICALP is a series of annual conferences of the European Association for Theoretical Computer Science (EATCS). The first ICALP took place in 1972 and the ICALP program currently consists of track A (focusing on algorithms, automata, complexity, and cryptography) and track B (focusing on databases, logics, semantics, and principles of programming).

In response to the call for papers, the program committee received 379 papers, 272 for track A and 107 for track B. This is the highest number of submitted papers in the history of ICALP conferences. The program committees selected 97 papers for inclusion into the scientific program. The program committee for track A met on March 27 and 28 in Barcelona and selected 69 papers from track A. The program committee for track B selected 28 papers from track B in the course of an electronic discussion lasting for two weeks in the second half of March.

The selections were based on originality, quality, and relevance to theoretical computer science. We wish to thank all authors who submitted extended abstracts for consideration, the program committee for its hard work, and all referees who assisted the program committee in the evaluation process.

The EATCS best paper award for track A was given to the paper "Quantum Query Complexity of Some Graph Problems" by Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla, and the award for track B was given to the paper "Tree-Walking Automata Cannot Be Determinized" by Mikolaj Bojanczyk and Thomas Colcombet. Ryan Williams received the best student paper award for track A for his contribution "A New Algorithm for Optimal Constraint Satisfaction and Its Implications", and the best student paper award for track B was given to Olivier Serre for his paper "Games with Winning Conditions of High Borel Complexity".

ICALP 2004 was held in conjunction with the 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004). ICALP 2004 was also notable as, for the first time, ICALP returned to the same town: Turku also hosted ICALP 1977. The ICALP 2004 webpages can be found at http://www.math.utu.fi/icalp04/.

During ICALP 2004 the following special events also took place: A colloquium in honor of Academician Arto Salomaa on the occasion of his 70th birthday was organized on July 11, and the EATCS award was given to Arto Salomaa. The following workshops were held as satellite events of ICALP 2004 and LICS 2004

with Mika Hirvensalo as the coordinator: 1st International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS), Workshop on Discrete Models for Complex Systems (DMCS), Workshop on Foundations of Computer Security (FCS), Workshop on Intersection Types and Related Systems (ITRS), 6th International Workshop on Logic and Computational Complexity (LCC), Workshop on Logics for Resources, Processes, and Programs (LRPP), Workshop on Logic and Systems Biology (LSB), 2nd International Workshop on Quantum Programming Languages (QPL), Workshop on Word Avoidability, Complexity and Morphisms (WACAM), Workshop on Logical Foundations of an Adaptive Security Infrastructure (WOLFASI), and Workshop on Synchronizing Automata (WSA).

We thank the sponsors and the Turku University Mathematics Department and Turku Centre for Computer Science for hosting ICALP 2004. We are also grateful to the Turku University Congress Office, organizing committee, and to the local automata theory group: Vesa Halava, Tero Harju, Jarkko Kari, Elisa Mikkola, Kalle Saari, Petri Salmela, Magnus Steinby, and, in particular, Mika Hirvensalo.

April 2004

Josep Diaz
Juhani Karhumäki
Arto Lepistö
Donald Sannella

Program Committee

Track A

A. Atserias	*Barcelona, Spain*
G. Brodal	*Aarhus, Denmark*
J. Cassaigne	*Marseille, France*
J. Diaz, *Chair*	*Barcelona, Spain*
R. Fleischer	*Hong Kong, China*
H. Gabow	*Boulder, USA*
L. Goldberg	*Warwick, UK*
J. Hromkovic	*Aachen, Germany*
G. Italiano	*Rome, Italy*
T. Jiang	*Riverside, USA*
C. Kaklamanis	*Patras, Greece*
J. Kari	*Turku, Finland*
C. Moore	*Santa Fe, USA*
P. Pudlak	*Prague, Czech Republic*
P. Raghavan	*Verity, Stanford, USA*
M. Santha	*Paris, France*
B. Voecking	*Dortmund, Germany*
G. Woeginger	*Twente, The Netherlands*
M. Yung	*Columbia University, USA*

Track B

R.-J. Back	*Turku, Finland*
P.-L. Curien	*Paris, France*
A. Gordon	*Microsoft, Cambridge, UK*
S. Hayashi	*Kobe, Japan*
T. Henzinger	*Berkeley, USA*
M. Hofmann	*Munich, Germany*
B. Jacobs	*Nijmegen, The Netherlands*
E. Moggi	*Genoa, Italy*
J. Parrow	*Uppsala, Sweden*
C. Palamidessi	*University Park, PA, USA*
B. Pierce	*Philadelphia, USA*
A. Rabinovich	*Tel Aviv, Israel*
D. Sannella, *Chair*	*Edinburgh, UK*
W. Thomas	*Aachen, Germany*
I. Walukiewicz	*Bordeaux, France*

Organizing Committee

J. Karhumäki, *Conference Chair*
T. Järvi, *Co-chair* (ICALP)
L. Hella, *Co-chair* (LICS)
V. Halava
M. Hirvensalo
I. Petre
P. Sibelius
T. Knuutila

List of Referees

Scott Aaronson, Martín Abadi, Parosh Abdulla, Andreas Abel, Dimitris Achlioptas, Susanne Albers, Jürgen Albert, Jean-Paul Allouche, Noga Alon, Jan-Hendrik Altenbernd, Thorsten Altenkirch, Carme Alvarez, Andris Ambainis, Torben Amtoft, Davide Ancona, Suzana Andova, Nicola Apollonio, Lars Arge, André Arnold, V. Arvind, Eugene Asarin, David Aspinall, Robert Atkey, Arnon Avron, Patrick Baillot, Steffen van Bakel, Meenakshi Balasubramanian, Michael Baldamus, Howard Barnum, Yair Bartal, Peter Bartlett, Paul Beame, Daniele Beauquier, Moritz Becker, Eli Ben-Sasson, Nick Benton, Stefano Berardi, Petra Berenbrink, Martin Berger, Lennart Beringer, Ron Berman, Julien Bernat, Dietmar Berwanger, Guy Blelloch, Achim Blumensath, Hans Bodlaender, HansJoachim Boeckenhauer, Bernard Boigelot, Mikołaj Bojańczyk, Dirk Bongartz, Johannes Borgström, Victor Bos, Ahmed Bouajjani, Alexis-Julien Bouquet, Claus Brabrand, Julian Bradfield, Cees-Bart Breunesse, François Bry, Adam Buchsbaum, Yann Bugeaud, Michele Bugliesi, Stefan Burkhardt, Jesper Byskov, Cristiano Calcagno, Manuel Campagnolo, Ran Canetti, Alberto Caprara, Arnaud Carayol, Marco Carbone, Sébastien Carlier, Rafel Cases, Paul Caspi, Chandra Chekuri, Bogdan Chlebus, Hana Chockler, Christian Choffrut, Marek Chrobak, Gabriel Ciobanu, Aaron Clauset, Bruno Codenotti, Christopher Colby, Richard Cole, Murray Cole, Adriana Compagnoni, Vince Conitzer, Matthew Cook, Mario Coppo, Graham Cormode, Felix Costa, Nadia Creignou, Claude Crepeau, Mary Cryan, Felipe Cuker, Thomas Cusick, Eugen Czeizler, Artur Czumaj, Mads Dam, Vincent Danos, Mayur Datar, Anuk Dawar, Giorgio Delzanno, Yuxin Deng, Yvo Desmedt, Nikhil Devanur, Mariangiola Dezani, Martin Dietzfelbinger, Yevgeniy Dodis, Jaques Duparc, Bruno Durand, Christophe Durr, Peter Dybjer, Martin Dyer, David Eichmann, Cindy Eisner, Edith Elkind, Lars Engebretsen, Leah Epstein, Thomas Erlebach, William S. Evans, Marco Faella, Rolf Fagerberg, Wenfei Fan, Sandor Fekete, Jon Feldman, Fabrice Le Fessant, Jiri Fiala, Amos Fiat, Alain Finkel, Philippe Flajolet, Fedor Fomin, Lance Fortnow, Dimitris Fotakis, Cédric Fournet, Gudmund Skovbjerg Frandsen, Katalin Friedl, Alain Frisch, Murdoch Gabbay, Peter Gacs, Fabio Gadducci, Anna Gal, Shmuel Gal, Vladimir Gapeyev, Naveen Garg, Luisa Gargano, Pablo Garralda, Leszek Gasieniec, Ricard Gavalda, Simon Gay, Herman Geuvers, Pablo Giambiagi, Raffaele Giancarlo, Anna Gilbert, Jean-Yves Girard, Rob van Glabbeek, Andrew Goldberg, Paul Goldberg, Massimiliano Goldwurm, Mordechai Golin, Clemens Grabmayer, Erich Graedel, Etienne Grandjean, Vladimir Grebinski, M. Greenwald, Roberto Grossi, Stefano Guerrini, S. Guha, Anupam Gupta, Claudio Gutierrez, Peter Habermehl, Esfandiar Haghverdi, Masami Hagiya, Péter Hajnal, Vesa Halava, Joe Hallett, Jiri Hanika, Tero Harju, James Harland, Sariel Har-Peled, Robert Harper, Alan Hartman, Frédéric Herbreteau, Miki Hermann, Holger Hermanns, Andreas Herzig, Jane Hillston, Peter Hines, Hiromi Hiraishi, Daniel Hirschkoff, Yoram Hirshfeld, Mika Hirvensalo, John M. Hitchcock, Jaap-Henk Hoepman, Juha Honkala, Furio Honsell, Han Hoogeveen, Peter Hoyer, Engelbert Hubbers, Jesse Hughes, Cor Hurkens, Martin Hyland, John Iacono,

Russell Impagliazzo, Sandy Irani, Gábor Ivanyos, Paul Jackson, Riko Jacob, Jens Jaegerskuepper, Radha Jagadeesan, David Janin, Klaus Jansen, Wojciech Jawor, Ole Hoegh Jensen, Mark Jerrum, Bengt Jonsson, Tibor Jordan, Stasys Jukna, Marcin Jurdzinski, Eija Jurvanen, Haim Kaplan, Marek Karpinski, Mikhail Kats, Shin-ya Katsumata, Jonathan Katz, David Kempe, Julia Kempe, Delia Kesner, Assaf Kfoury, S. Khanna, Joe Kiniry, Christian Kirkegaard, Lefteris Kirousis, Jyrki Kivinen, Felix Klaedtke, Bettina Klinz, Jan Willem Klop, Hirotada Kobayashi, Pascal Koiran, Petr Kolman, Beata Konikowska, Swastik Kopparty, Vaclav Koubek, Elias Koutsoupias, Lukasz Kowalik, Jan Krajicek, Daniel Kral, Robert Krauthgamer, Pavel Krcal, Marc van Kreveld, Jean-Louis Krivine, Andrei Krokhin, Antonin Kucera, Manfred Kufleitner, Joachim Kupke, Maciej Kurowski, Dietrich Kuske, Jyrki Lahtonen, Linas Laibinis, Jim Laird, Cosimo Laneve, Martin Lange, Sophie Laplante, Oukseh Lee, James Leifer, Marina Lenisa, Stefano Leonardi, Arto Lepistö, Martin Leucker, Asaf Levin, Michael Levin, Jing Li, Johan Lilius, Christof Loeding, John Longley, Lazlo Lovatz, Chi-Jen Lu, Ga'bor Lugosi, Olivier Ly, Rune Bang Lyngs, Kenneth MacKenzie, Rupak Majumdar, Henning Makholm, Christos Makris, Oded Maler, David Manlove, Yishay Mansour, Russ Martin, Conrado Martinez, Yossi Matias, Ralph Matthes, Marios Mavronicolas, Dominic Mayers, Elvira Mayordomo, Richard Mayr, Jacques Mazoyer, Paul-André Melliès, Michael Mendler, Marino Miculan, Dale Miller, Peter Bro Miltersen, Michael Mislove, Michael Mitzenmacher, Faron Moller, Michael Molloy, Alberto Momigliano, T. Monteil, Remi Morin, Christian Worm Mortensen, Gabriel Moruz, Michele Mosca, Haiko Mueller, Tetsuya Murai, Andrzej Murawski, Anca Muscholl, Muthu Muthukrishnan, Jerzy Mycka, Hiroshi Nakano, Francesco Zappa Nardelli, Ashwin Nayak, Rolf Niedermeier, Mogens Nielsen, Damian Niwinski, Thomas Noll, Antje Nowack, Dirk Nowotka, Satoshi Obana, Mizuhito Ogawa, Atsushi Ohori, Mitsuhiro Okada, Roberto Oliveira, Nicolas Ollinger, Luke Ong, Martijn Oostdijk, Gianpaolo Oriolo, Pekka Orponen, Masanao Ozawa, Rasmus Pagh, Jens Palsberg, Christos Papadimitriou, Anargyros Papageorgiou, Kunsoo Park, Mike Paterson, Dirk Pattinson, Wojciech Peczek, Christian N.S. Pedersen, Rudi Pendavingh, Thomas Perst, Ion Petre, Elena Petre, Anna Philippou, Pierre Philipps, Jean-Eric Pin, David Pisinger, Marco Pistore, Andrew Pitts, Erik Poll, Enrico Pontelli, Ivan Porres, François Pottier, John Power, James Propp, Kirk Pruhs, Tomasz Radzik, Nasir Rajpoot, Sergio Rajsbaum, Rajeev Raman, Dana Randall, Srinivasa Rao, Ivan Rapaport, Julian Rathke, Ran Raz, Oded Regev, Laurent Regnier, Ari Renvall, Tamara Rezk, Yossi Richter, James Riely, Noam Rinetzky, Liam Roditty, Vojtech Rodl, Philipp Rohde, Dana Ron, Amir Ronen, Giuseppe Rosolini, Michel de Rougemont, Tim Roughgarden, Salvador Roura, James Royer, Ronitt Rubinfeld, Alexander Russell, Wojciech Rytter, Morten Sørensen, Andrei Sabelfeld, Amr Sabry, Cenk Sahinalp, Jared Saia, Chiaki Sakama, Saeed Salehi, Arto Salomaa, Davide Sangiorgi, Vishal Sanwalani, Vladimiro Sassone, Alexis Saurin, Petr Savicky, Vladimir Sazonov, Nicolas Schabanel, Gilles Schaeffer, Christian Scheideler, Philippe Schnoebelen, Peter Schuster, Eric Schwabe, Michael Schwartzbach, Nicole Schweikardt, Thomas Schwentick, Roberto Segala, Sebastian Seibert,

Helmut Seidl, Kaisa Sere, Maria Serna, Peter Sewell, Jiri Sgall, Hadas Shachnai, Cosma Shalizi, John Shawe-Taylor, Patrick Sibelius, Jérome Simeon, Alex Simpson, Rene Sitters, Martin Skutella, Michiel Smid, Christian Sohler, Paul Spirakis, Bas Spitters, Jiri Srba, Aravind Srinivasa, Jesssica Staddon, Ian Stark, Sam Staton, Rob van Stee, Gheorghe Stefanescu, Cliff Stein, Magnus Steinby, Colin Stirling, Marielle Stoelinga, Viggo Stoltenberg-Hansen, Leen Stougie, M. Strauss, Thomas Streicher, Zhendong Su, Benny Sudakov, Grégoire Sutre, Maxim Sviridenko, Yasuhiro Takahashi, Jean-Marc Talbot, Arie Tamir, Alain Tapp, Dennis Therien, Dimitris Thilikos, Ferucio Laurentiu Tiplea, Alwen Tiu, Jacobo Torán, Salvatore La Torre, Luca Trevisan, Stavros Tripakis, Costas Tsichlas, Gene Tsudik, Walter Unger, Christian Urban, Pawel Urzyczyn, Patchrawat Uthaisombut, Phil Wadler, Frank Valencia, David Walker, John Watrous, Joseph Vanderwaart, Daniele Varacca, Kasturi Varadarajan, Martijn Warnier, Vasco Vasconcelos, Pascal Weil, Joe Wells, Yves Verhoeven, Joannes Vermorel, Björn Victor, Maria Grazia Vigliotti, David Williamson, Glynn Winskel, Lucian Wischik, Stefan Woehrle, Walter Vogler, Ronald de Wolf, Pierre Wolper, Nick Wormald, Joakim von Wright, Laurent Vuillon, Jens Vygen, Akihiro Yamamoto, Amiram Yehudai, Greta Yorsh, Neal Young, Shoji Yuen, Stanislav Zak, Steve Zdancewic, Jie Zheng, Wieslaw Zielonka, Silvano Dal Zilio, Pascal Zimmer, David Zuckerman, Uri Zwick

Sponsors

Academy of Finland
City of Turku
Finnair
Finnish Academy of Sciences and Letters
- Väisälä Foundation
- Mathematics Foundation
Finnish Cultural Foundation
Fujitsu-Siemens
IBM
MasterPlanet
Nokia
Nordea
Sampo Life insurance Company Limited
Stiftelsen för Åbo Akademi
Turku Centre for Computer Science (TUCS)
Turku University Foundation
Turun Seudun Osuuspankki
University of Turku

Table of Contents

Invited Talks

Contributed Papers

This page intentionally left blank

Self-Adjusting Computation*

Robert Harper

Carnegie Mellon University
Computer Science Department
Pittsburgh, PA 15213 USA
rwh@cs.cmu.edu

A *static algorithm* is one that computes the result of a query about the output for a single, fixed input. For example, a static sorting algorithm is one that takes as input a set of keys, and permits queries about the relative order of these keys according to some ordering relation. A *dynamic,* or *incremental, algorithm* is one that permits queries about the output to be interleaved with operations that incrementally modify the input. For example, a dynamic sorting algorithm is one that would permit insertion or deletion of keys to be interleaved with queries about their relative ordering.

It is often easier to find a static algorithm than a dynamic algorithm for a given problem. There is a large and growing literature on dynamic algorithms for a broad range of problems.[1] *Self-adjusting computation* is a method for deriving a dynamic algorithm for a problem by "dynamizing" a static algorithm for it [4]. We have studied three main techniques for dynamization:

1. *Adaptivity* [1]. An *adaptive computation* is one which is capable of adjusting its output in response to a specified class of changes to its input. Execution records sufficient information about the dependencies among sub-computations as to permit quick identification of the affected parts, and the re-execution of those parts invalidated by the change.
2. *Selective memoization* [3]. Conventional memoization is *data-driven:* the output of a function for a given input is stored so that it may be recalled if the function is called again with the same input. Selective memoization is *control-driven:* the output associated with a given dynamic control flow path arising from a partial examination of the input is stored for future use should it arise again.
3. *Adaptive memoization* [2]. Adaptive memoization associates an adaptive computation, rather than just a value, with a control path in a program. This permits recovery of invalid sub-computations whose validity may be restored by adapting it to the dynamic memoization context.

We have used these three techniques to obtain a dynamic version of Quicksort that permits insertion and deletion of keys in expected $O(\lg n)$ time [2], and a dynamic version of Parallel Tree Contraction [4]. We have also used these methods to obtain kinetic versions of Quicksort and MergeSort (with expected constant-time response to kinetic changes) and of QuickHull and MergeHull for computing convex hulls [5].

Our techniques for self-adjusting computation are formally defined by a static and a dynamic semantics. The static semantics is based on a Curry-Howard interpretation

* Joint work with Umut A. Acar and Guy E. Blelloch.

[1] Please see the papers cited below for a survey of some of this work.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1–2, 2004.

of two forms of *modal logic,* namely *lax logic* (for adaptivity) and *S4 modal logic* (for selective memoization). The lax modality captures the distinction between *stable* and *changeable* computations, those that do and do not, respectively, depend on admissible modifications to the input. The S4 necessity modality captures the partial dependence of the output on a limited aspect of a data structure, permitting the greatest flexibility in re-using old results. These typing mechanisms ensure that dependencies are not lost, that changes are propagated correctly, and that memoized computations are properly adapted to changes before they are re-used.

The dynamic semantics of self-adjusting computation is based on the maintenance of *dynamic dependency graphs,* or *DDG's,* and *control paths,* or *CP*'s. The role of DDG's is to maintain a record of the dependencies among sub-computations in such a way that the (direct and indirect) effects of a change to the input may be isolated and invalidated. The DDG permits an invalid computation to be reconstructed for the revised input by re-executing the code used to create it in the first place. The role of CP's is to record the dependency of the output of a computation on only some aspect of its input (such as a certain portion of a complex structure) or some abstraction of it (such as the diameter of a graph). For the sake of execution efficiency, DDG's are implemented using a variant of the Dietz-Sleator order maintenance data structure [6], and CP's are implemented using conventional hashing techniques.

The effectiveness of adaptive computation may be attributed in equal measure to the application of methods from discrete algorithms and language semantics. The linguistic tools permit the concise formulation of a dynamic version of a static algorithm with minimal complication. The algorithmic tools permit their efficient implementation so that the cost of the adaptive computation mechanisms does not exceed their (asymptotic) benefit. The formal semantics supports a precise complexity analysis and correctness proof of the dynamic algorithms we consider.

References

[1] U. A. Acar, G. E. Blelloch, and R. Harper. Adaptive functional programming. In *Proceedings of the 29th Annual ACM Symposium on Principles of Programming Languages,* pages 247–259, 2002.

[2] U. A. Acar, G. E. Blelloch, and R. Harper. Adaptive memoization. Technical report, Department of Computer Science, Carnegie Mellon University, March 2003. Available at http://www.cs.cmu.edu/~rwh/papers.htm#admem.

[3] U. A. Acar, G. E. Blelloch, and R. Harper. Selective memoization. In *Proceedings of the 30th Annual ACM Symposium on Principles of Programming Languages,* 2003.

[4] U. A. Acar, G. E. Blelloch, R. Harper, J. L. Vittes, and M. Woo. Dynamizing static algorithms with applications to dynamic trees and history independence. In *ACM-SIAM Symposium on Discrete Algorithms (SODA),* 2004.

[5] U. A. Acar, G. E. Blelloch, and J. L. Vittes. Convex hulls for dynamic data, 2004. In preparation.

[6] P. F. Dietz and D. D. Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the 19th ACM Symposium on Theory of Computing,* pages 365–372, 1987.

The Past, Present, and Future of Web Search Engines

Monika Henzinger

Google Inc
1600 Amphitheatre Parkway
Mountain View, CA 94043
monika@google.com

Web search engines have emerged as one of the central applications on the Internet. In fact, search has become one of the most important activities that people engage in on the the Internet. Even beyond becoming the number one source of information, a growing number of businesses are depending on web search engines for customer acquisition.

The first generation of web search engines used text-only retrieval techniques. Google revolutionized the field by deploying the PageRank technology – an eigenvector-based analysis of the hyperlink structure – to analyze the web in order to produce relevant results. Moving forward, our goal is to achieve a better understanding of a page with a view towards producing even more relevant results.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, p. 3, 2004.

What Do Program Logics and Type Systems Have in Common?

Martin Hofmann*

Department of Informatics, University of Munich,
Oettingenstraße 67, 80538 München
Germany
mhofmann@informatik.uni-muenchen.de

This talk tries to contribute to a discussion started by John Reynolds' in his short presentation ("five minute madness talk") at the SPACE workshop 2004 (http://www.diku.dk/topps/space2004/).

Program logics such as Hoare logic or indeed any formalisation of operational semantics allow one to specify properties of programs and to formally prove them. In particular, simple safety properties such as absence of "method not understood" or non-violation of array bounds have been successfully established using program logics for substantial pieces of code.

Type systems seem to serve a similar purpose; they, too, promise to ensure safety properties of programs starting from R Milners celebrated slogan "well-typed programs do not go wrong."

The big advantage of a type system is its low (practical) complexity and its guaranteed success. A disadvantage of a more sociological nature is the enormous quantity of mutually incompatible type systems that have been developed and also the syntactic nature of the properties they purport to establish which makes it sometimes difficult to compare their merits.

The main advantage of type systems over program logics seems to dwindle in view of impressive recent progress in the area of automatic inductive theorem proving and software model checking. Will type systems therefore die out? I will argue in the talk that the answer is no and propose a useful synthesis between type systems and program logics that would also help addressing the aforementioned compatibility problem.

In a nutshell, the synthesis is that a type system can help to automatically generate invariants to be used in a proof based on program logic. In other words, the type system provides a high-level front end to a program logic.

One concrete example that I will use in the talk is a Java version of insertion sort using a free list. The goal is to prove (as automatically as possible!) that the line marked /*DON'T*/ is never executed.

* Partial support by the EU-funded project "Mobile Resource Guarantees" (IST-2001-33149) is herewith gratefully acknowledged.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 4–7, 2004.

```
class List{ static List freelist;

    int head;
    List tail;

    List(int head,List tail){
        this.head=head;
        this.tail=tail;
    }

    public static void init(int n) {
    // initialise freelist
        for(int i = 1;i<=n;i++){
            freelist = new List(0,(List)freelist);
        }
    }

    public static void free(List l) {
        l.tail = freelist;
        freelist = l;
    }

    public static List make(){
        if (freelist != null) {
            List res = freelist;
            freelist = freelist.tail;
            return res;
        } else {
/* DON'T */ System.out.println("Called new within make");
            return new List(0,(List)null);
        }
    }

    public static List nil(){
        return null;
    }

    public static List cons(int i,List l){
        List res = make();
        res.head = i;
        res.tail = l;
        return res;
    }
```

```
    public static boolean isnil(List l){
        return (l==null);
    }

    public static int head(List l){
        return l.head;
    }

    public static List tail(List l){
        return l.tail;
    }
}



class InsSort{
    public static void main(String[] args){
        List.init(2 * (args.length-1));

        List l = List.nil();

        for(int i=1;i<args.length;i++) {
            l = List.cons(Integer.valueOf(args[i]).intValue(),l);
        }
        printlist(sort(l));
    }

    public static void printlist(List l){
        if (List.isnil(l))
            System.out.println();
        else {
            System.out.print(List.head(l)+" ");
            printlist(List.tail(l));
        }
    }

    public static List insert(int i, List l){
        if (List.isnil(l))
            return List.cons(i,List.nil());
        else {
            int h = List.head(l);
            List t = List.tail(l);
            List.free(l);
            if (h <= i){
                return List.cons(h, insert(i,t));
```

```
            } else {
                return List.cons(i,List.cons(h,t));
            }
        }
    }

    public static List sort(List l){
        if (List.isnil(l))
            return List.nil();
        else {
            int h = List.head(l);
            List t = List.tail(l);
            return insert(h,sort(t));
        }
    }
}
```

Feasible Proofs and Computations: Partnership and Fusion

Alexander A. Razborov*

Institute for Advanced Study
School of Mathematics
Princeton, NJ 08540, USA
razborov@math.ias.edu

Abstract. A computation or a proof is called feasible if it obeys prescribed bounds on the resources consumed during its execution. It turns out that when restricted to this world of feasibility, proofs and computations become extremely tightly interrelated, sometimes even indistinguishable. Moreover, many of these rich relations, underlying concepts, techniques etc. look very different from their "classical" counterparts, or simply do not have any. This talk is intended as a very informal and popular (highly biased as well) attempt to illustrate these fascinating connections by several related developments in the modern complexity theory.

1 Introduction

Proofs and computations are among the most basic concepts pertinent to virtually any intellectual human activity. Both have been central to the development of mathematics for several millenniums. The effort to study these concepts themselves in a rigorous, metamathematical way initiated in the 20th century led to flourishing of the mathematical logic and derived disciplines like those which are the focus of attention of both conferences gathering here.

The relation between proofs and computation in mathematics never was easy. In particular, the debate as to what makes a worthy mathematical result – a deductive inference from accepted axioms, possibly shockingly non-constructive or a practical procedure leading to the desired results but possibly lacking a rigorous justification – was sometimes hot (at the time of crises), sometimes lukewarm, but ever-present it was. And as we all know well, precise formalizations of both these fundamental concepts given by great logicians of the last century at least put this debate onto a solid ground, even if this did not seem to extinguish it completely. Since that time we at least more or less universally agree on *what* is a proof and *what* is a computation. The connections between them are extremely diverse, rich and mutually beneficial: we study computations with formal proofs, try to write programs for computer-aided theorem proving, use formal proofs for the program verification etc. Still, it appears (at least to the speaker) that no matter how we are playing with computations and proofs, they keep their unique

* On leave from Steklov Mathematical Institute, Moscow, Russia. Supported by the State of New Jersey, The Bell Companies Fellowship, The James D. Wolfensohn Fund, and The Ellentuck Fund.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 8–14, 2004.

identities, and at every particular moment it is utterly clear to which of the two realms the object of interest belongs. In our communities this difference is often articulated as the difference between the syntax and the semantics.

Most of the above are, of course, just common places for this audience (and we will go over them very quickly in the actual talk). But this makes a necessary background for the main point we will try to make. Namely, when we restrict our attention to *feasible* proofs and computations (which are most often defined as polynomially bounded in terms of length or time), then this clear classical (that is, in the absence of such restrictions) picture all of a sudden becomes rather different, and in general more intricate and saturated. Some of the relations between classical proofs and computations gain in importance, whereas some become almost obsolete. New unexpected and beautiful connections emerge on the conceptual level, as well as on the level of proof techniques. Finally, even the separate identities of proofs and computations are compromised by the important discovery of "interactive proofs" that can be (and are) thought of as both proofs and computations, subjectively and interchangeably[1]. And, to make the story even more interesting, all these trends are interweaving and re-enforcing each other.

Inherent reasons for these *qualitative* changes in behaviour resulting from a *quantitative* change in the framework (that is, when we place some *numerical* bounds on the resources) are far from being understood, and this talk will not pretend to offer any such explanation beyond one simple observation. Our main purpose will simply be to illustrate the wonderful interplay between feasible proofs and feasible computations by a few important examples. These will be borrowed from several rather different subareas of the modern complexity theory. As a result, this talk should not be considered as a survey in none of them, and it will, out of necessity, be lacking precise definitions and statements (Section 3 below contains some suggested literature for further study of the subjects we will only superficially touch in the talk). Moreover, the selection of topics from each area will be heavily biased by our main goal: illustrate various intricate connections existing between proofs and computations when both are restricted to the world of feasible objects.

2 Plan of the Talk

We will try to do as much of the following as time permits (although, it does not seem realistic to cover all these issues). The arrangement of topics is somewhat arbitrary, although we will try to stick to this general principle: begin with concepts that still look similar to their "classical" counterparts, gradually moving to the regions where these similarities fade away.

It is (or at least should be) already well-known these days what is a "feasible" computation: this is a computation that obeys a prescribed bound on computational resources like time, space etc. It is intuitively less clear what is a feasible proof. It is

[1] Needless to say, these latter creatures completely decline to behave like "straight-line programs" when viewed as a computation or as a Hilbert-style inference when viewed as a proof! Instead they pay quite a fair share in the analysis of "normal" proofs and "normal" computations.

natural to assume that a feasible proof should not involve objects that are unfeasibly (= exponentially) large, but is this sufficient? We will begin with one canonical example (Fermat's Little Theorem) illustrating that the right answer should be "no", and that all objects involved in a feasible *proof* must be also feasibly *computable.*

We then move on to *Bounded Arithmetic* which is a generic name for a series of first-order or second-order theories capturing this notion of a feasible proof, and cover a few (relatively simple) *witnessing theorems.* The question whether the hierarchy of these theories is proper is the central open problem in this area, which is reminiscent of just the same situation in the computational world.

From this point on, almost everything in this talk will be about propositional (as opposed to first-order) logic, and we will be mostly concerned with two fundamental (and dual to each other) questions:

- How to prove efficiently that a propositional formula ϕ is a tautology?
- How to prove efficiently that a propositional formula ϕ is satisfiable?

We compare these questions with their first-order counterparts, and note a drastic difference in their behaviour.

Then we will address the "textbook" approach to proving that a propositional formula ϕ is a tautology, which consists in deriving ϕ in a Hilbert-style (or Gentzen-style) propositional calculus. We will be interested in the complexity (most often measured by the bit size) of such *propositional proofs,* and we will indicate numerous connections existing between the complexity of propositional proofs and circuit complexity. We will give a couple of lower bound results illustrating the fruitfulness of these connections for both areas. In particular, we will spend some time on the so-called *Feasible Interpolation Theorem,* as well as on the results limiting its use for stronger proof systems that are based upon complexity hardness assumptions.

Next we will move on to the question of feasible provability of a non-uniform version of $\mathbf{P} \neq \mathbf{NP}$. In particular, we will address one approach to this question based upon an adaptation of the notion of a pseudorandom function generator to proof complexity. Specifically, we will talk about the conjecture stating that for the so-called *Nisan-Wigderson generators* their computational hardness always implies hardness for the corresponding proof system, and survey known results for weak proof systems supporting this conjecture.

In everything we have encountered so far, proofs and computations often came very close to each other, but still they did not blend together. The Pandora's box was open in one of the most influential mathematical articles of the last century, [16] which is even entitled suggestively "The complexity of theorem *proving procedures*". Namely, the definition of the fundamental complexity class **NP** given in that paper is inherently dual, and can be viewed both in terms of non-deterministic *computations* and *proofs* of membership. We will reflect a little bit on this duality.

After the news of the marriage between proofs and computations spread around, and this idea soon became one of the main paradigms of the modern complexity theory, it was only a matter of time before more offsprings would be conceived, and the most fruitful notion born in this way was that of *interactive proofs.* The prover no longer prepares a proof on a sheet of paper in the silence of her office and then submits it to a journal for

verification, but rather interacts with the verifier trying to convince him in the validity of her results in the "good common sense". Remarkably, both are making a non-trivial *computational* effort during this interaction. One of the most unexpected results of the complexity theory states, in a weaker form, that one can efficiently prove in this way that a propositional formula ϕ is a tautology, something we strongly believe no "ordinary" (say, strictly propositional) proof system will ever achieve! Unfortunately, even a sketch of this remarkable result is way too technical to fit into this lecture, but we will at least try to illustrate the power of interactive proofs using (more or less standard) example of GRAPH NON-ISOMORPHISM.

So far we concentrated on our first task itemized above, and there seems to be a very good reason for this: it is very easy to prove that ϕ is satisfiable simply be exhibiting a satisfying assignment (it is an entirely different story, of course, how *to find* such an assignment). This trivial proof is easily checkable, and it is feasible in any sense we have seen so far. It, however, turns out, that in the realm of interactive proofs we can employ much more severe notion of feasibility than just "poly-time verifiable" and require a proof to be presented in such a format that its validity can be verified by checking a constant number of (randomly chosen) places in the proof. This is the celebrated *PCP*[2] *theorem* which is in fact extremely tightly connected with interactive proof systems.

In another unexpected turn, the PCP theorem and its many variants became the major tool in analyzing the complexity of finding approximate solutions to combinatorial optimization problems. This is one of the most practical areas of Theoretical Computer Science that, prior to the PCP breakthrough, did not have anything to do with proofs, and in general was not too successful at solving its major problems. We will sample several typical applications in this area.

Next, we we will link this latter topic with propositional proof complexity by considering the optimization problem of finding the best proof for a given tautology in a given propositional proof system. This naturally leads to the important concept of *automatizability* of such systems. We will mention one tight connection between Feasible Interpolation and automatizability, and give one example of a proof system for which they (apparently) differ.

Finally, we will return to the question of feasible provability of the $\mathbf{P} \neq \mathbf{NP}$ question previously considered by us in the context of the propositional proof complexity. It was also studied in the framework of so-called *Natural Proofs,* where "proofs" are defined this time by a set of properties (of computational nature) shared by all known arguments. We will show the main theorem of this mini-theory, which is exactly the result of a kind we are still missing in the propositional proof complexity.

3 Historical Remarks and Literature for Further Reading

Some of the topics above (especially those from the first, "classical" part) appeared in my ICALP lecture 8 years ago in much more elaborated and systematic way, and the extended abstract of that talk [38] contains some additional literature.

Bounded Arithmetic was apparently considered for the first time by Parikh in [31], and was studied by Paris and Wilkie in the 1980s (see e.g. [32]). Systematically this

[2] for Probabilistically Checkable Proofs

subject was treated in Buss's book [15] which still remains a very good source for a quick introduction to it. Other choices include the monograph [25] and the last section of [23].

The first non-trivial lower bound in propositional proof complexity is due to Tseitin [45] which well preceded the general theory developed by Cook and Reckhow in [17]. Feasible Interpolation Theorem evolved from a sequence of papers [24,37,13,26], and its elegant proof sketched in this talk is due to Pudlák [33]. The limitations of Feasible Interpolation for stronger proof systems were established in [29,14,12].

The proper formalization of the non-uniform version of $\mathbf{P} \neq \mathbf{NP}$ was proposed by Razborov in [36], and it was also stressed there that the proofs of (apparently) all known partial results toward this goal become feasible in this framework. The approach based upon pseudo-random generators was proposed by Alekhnovich, Ben-Sasson, Razborov, Wigderson [1] and Krajíček [27]; the first paper also contained specific suggestions as to the use of Nisan-Wigderson generators. Partial results in that direction were proved in [1,3,35,40,28,39], and the introduction in [39] also contains an extended summary of the whole approach, and of the speaker's view of the subject.

There are several good surveys on propositional proof complexity as a whole, see e.g. [46,25,38,11,34].

Interactive proofs were introduced by Goldwasser, Micali, Rackoff [21] and Babai [8], and the protocol for the GRAPH NON-ISOMORPHISM was given by Goldwasser, Micali, Wigderson [22]. The result that all of PSPACE has interactive proofs (which is much stronger than its partial case mentioned in the talk) is due to Lund, Fortnow, Karloff, Nisan [30] and Shamir [44].

The original form of the PCP theorem evolved from [10,18], and was proved in the papers by Arora, Safra [6] and Arora, Lund, Motwani, Sudan, Szegedy [5]. The connection to the complexity of approximation was understood already in those early papers (in fact, it was one of their primary motivations).

There are many good surveys on interactive proofs, PCP and hardness of approximation, see e.g. [9,19,4]. We would also like to mention the books [20,7], as well as the unique on-line compendium `http://www.nada.kth.se/~viggo/problemlist/compendium.html` compiling known results on the complexity of approximation.

The concept of automatizability was introduced by Bonet, Pitassi, Raz [14], and the remark that automatizability implies feasible interpolation is due to Impagliazzo (unpublished). Alekhnovich, Razborov [2] proved (modulo strong complexity assumptions) that Resolution (which does allow Feasible Interpolation) is not automatizable.

Natural proofs were introduced by Razborov, Rudich [41], and [42] gave some unexpected applications in quite a different area. This theory was further developed by Rudich [43].

References

[1] M. Alekhnovich, E. Ben-Sasson, A. Razborov, and A. Wigderson. Pseudorandom generators in propositional complexity. In *Proceedings of the* 41*st IEEE FOCS,* pages 43–53, 2000. Journal version to appear in *SIAM Journal on Computing.*

[2] M. Alekhnovich and A. Razborov. Resolution is not automatizable unless $W[P]$ is tractable. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science,* pages 210–219, 2001.

[3] M. Alekhnovich and A. Razborov. Lower bounds for the polynomial calculus: non-binomial case. *Proceedings of the Steklov Institute of Mathematics,* 242:18–35, 2003.

[4] S. Arora. The approximability of NP-hard problems. In *Proceedings of the 30th ACM Symposium on the Theory of Computing,* pages 337–348, 1998.

[5] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science,* pages 13–22,1992.

[6] S. Arora and M. Safra. Probabilistic checking of proofs: a new characterization of np. *Journal of the ACM*, 45(1):70–122, 1998.

[7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation. Combinatorial optimization problems and their approximability properties.* Springer-Verlag, 1999.

[8] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on the Theory of Computing,* pages 421–429, 1985.

[9] L. Babai. Transparent proofs and limits to approximations. In *Proceedings of the First European Congress of Mathematics, Vol. I,* pages 31–91. Birkhauser, 1994.

[10] L. Babai, L. Fortnow, C. Lund, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd ACM Symposium on the Theory of Computing,* pages 21–31, 1991.

[11] P. Beame and T. Pitassi. Propositional proof complexity: Past, present and future. Technical Report TR98-067, Electronic Colloquium on Computational Complexity, 1998. Available at ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/1998/TR98-067/index.html.

[12] M. Bonet, C. Domingo, R. Gavaldá, A. Maciel, and T. Pitassi. Non-automatizability of bounded-depth Frege proofs. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity,* pages 15–23, 1999.

[13] M. Bonet, T. Pitassi, and R. Raz. Lower bounds for cutting planes proofs with small coefficients. *Journal of Symbolic Logic,* 62(3):708–728, 1997.

[14] M. Bonet, T. Pitassi, and R. Raz. On interpolation and automatization for Frege systems. *SIAM Journal on Computing,* 29(6):1939–1967, 2000.

[15] S. R. Buss. *Bounded Arithmetic.* Bibliopolis, Napoli, 1986.

[16] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing,* pages 151–158, 1971.

[17] S. A. Cook and A. R. Reckhow. The relative efficiency of propositional proof systems. Journal of Symbolic Logic, 44(1):36–50, 1979.

[18] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM,* 43(2):268–292, 1996.

[19] O. Goldreich. Probabilistic proof systems. In *Proceedings of the International Congress of Mathematicians (Zurich, 1994),* pages 1395–1406. Birkhauser, 1995.

[20] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness, Algorithms and Combinatorics, Vol. 17.* Springer-Verlag, 1998.

[21] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing,* 18(1):186–208, 1985.

[22] S. Goldwasser, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity, and a methodology of cryptographic protocol design. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science,* pages 39–48, 1986.

[23] P. Hájek and P. Pudlák. *Metamathematics of First-Order Arithmetic.* Springer-Verlag, 1993.

[24] **J. Krajíček.** Lower bounds to the size of constant-depth propositional proofs. *Journal of Symbolic Logic,* 59(1):73–86, 1994.

[25] **J. Krajíček.** *Bounded arithmetic, propositional logic and complexity theory.* Cambridge University Press, 1995.

[26] **J. Krajíček.** Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. *Journal of Symbolic Logic,* 62(2):457–486, 1997.

[27] **J. Krajíček.** On the weak pigeonhole principle. *Fundamenta Mathematicae,* 170(1-3):123–140, 2001.

[28] **J. Krajíček.** Dual weak pigeonhole principle, pseudo-surjective functions, and provability of circuit lower bounds. *Journal of Symbolic Logic,* 69(1):265–286, 2004.

[29] **J. Krajíček** and P. Pudlák. Some consequences of cryptographical conjectures for S_2^1 and *EF*. *Information and Computation,* 142:82–94, 1998.

[30] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM,* 39(4):859–868, 1992.

[31] R. J. Parikh. Existence and feasibility in arithmetic. *Journal of Symbolic Logic,* 36:494–508, 1971.

[32] J. Paris and A. Wilkie. Counting problems in bounded arithmetic. In *Methods in Mathematical Logic, Lecture Notes in Mathematics 1130,* pages 317–340. Springer-Verlag, 1985.

[33] P. Pudlák. Lower bounds for resolution and cutting planes proofs and monotone computations. *Journal of Symbolic Logic,* 62(3):981–998, 1997.

[34] P. Pudlák. The lengths of proofs. In S. Buss, editor, *Handbook of Proof Theory,* pages 547–637. Elsevier, 1998.

[35] R. Raz. Resolution lower bounds for the weak pigeonhole principle. *Journal of the ACM,* 51(2):115–138, 2004.

[36] A. Razborov. Bounded Arithmetic and lower bounds in Boolean complexity. In P. Clote and J. Remmel, editors, *Feasible Mathematics II. Progress in Computer Science and Applied Logic, vol.* 13, pages 344–386. Birkhaüser, 1995.

[37] A. Razborov. Unprovability of lower bounds on the circuit size in certain fragments of bounded arithmetic. *Izvestiya of the RAN,* 59(1):201–224, 1995.

[38] A. Razborov. Lower bounds for propositional proofs and independence results in Bounded Arithmetic. In F. M. auf der Heide and B. Monien, editors, *Proceedings of the* 23*rd ICALP, Lecture Notes in Computer Science,* 1099, pages 48–62, New York/Berlin, 1996. Springer-Verlag.

[39] A. Razborov. Pseudorandom generators hard for k-**DNF** resolution and polynomial calculus resolution. Manuscript available at http://www.genesis.mi.ras.ru/~razborov, 2002.

[40] A. Razborov. Resolution lower bounds for perfect matching principles. In *Proceedings of the* 17*th IEEE Conference on Computational Complexity,* pages 29–38, 2002.

[41] A. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Sciences,* 55(1):24–35, 1997.

[42] K. Regan, D. Sivakumar, and J. Cai. Pseudorandom generators, measure theory, and natural proofs. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science,* pages 26–35, 1995.

[43] S. Rudich. Super-bits, demi-bits, and NP/qpoly-natural proofs. In *Proceedings of the International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM 97), Lecture Notes in Computer Science,* 1269, pages 85–93, New York/Berlin, 1997. Springer-Verlag.

[44] A. Shamir. *IP = PSPACE. Journal of the ACM,* 39(4):869–877, 1992.

[45] G. C. Tseitin. On the complexity of derivations in propositional calculus. In *Studies in constructive mathematics and mathematical logic, Part II.* Consultants Bureau, New-York-London, 1968.

[46] A. Urquhart. The complexity of propositional proofs. *Bulletin of Symbolic Logic,* 1:425–467, 1995.

Grammar Compression, LZ-Encodings, and String Algorithms with Implicit Input

Wojciech Rytter*

Instytut Informatyki, Warsaw University, Poland, and
Department of Computer Science, NJIT, USA
rytter@oak.njit.edu

Abstract. We survey several complexity issues related to algorithmic problems on words given in an *implicit* way: by a *grammar, LZ-encoding* or as a minimal solution of a *word equation.* We discuss the relation between two implicit representations, the role of word compression in solvability of word equations and compressed language recognition problems. The grammar compression is more convenient than *LZ*-encoding, its size differs from that of *LZ*-encoding by at most logarithmic factor, the constructive proof is based on the concept similar to balanced trees.

1 Introduction

Algorithmics on compressed objects is a recently developed area of theoretical computer science. Its is motivated by the increase in the volume of data and the need to store and transmit masses of information in *compressed* form. The compressed information has to be quickly accessed and processed without explicit decompression. The main problem is how to deal efficiently with strings given implicitly.

We discuss three types of implicit representation: a context free grammar generating a single string, a Lempel-Ziv encoding and word equations describing a string as a minimal solution. The last type representation is the most complex, the best upper bound on the size of minimal solution is doubly exponential while it is believed that it is only singly exponential. The implicit representations of strings is the main tool in the best known algorithms for testing solvability of word equations.

Lempel-Ziv Encodings

Intuitively, LZ algorithm compresses the input word because it is able to discover some repeated subwords, see [8]. The Lempel-Ziv code defines a natural factorization of the encoded word into subwords which correspond to intervals in the code. The subwords are called *factors.* Assume that Σ is an underlying alphabet and let w be a string over Σ. The LZ-factorization of w is given by

* Supported by the grants ITR-CCR-0313219 and KBN 4T11C04425.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 15–27, 2004.

a decomposition: $w = f_1 \cdot f_2 \cdot \cdot f_k$, where $f_1 = w[1]$ and for each $2 \leq i \leq k$, f_i is the longest prefix of $f_i \ldots f_k$ which occurs in $f_1 \ldots f_{i-1}$ or a single symbol if there is no such nonempty prefix. We can identify each f_i with an interval $[p, q]$, such that $f_i = w[p \ldots q]$ and $q \leq |f_1 \ldots f_{i-1}|$. We identity LZ-factorization with $LZ(w)$. Its size if the number of factors.

Example 1. The LZ-factorization of the 7-th Fibonacci word Fib_7 is given by:

$$abaababaabaab = f_1\ f_2\ f_3\ f_4\ f_5\ f_6 = a\ b\ a\ aba\ baaba\ ab$$

Grammar Compression

Text compression based on context free grammars, or equivalently, on straight-line programs, has attracted much attention, see [5,20,24,25,26,38]. The grammars give a more structured type of compression and are more convenient for example in compressed pattern-matching, see [38]. In a grammar-based compression a single text w of length n is generated by a context-free grammar G. Computing exact size of the minimal grammar-based compression is known to be NP-complete. For simplicity assume that the grammars are in Chomsky normal form. The size of the grammar G, denoted by $|G|$, is the number of productions (rules), or equivalently the number of nonterminals of a grammar G in Chomsky normal form. Grammar compression is essentially equivalent to straight-line programs. A *grammar (straight-line program)* is a sequence:

$$X_1 = expr_1;\ X_2 = expr_2; \ldots;\ X_m = expr_m,$$

where X_i are nonterminals and $expr_i$ is a single terminal symbol, or $expr_i$ is of a form $X_j \cdot X_k$, for some $j, k < i$, where $\cdot$ denotes the concatenation of X_j and X_k. For each nonterminal X_i, denote by $val(X_i)$ the value of X_i, it is the string described by X_i. The string described by the whole straight-line program is $val(X_m)$. The size of the straight-line program is m.

Example 2.
Let us consider the following grammar G_7 which describes the *7th Fibonacci word* $Fib_7 = abaababaabaab$. We have $|G_7| = 7$. This is the smallest size grammar in Chomsky normal form for Fib_7.

$$X_7 = X_6 \cdot X_5;\ X_6 = X_5 \cdot X_4;\ X_5 = X_4 \cdot X_3;$$
$$X_4 = X_3 \cdot X_2;\ X_3 = X_2 \cdot X_1;\ X_2 = a;\ X_1 = b;$$

Word Equations

The problem of solving word equations is not well understood. Word equations can be used to define various properties of strings, e.g. general versions of pattern-matching with variables. Instead of dealing with very long solutions we can deal with their Lempel-Ziv encodings. Each minimal solution of a word equation is highly compressible (exponentially compressible for long solutions) in terms of Lempel-Ziv encoding. The best known algorithm for general word equations

works in *P-SPACE*. The polynomial space complexity is possible due to implicit representation of huge strings.

Compressed Membership Problems for Formal Languages

The problem consists in checking if an input word w is in a given language L, when we are only given a compressed representation of w. We present several results related to language recognition problems for compressed texts. These problems are solvable in polynomial time for uncompressed words and some of them become NP-hard for compressed words. The complexity depends on the type and description of the language L. In particular the membership problem is in polynomial-time for regular expressions. It is P-TIME complete for a fixed regular language. However it is NP-hard for semi-extended regular expressions and P-SPACE complete for context-free languages. The membership problem is NP-complete for unary regular expressions with compressed constants. Also for unary languages compressed recognition of context-free languages is NP-complete.

2 Relation Between Minimal Grammar Compression and LZ-Encodings

The problem of finding the smallest size grammar (or equivalently, straight line program) generating a given text is NP-complete. If A is a nonterminal of a grammar G then we sometimes identify A with the grammar G with the starting nonterminal replaced by A, all useless unreachable nonterminals being removed. In the parse tree for a grammar with the starting nonterminal A we can also sometimes informally identify A with the root of the parse tree.

For a grammar G generating w we define the parse-tree $Tree(G)$ of w as a derivation tree of w, in this tree we identify (conceptually) terminal symbols with their parents, in this way every internal node has exactly two sons. Define the partial parse-tree, denoted $PTree(G)$ as a maximal subtree of $Tree(G)$ such that for each internal node there is no node to the left having the same label. We define also the grammar factorization, denoted by G-factorization, of w, as a sequence of subwords generated by consecutive bottom nonterminals of $PTree(G)$. Alternatively we can define G-factorization as follows: w is scanned from left to right, each time taking as next G-factor the longest unscanned prefix which is generated by a single nonterminal which has already occurred to the left or a single letter if there is no such nonterminal. The factors of LZ- and G-factorizations are called LZ-factors and G-factors, respectively.

Example 3. The G_7-factorization for the 7-th Fibonacci strings is:

$$g_1\ g_2\ g_3\ g_4\ g_5\ g_6\ =\ a\ b\ a\ ab\ aba\ abaab$$

It can be shown that the number of factors in LZ-factorizations is not larger than the number of G-factors.

Theorem 1.
For each string w and its grammar-based compression G $|LZ(w)| \leq |G|$.

AVL- Grammars

AVL-grammars correspond naturally to *AVL*-trees. The first use of a different type balanced grammars has appeared in [17]. AVL-trees are usually used in the context of binary search trees, here we use them in the context of storing in the leaves the consecutive symbols of the input string w. The basic operation is the concatenation of sequences of leaves of two trees. We use the standard *AVL*-trees, for each node v the balance of v, denoted $bal(v)$ is the difference between the height of the left and right subtrees of the subtree of T rooted at v. T is *AVL-balanced* iff $|bal(v)| \leq 1$ for each node v. We say that a grammar G is *AVL-balanced* if *Tree*(G) is *AVL-balanced.* Denote by *height*(G) the height of *Tree*(G) and by *height*(A) the height of the parse tree with the root labelled by a nonterminal A. The following fact is a consequence of a similar fact for AVL-trees, see [22].

Lemma 1. *If the grammar G is AVL-balanced then* $height(G) = O(\log n)$.

In case of *AVL-balanced* grammars in each nonterminal A additional information about the balance of A is kept: *bal*(A) is the balance of the node corresponding to A in the tree *Tree*(G). We do not define the balance of nodes corresponding to terminal symbols, they are identified with their fathers: nonterminals generating single symbols. Such nonterminals are leaves of *Tree*(G), for each such nonterminal B we define *bal*(B) = 0.

Example 4. The grammar G_7 for the 7th Fibonacci word is *AVL-balanced.*

Lemma 2. *Assume A, B are two nonterminals of AVL-balanced grammars. Then we can construct in* $O(|height(A) - height(B)|)$ *time a AVL-balanced grammar G = Concat(A,B), where val(G) = val(A) · val(B), by adding only* $O(|height(A) - height(B)|)$ *nonterminals.*

Assume we have an *LZ*-factorization $f_1 f_2 \ldots f_k$ of w. We convert it into a grammar whose size increases by a logarithmic factor. Assume we have *LZ*-factorization $w = f_1 f_2 \ldots f_k$ and we have already constructed *good* (*AVL-balanced* and of size $O(i \cdot \log n)$) grammar G for the prefix $f_1 f_2 \ldots f_{i-1}$. If f_i is a terminal symbol generated by a nonterminal A then we set $G := Concat(G,A)$. Otherwise we locate the segment corresponding to f_i in the prefix $f_1 f_2 \ldots f_{i-1}$. Due to the fact that G is balanced we can find a logarithmic number of nonterminals $S_1, S_2, \ldots S_{t(i)}$ of G such that $f_i = val(S_1) \cdot val(S_2) \cdot \ldots val(S_{t(i)})$. The sequence $S_1, S_2, \ldots S_{t(i)}$ is called the *grammar decomposition* of the factor f_i.

We concatenate the parts of the grammar corresponding to this nonterminals with G, using the operation *Concat* mentioned in Lemma 2. Assume the first $|\Sigma|$ nonterminals corresponds to letters of the alphabet, so they exist at the beginning. We initialize G to the grammar generating the first symbol of w and containing all nonterminals for terminal symbols, they don't need to be initially

connected to the string symbol. If *LZ*-factorization is too large (exceeds $n/\log n$) then we neglect it and write a trivial grammar of size n generating a given string. Otherwise we have only $k \leq n \cdot \log n$ factors, they are processed from left to right. We perform the algorithm *Construct-Grammar.*

ALGORITHM. *Construct-Grammar*; {for string w of size n}
we are given LZ factorization $f_1 f_2 f_3 \ldots f_k$ of w
if $k > n/\log(n)$ **then return** trivial $O(n)$ size grammar
else for $i = 1$ **to** k **do**
(1) Let $S_1, S_2, \ldots, S_{t(i)}$ be grammar decomposition of f_i;
(2) $H := Concat(S_1, S_2, \ldots, S_{t(i)})$;
(3) $G := Concat(G, H)$;

Due to Lemma 2 we have $t(i) = O(\log n)$, so the number of two-arguments concatenations needed to implement single step (2) is $O(\log n)$, each of them adding $O(\log n)$ nonterminals. Steps (1) and (3) can be done in $O(\log n)$ time, since the height of the grammar is logarithmic. Hence the algorithm gives $O(\log^2(n)$-ratio approximation. At the cost of slightly more complicated implementation of step (2) $\log^2 n$-ratio can be improved to a $\log n$-ratio approximation. The key observation is that the sequence of heights of subtrees corresponding to segments S_i of next *LZ*-factar is *bitonic.* We can split this sequence into two subsequences: height-nondecreasing sequence $R_1, R_2, \ldots R_k$, called *right-sided,* and height-nonincreasing sequence $L_1, L_2, \ldots L_r$, called *left-sided.*

Lemma 3. *Assume $R_1, R_2, \ldots R_k$ is a right-sided sequence, and G_i is the AVL-grammar which results by concatenating $R_1, R_2, \ldots R_i$ from left-to-right. Then* $|height(R_i) - height(G_{i-1})| \leq \max\{(height(R_i) - height(R_{i-1}), 1\}$

Theorem 2. *We can construct in a $O(n \log |\Sigma|)$ time a $O(\log n)$-ratio approximation of a minimal grammar-based compression.*
Given LZ-factorization of length k we can construct a corresponding grammar of size $O(k \log n)$ in time $O(k \log n)$.

Proof. The next factor f_i is decomposed into segments $S_1, S_2, \ldots S_{t(i)}$. It is enough to show that we can create in $O(\log n)$ time an AVL-grammar for the concatenation of $S_1, S_2, \ldots, S_{t(i)}$ by adding only $O(\log n)$ nonterminals and productions to G, assuming that the grammars for $S_1, S_2, \ldots, S_{t(i)}$ are available.

The sequence $(S_1, S_2, \ldots, S_{t(i)}$ consists of a right-sided sequence and left-sided sequence. The grammars H', H'' corresponding to these sequences are computed (by adding logarithmically many nonterminals to G), due to Lemma 3. Then H', H'' are concatenated. Assume $R_1, R_2, \ldots R_k$ are right-sided subtrees. Then the total work and number of extra nonterminals needed to concatenate $R_1, R_2, \ldots R_k$ can be estimated as follows:

$$\sum_{i=2}^{k} |height(R_i) - height(G_{i-1})| \leq \sum_{i=2}^{k} \max\{height(R_i) - height(R_{i-1}),\ 1\}$$

$$\leq \sum_{i=2}^{k} (height(R_i) - height(R_{i-1})) + \sum_{i=2}^{k} 1 \leq height(R_k) + k = O(\log n).$$

The same applies to the left-sided sequence in a symmetric way. Altogether processing each factor f_i enlarges the grammar by an $O(\log n)$ additive factor and needs $O(\log n)$ time. To get $\log n$-ratio we consider only the case when the number k of factors is $O(n/\log n)$. *LZ*-factorization is computed in $O(n \log |\Sigma|)$ time using suffix trees, ($O(n)$ time for integer alphabets, see [19,11])).

There is possible a rather cosmetic improvement of the approximation ratio. Let g be the size of the minimal grammar-based compression and assume we have a greedy LZ-factorization of a string w of size n into s factors, the number s is also a lower bound on g. The improvement is a direct application of a method from the paper on compressed matching of Farach and Thorup [10], (In their notation $n = U$, $g = n$). In [10] they improved a starting factor $\log n$ to $\log(n/g)$ by introducing new cut-points and refining factorization. Exactly in the same way $\log n$ can be improved to get $\log(n/g)$.

Theorem 3. [6,37]
We can construct in polynomial time $O(\log(n/g))$-ratio approximation of a minimal grammar compression, where g is the size of the minimal grammar based compression of a given string of length n.

3 String Compression and Word Equations

Word equations are used to describe properties and relations of words, e.g. pattern-matching with variables, imprimitiveness, periodicity, conjugation, [18].

Let Σ be an alphabet of constants and Θ be an alphabet of variables. We assume that these alphabets are disjoint. A word equation E is a pair of words $(u, v) \in (\Sigma \cup \Theta)^* \times (\Sigma \cup \Theta)^*$ usually denoted by $u = v$. The *size* of an equation is the sum of lengths of u and v. A *solution* of a word equation $u = v$ is a morphism $h : (\Sigma \cup \Theta)^* \to \Sigma^*$ such that $h(a) = a$, for $a \in \Sigma$, and $h(u) = h(v)$. For example assume we have the equation

$$abx_1x_2x_2x_3x_3x_4x_4x_5 = x_1x_2x_3x_4x_5x_6,$$

and the length of x_i's are consecutive Fibonacci numbers. Then the solution is $h(x_i) = Fib_i$.

It is known that the solvability problem for word equations is in *P-SPACE* and is *N P*-hard, even if we consider (short) solutions with the length bounded by a linear function and the right side of equations contains no variables, see [4]. The main open problem is to close the gap between *NP* and *P-SPACE,* and to show the following.

Conjecture A: the problem of solving word equations is in *NP*.

Assume n is the size of the equation and N is the minimal length of the solution (if one exists). It is generally believed that another conjecture is true (at least no counterexample is known):

Conjecture B: N is at most singly exponential w.r.t. n.

A motivation to consider compressed solutions follows from the following fact.

Lemma 4. [34]
If we have grammar-encoded values of the variables then we can verify the word equation in polynomial time with respect to the size of the equation and the total size of the encodings.

Assume $h(u) = h(v) = \mathcal{T}$ is a solution of a given word equation E. A *cut* in $\mathcal{T}$ is a border of a variable or a constant in $\mathcal{T}$. We say that a subword w of $\mathcal{T}$ *overlaps* a cut γ iff an occurrence of w extends to the left and right of γ or γ is a border of an occurrence.

Lemma 5 (overlap lemma). [36]
Assume $\mathcal{T}$ is the minimal length solution of the equation E. Then each subword of $\mathcal{T}$ has an occurrence which overlaps at least one cut in $\mathcal{T}$.

The overlap lemma is crucial in proving the following fact.

Theorem 4. [36]
Assume N is the size of minimal solution of a word equation of size n. Then each solution of size N can be LZ-compressed to a string of size $O(n^2 \log^2(N)(\log n + \log\log N))$.

As a direct consequence we have:

Corollary 1. Conjecture B implies conjecture A.

Proof.
If N is exponential then the compressed version of the solution is of a polynomial size. The algorithm below solves the problem in nondeterministic polynomial time. The first step works in nondeterministic polynomial time, the second one works in deterministic polynomial time due to Lemma 4.

ALGORITHM. *Solving_by_LZ-Encoding* ;
guess LZ-encoded solution of size
$O(n^2 \log^2 N(\log n + \log\log N))$;
verify its correctness using the polynomial
time deterministic algorithm from Lemma 4.

Using more complicated algorithm it can be shown the following:

Theorem 5. *Assume the length of all variables are given in binary by a function f. Then we can test solvability in polynomial time, and produce polynomial-size compression of the lexicographically first solution (if there is any).*

Compressed Proofs of Solvability of Word Equations

The periodicity index for a given string x is the maximal k such that u^k is a subword of x, for a nonempty u.

Example 5.
For example index of periodicity of *abbababababababbaba* is 5 since we can write:

$$abbababababababbaba = abb(ab)^5baba;$$

Denote by $index\_per(n)$ the maximal index of periodicity of minimal length solutions of word equations of length n.

Theorem 6 (periodicity index lemma). [23]
$index\_per \leq 2^{cn}$ *for a constant* c.

It has been shown by W. Plandowski that the solvability of word equations is in P-SPACE, this algorithm is the milestone achievement is this area. The algorithm consists of nondeterministically finding a (compressed) compressed syntactic derivation of the equation. All equations in the derivation have lengths bounded by $p(n)$, where $p(n)$ is a fixed polynomial (the same for all equations) and n is the size of the original equation.

Assume Γ is the set of additional variables called pseudo-constants, and Σ is the set of original constants. Let $\mathcal{X}$ be the set of original variables, assume $\mathcal{X} \cap \Gamma = \emptyset$.

An exponential expression is a compressed representation of a word in terms of concatenation and exponentiation, e.g.

$$a^{237}bac^{1024}cdb^{23}$$

with the main invariant: all exponents are singly exponential w.r.t. n, hence can be stored in P-SPACE.

Each step in the compressed syntactic derivation should be accompanied by all possible reductions of the exponential expressions, to guarantee that their height is at most one. This can be done nondeterministically, guessing the final form and testing if the initial expression and the required one are equivalent. The *compressed syntactic derivation* consists in performing locally in one step one of the following syntactic operations :

1. Replace **each** occurrence of a selected pseudo-constant by an exponential expressions of size $O(n^3)$ over the alphabet $\Gamma \cup \Sigma$;
2. Replace a subword α over the alphabet $\Gamma \cup \Sigma$ in the actual equation by a variable $X \in \mathcal{X}$. For the same variable α should be the same in one iteration.

3. Replace a fragment $X \cdot R$, where ($X \in \mathcal{X}$ and $R \in (\Gamma \cup \Sigma)^*$), by X, for an original variable X, this should be done for **all** occurrences of X in the actual equation

We start with $w = w$ and end with the original equation preserving the invariant: the actual equation is solvable We show a *compressed syntactic derivation* of the following equation (the variables are written with capital letters)

$$YYXbaaaaZ = ZYYXaaaab$$

The compressed syntactic derivation provides a compressed proof that this equation is solvable.

$$w = w \overset{w \Rightarrow uvuuv}{\longrightarrow} uvuuv = uvuuv \overset{Z \Leftarrow uv}{\longrightarrow} uvuZ = Zuuv$$

$$\overset{v \Rightarrow ce}{\longrightarrow} uceuZ = Zuuce \overset{u \Rightarrow ccd}{\longrightarrow} ccdceccdZ = Zccdccdce \overset{Y \Leftarrow c,\ X \Leftarrow c}{\longrightarrow}$$

$$YcdYecXdZ = ZYcdYcdXe \overset{c \Rightarrow ef}{\longrightarrow} YefdYeefXdZ = ZYefdYefdXe \overset{Y \Leftarrow Ye}{\longrightarrow}$$

$$YfdYefXdZ = ZYfdYfdXe \overset{e \Rightarrow fb}{\longrightarrow} YfdYfbfXdZ = ZYfdYfdXfb$$

$$\overset{Y \Leftarrow Yf}{\longrightarrow} YdYbfXdZ = ZYdYdXfb$$

$$\overset{d \Rightarrow bf}{\longrightarrow} YbfYbfXbfZ = ZYbfYbfXfb \overset{Y := \Leftarrow Ybf}{\longrightarrow} YYXbfZ = ZYYXfb$$

$$\overset{f \Rightarrow a^4}{\longrightarrow} YYXbaaaaZ = ZYYXaaaab$$

Another possibility to check solvability of the equation is to guess the following values for the variables (then test the example word equation).

$$x = aaaabaaaa;\ y = aaaabaaaaaaaabaaaabaaaa$$
$$z = aaaabaaaaaaaabaaaabaaaaaaaabaaaaaaaab$$

After substituting these values into the equation the length of each of its sides becomes 97.

Using the overlap-lemma and the periodicty-index lemma the following theorem has been shown. Observe that the only nontrivial part here is the "⇒" implication in point (1).

Theorem 7. [33]
(1) *The word equation has a solution iff it has a syntactic derivation;*
(2) *The solvability problem for word equations is in P-SPACE.*

Let A be a deterministic finite automaton. The pseudo-constants and variables which appear in the syntactic derivation can be augmented by additional information: transition tables which say to which state we move after reading a subword (consisting of final constants) corresponding to the pseudo-constant or the actual value of the variable after starting in each possible state. In this way it can be shown the following:

Theorem 8. *Solvability of word equations with regular constraints (values of variables should be in given regular sets) is P-SPACE complete.*

4 Membership of Compressed Strings in Formal Languages

For a formal language L and a compressed word x, given implicitly by a grammar, we are to check if $x \in L$. The compressed string matching can be treated as a language recognition problem:

check if $w \in \{x\#y \ : \ x$ is a subword of y, and x, y do not contain $\#\}$.

Compressed Recognition Problems for Regular Expressions

We consider three classes of regular expressions as descriptions of regular languages:

1. (standard) regular expressions (using uncompressed constants and operations $\cup$, *,$\cdot$);
2. regular expressions with compressed constants (constants given in compressed forms);
3. semi-extended regular expressions (using additionally the operator $\cap$ and only uncompressed constants)

The size of the expression is a part of the input size.

Theorem 9.
(a) *We can decide for compressed words the membership in a language described by given regular expression W in $O(n \cdot m^3)$ time, where $m = |W|$.*
(b) *We can decide for compressed words the membership in a language described by given deterministic automaton M in $O(n \cdot m)$ time, where m is the number of states of M.*

The following problem is useful to show *NP*-hardness of several compressed recognition problems.

SUBSET SUM Problem:
Input instance: Finite set $A = \{a_1, a_2, \ldots, a_n\}$ of integers and an integer K. The size of the input is the number of bits needed for the binary representation of numbers in A and K.
Question: Is there a subset $A' \subseteq A$ such that the sum of the elements in A' is exactly K?

Lemma 6. [12] *The problem SUBSET SUM is NP-complete.*

We show an application of the SUBSET SUM problem in the proof of the following fact.

Theorem 10. [35] *The problem of checking membership of a compressed unary word in a language described by a* star-free *regular expression with compressed constants is NP-complete.*

Proof.
The proof of *NP*-hardness is a reduction from the SUBSET SUM problem. We can construct easily a straight-line program such that $value(X_i) = d^{a_i}$ and $w = d^K$. Then the SUBSET SUM problem is reduced to the membership:

$$w \in (value(X_1) \cup \varepsilon) \cdot (value(X_2) \cup \varepsilon) \cdots (value(X_n) \cup \varepsilon).$$

The empty string ε can be easily eliminated in the following way. We replace each ε by a single symbol d and each number a_i by $(a_i + 1)$. Then we check whether d^{n+K} is generated by the obtained expression.

The problem is in *NP* since expressions are *star-free.* We can construct an equivalent nondeterministic finite automaton *A* and guess an accepting path. All paths are of polynomial length due to *star-free* condition. We can check in polynomial time if concatenation of constants on the path equals an input text *P.* This completes the proof.

Theorem 11. [35] *The problem of checking membership of a compressed unary word in a language described by a given regular expression with compressed constants is in NP.*

Theorem 12. [35] *The problem of checking membership of a compressed word in a language described by a semi-extended regular expression is NP-hard.*

Recently it has been shown that the membership problem for a fixed regular language is comlete in the class of deterministic polynomial time computations.

Theorem 13. [28] *There is a fixed finite automaton for which the compressed membership problem is P-COMPLETE.*

Compressed Membership Problem for Context-Free Languages

The compressed membership problem is more difficult than that for regular languages, though in the uncompressed setting both can be soved in deterministic polynomial time.

Theorem 14. [35] *The problem of checking membership of a compressed unary word in a given cfl is NP-complete.*

The compressed context-free memebership problem for general alphabets has surprisingly high complexity status.

Theorem 15. [28]
The compressed membership problem for context free languages is P-SPACE complete.

References

1. A. Amir, G. Benson and M. Farach, *Let sleeping files lie: pattern-matching in Z-compressed files,* in *SODA '94.*
2. A. Amir, G. Benson, *Efficient two dimensional compressed matching, Proc. of the 2nd IEEE Data Compression Conference* 279-288 (1992).
3. A. Amir, G. Benson and M. Farach, *Optimal two-dimensional compressed matching,* in *ICALP'94* pp.215-225.
4. Angluin D., Finding patterns common to a set of strings, *J.C.S.S.,* **21(1)**, 46-62, 1980.
5. A. Apostolico, S. Leonardi, Some theory and practice of greedy off-line textual substitution, DCC 1998, pp. 119-128
6. Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, April Rasala, Amit Sahai and Abhi Shelat, Approximating The Smallest Grammar: Kolmogorov Complexity in Natural Models, STOC 2002
7. Choffrut, C., and Karhumäki, J., Combinatorics of words, *in* G.Rozenberg and A.Salomaa (eds), *Handbook of Formal Languages,* Springer, 1997.
8. M. Crochemore, W. Rytter, *Jewels of stringology - Text algorithms,* World Scientific 2003
9. Eyono Obono, S., Goralcik, P., and Maksimenko, M., Efficient solving of the word equations in one variable, *in Proc. MFCS'94,* LNCS 841, Springer Verlag, 336-341, 1994.
10. Farah, M., Thorup M., String matching in Lempel-Ziv compressed strings, *STOC'95,* 703-712, 1995.
11. Farach, M., Optimal suffix tree construction with large alphabets, FOCS 1997.
12. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, New York (1979).
13. L. Gąsieniec, M. Karpiński, W. Plandowski and W. Rytter, *Efficient algorithms for compressed strings,* in proceedings of the SWAT'96, LNCS 1097, 392-403, 1996.
14. L. Gąsieniec, M. Karpiński, W. Plandowski and W. Rytter, *Randomized Efficient Algorithms for Compressed Strings: the finger-print approach.* in proceedings of the CPM'96, LNCS 1075, 39-49, 1996.
15. L.Gasieniec, W. Rytter, Almost optimal fully compressed LZW-matching, in *Data Compression Conference,* IEEE Computer Society 1999
16. L.Gasieniec, A.Gibbons, W. Rytter, The parallel complexity of pattern-searching in highly compressed texts, in *MFCS* 1999
17. M. Hirao, A. Shinohara, M. Takeda, S. Arikawa, Faster fully compressed pattern matching algorithm for balanced straight-line programs", Proc. of 7th International Symposium on String Processing and Information Retrieval (SPIRE2000), pp. 132-138. IEEE Computer Society, September 2000
18. Karhumäki J., Mignosi F., Plandowski W., The expressibility of languages and relations by word equations, *in ICALP'97,* LNCS 1256, 98-109, 1997.
19. J. Karkkainen, P. Sanders, Simple linear work suffix array construction, ICALP 2003
20. M. Karpinski, W. Rytter and A. Shinohara, *Pattern-matching for strings with short description,* in *Combinatorial Pattern Matching,* 1995.
21. T. Kida, Y. Shibara, M. Takeda, A. Shinohara, S. Arikawa, A unifying framework for compressed pattern matching, SPIRE'99
22. D. Knuth, *The Art of Computing, Vol. II: Seminumerical Algorithms. Second edition.* Addison-Wesley (1981).

23. Koscielski, A., and Pacholski, L., Complexity of Makanin's algorithm, *J. ACM* **43**(4), 670-684, 1996.
24. J. Kieffer, E. Yang, Grammar-based codes: a new class of universal lossless source codes, IEEE Trans, on Inf. Theory 46 (2000) pp. 737-754
25. J. K. Lanctot, Ming Li, En-hui Yang, Estimating DNA Sequence Entropy, SODA 2000
26. E. Lehman, A. Shelat, Approximation algorithms for grammar-based compression, SODA 2002
27. A. Lempel, J. Ziv, On the complexity of finite sequences, *IEEE Trans, on Inf. Theory,* 22, 75-81, 1976.
28. Markus Lohrey, Word problems on compressed words ICALP 2004
29. Makanin, G.S., The problem of solvability of equations in a free semigroup, *Mat. Sb.,* Vol. 103,(145), 147-233, 1977. English transl. in *Math. U.S.S.R.* Sb. Vol 32, 1977.
30. U. Manber, A text compression scheme that allows fast searching directly in the compressed file, *ACM Transactions on Information Systems,* 15(2), pp. 124-136, 1997
31. N. Markey, Ph. Schnoebelen, A P-Time complete matching problem for SLP-compressed words, IPL 2004
32. M. Miyazaki, A. Shinohara, M. Takeda, An improved pattern-matching algorithm for strings in terms of straight-line programs, Journal of Discrete Algorithms, Vol.1, pp. 187-204, 2000.
33. W. Plandowski, Satisfiability of word equations with constants is in P-Space, JACM 2004
34. W. Plandowski, Testing equivalence of morphisms on context-free languages, *Proceedings of the* 2^{nd} *Annual European Symposium on Algorithms* (ESA'94), LNCS 855, Springer-Verlag (1994), pp. 460–470.
35. W. Plandowski, W. Rytter, Complexity of compressed recognition of formal languages, in "*Jewels forever*", Springer Verlag 1999 (ed. J. Karhumaki)
36. W.Plandowski, W.Rytter, Applying Lempel-Ziv encodings to the solution of word equations, *ICALP* 1998
37. W. Rytter, Application of Lempel-Ziv Factorization to the Approximation of Grammar-Based Compression. TCS 1-3(299): 763-774 (2003), Preliminary version in Combinatorial Pattern Matching, June 2002,
38. W. Rytter, Compressed and fully compressed pattern-matching in one and two-dimensions, Proceedings of IEEE, November 2000, Volume 88, Number 11, pp. 1769-1778

Testing, Optimization, and Games

Mihalis Yannakakis

Department of Computer Science
Columbia University
New York, NY 10027
mihalis@cs.columbia.edu

Abstract. We discuss algorithmic problems arising in the testing of reactive systems, i.e. systems that interact with their environment. The goal is to design test sequences so that we can deduce desired information about the given system under test, such as whether it conforms to a given specification model, or whether it satisfies given requirement properties. Test generation can be approached from different points of view - as an optimization problem of minimizing cost and maximizing the effectiveness of the tests; as a game between tester and system under test; or as a learning problem. We touch on some of these aspects and related algorithmic questions.

1 Introduction

As systems become larger and more complex, there is a growing need for the development of effective automated methods to test them to ensure their correct functioning. In this talk we will give a sample of different aspects and problems arising in the testing of reactive systems, i.e. systems interacting with their environment. Such systems are conveniently modeled by various types of state machines. The goal is to design test sequences to exercise the system to determine whether it behaves correctly according to its specification, for example if its behavior conforms to a given reference model, or whether it satisfies given requirement properties.

Testing of software (and hardware) systems modeled by state machines (and by the related model of labeled transition systems) has been studied extensively for many decades and continues to be an active research area, due to its increasing importance. There is an extensive literature in this area (see eg. [7,22, 27]. Our purpose here is to highlight some of the different ways of looking at questions in testing, and give a sample of some of the issues and results.

One way to look at testing is as a game betwen a Tester and the System under test: the Tester is trying to find out if there is a fault in the system, while the System is trying to hide it. The question is whether the Tester has a winning strategy (an effective test) and how to design such a strategy. This is a game of incomplete information because the Tester does not have knowledge of the internals of the system and its evolution, but only gets incomplete information from the outputs that the system produces in response to the Tester's inputs.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 28–45, 2004.

We describe the framework in Section 2 and discuss from this point of view the conformance testing problem: Given a reference model (a finite state machine) that specifies how the system is supposed to behave, test the system to check whether it conforms to the model.

In Section 3 we discuss the problem of testing whether a System satisfies a given required property; this is called the black box checking problem - "black box" because we do not know the internals of the system. The "white box" version of this probleem is the well-known model checking problem: given a model (the complete description of a finite state machine) and a property, check whether the model satisfies the property [9]. We outline an approach to black box checking that combines algorithms from learning, model checking and conformance testing.

Testing can be viewed as an optimization problem: Use the minimum amount of testing to achieve a desired degree of fault coverage. In Section 4 we discuss optimal coverage problems of this type for ordinary finite state machines, as well as for compact represenations of machines using variables and hierarchy (extended and hierarchical finite state machines).

2 Testing and Games

Testing can be viewed as a game between two players, the Tester and the System – the IUT (Implementation Under Test). The setup is shown schematically in Figure 1. The two players communicate through two types of signals: inputs (from the Tester to the IUT) and outputs (from the IUT to the Tester). In each step, the Tester provides an input to the System; in response, the System provides an output to the Tester. It is assumed that each step takes a finite amount of time and its end is detectable by the Tester, so we can regard the absence of an explicit signal from the System as a special output symbol 'null'.

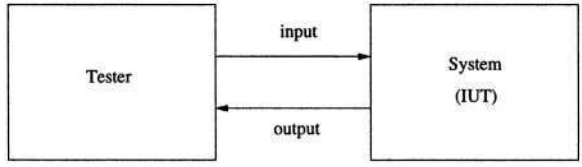


Fig. 1.

Here we will consider IUT's that are finite state machines with inputs and outputs. A deterministic FSM (or Mealy machine) M consists of a finite set Q of states, a finite input alphabet I, a finite output alphabet O, a transition function $\delta : Q \times I \mapsto O \times Q$ that maps each (state, input) pair to a (output, next state) pair; at the beginning of the test the FSM is in an initial state $s_0 \in Q$. Graphically, a FSM can be represented by its state transition diagram, a labeled directed graph that has one node for each state, and an edge for each state

transition labelled by the input/output pair a/b of the transition (see Fig. 2a). A nondeterministic FSM may have more than one transitions from the same state on the same input (i.e. δ maps $Q \times I$ to a subset of $O \times Q$), and can be represented also by its state diagram; it is assumed that the IUT is input-enabled, i.e. $\delta(s,a) \neq \emptyset$ for all $s \in Q, a \in I$. A probabilistic FSM is a nondeterministic FSM that has an associated probability with each transition so that the sum of the probabilities of all transitions from the same state with the same input is equal to 1; i.e., the transition function is a mapping $\nu : S \times I \times O \times S \mapsto [0,1]$, such that $\Sigma_{b \in O, t \in Q} \nu(s,a,b,t) = 1$ for all $s \in Q, a \in I$.

Before the game starts, the Tester has some a priori partial information about the IUT; i.e., he knows that the IUT belongs to a set U of possible IUT's. His objective is to interact with the IUT to discover some other desired missing information, i.e. to compute a function f from the set U of possible IUT's to a set V of possible decisions (verdicts). Each set U and function f determine a game. It is a game of incomplete partial information, since the Tester does not know the complete position of the game (the specific IUT M and the current state), but only gets partial information thorugh the outputs.

A deterministic *strategy* τ of the Tester is a mapping $\tau : (IO)^* \mapsto I \cup V$; that is, based on the past input-output history $h \in (IO)^*$, the Tester either decides on a verdict $\tau(h) \in V$ and terminates the game, or chooses the next input symbol $\tau(h) \in I$ which it will apply to the IUT. A randomized strategy τ of the Tester is a mapping $\tau : (IO)^* \times (I \cup V) \mapsto [0,1]$ such that $\Sigma_{a \in I \cup V} \tau(h,a) = 1$ for all $h \in (IO)^*$; the strategy specifies for each history h and input symbol or verdict a, the probability with which the Tester will decide verdict a (and terminate) or apply next the input a to the IUT.

A strategy as defined above is an *adaptive* strategy. A *preset* strategy is one in which the actions of the Tester do not depend on the output symbols, except only for the final verdict. That is, a preset deterministic strategy is a finite input sequence which the Tester applies to the IUT, followed by a final verdict that depends on the output sequence that is produced. A preset randomized strategy is a probability distribution on finite input sequences, followed again by the final verdict that depends on the output sequence.

The role of the System player is to choose first a IUT M in U (including an initial state) before the game starts, and then play the game with the Tester. Let us consider first the case of deterministic IUT M. In this case the System player has no more choices during the game: he just follows the transitions specified by the inputs received and produces the appropriate outputs. A play of the game (a *run*) is succesfull (or winning) for the Tester if it ends with the correct verdict $f(M)$. The interaction of a deterministic Tester strategy τ and a FSM M results in a unique run. Strategy τ is a *winning strategy* if it produces a succesfull run for every $M \in U$. For a probabilistic strategy τ and FSM M, there is a probability distribution on the resulting runs; let $P(\tau, M)$ be the probability that the resulting run is succesfull. Strategy τ wins with probability p if $P(\tau, M) \geq p$ for every $M \in U$.

For a given game (i.e. U and f) we want to determine if the Tester has a winning strategy, compute efficiently such a strategy if possible, and furthermore compute a "short" strategy, i.e. one that terminates in minimum time, and hence results is short tests. There are two notions of time complexity here: the *testing complexity,* i.e. the length of the test, and the *computational complexity,* the time it takes to compute the test (the strategy).

Example. In the *State Identification* (or Distinguishing sequence) problem, the Tester knows the complete state diagram of the IUT M, but does not know the initial state s_0; the objective of the test is to determine s_0. In this case, U consists of n FSM's (where n is the number of states of M), and $f(M)$ is the initial state of M. Each state diagram defines an instance of the game. In the instance shown in Figure 2a, the Tester has a winning strategy shown in Fig. 2b in the strategy tree form. The edges out of the circle nodes represent the actions of the Tester and are labeled by the corresponding input, the other edges correspond to the responses of the IUT, and each leaf is labelled with the verdict. In this particular example, the Tester has a winning strategy, but does not have a preset winning strategy. For general FSM, the Tester may have no winning strategy, even if the FSM is reduced (i.e. no two states are equivalent). As shown in [20], there is an efficient algorithm to determine whether there is a winning strategy for state identification for a FSM, and in that case there is one of length $O(n^2)$ which furthermore can be computed efficiently.

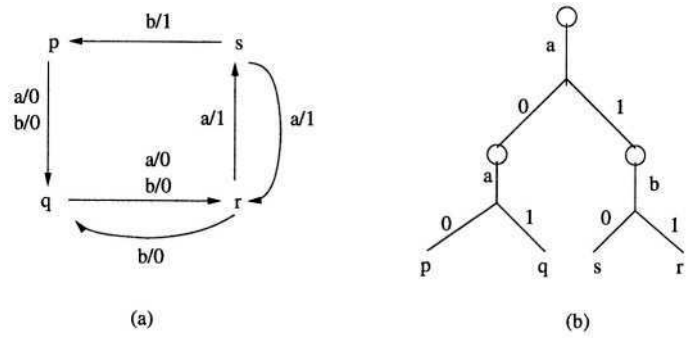


Fig. 2.

The most important problem in model-based testing is the *conformance testing* problem: Here there is a given specification FSM S, and the objective of the Tester is to determine whether the IUT 'conforms' to S, i.e. whether the response of the IUT for every possible input sequence is consistent with (could have been generated by) the FSM S. Thus, in this problem, $f(M) =$ 'pass', if M conforms to S, and $f(M) =$ 'fail' otherwise. Most of the works concern deterministic specification FSMs. For a deterministic specification FSM S, conformance means simply that the ITU M is equivalent to S. Note that, since every possible

input sequence has only one correct output sequence for a deterministic spec, adaptiveness does not matter in this case, i.e. it suffices to use a preset strategy.

Several assumptions are usually made on S and the possible ITU M for the problem to be solvable by a finite test: S is strongly connected, and it is reduced (this can be assumed without loss of generality). As for the ITU M, we assume that we know a bound on its number of states; most of the works actuallt assume that M has the same number of states as S, and the methods then extend in a straightforward way to larger number of states, though at an (unavoidable) cost. These assumptions on the ITU should be interpreted more as a yardstick for the extent of the fault detection capabilities of a test. If a test can handle all ITU's with the same number of states as S, then it means that it can detect all possible combinations of *output faults* (transitions producing the wrong output) and *next state faults* (transitions going to the wrong state); such a test is called a *checking sequence.*

There has been extensive work on the conformance testing problem since [23,16]. A survey can be found in [22]. We mention briefly the main results. One assumption that simplifies the problem, if it holds, is the existence of a *reliable reset,* i.e., a special input symbol, which when applied to the ITU, resets it back to the initial state. Suppose that S has n states, the ITU has at most m states, and the input alphabet has size p. As shown in [32,8], in the case of a reliable reset, if the ITU M does not have more states than S (i.e. $m \leq n$), then there is (preset) test of length $O(pn^3)$, and this is best possible (i.e., there are specification FSM's S, for which the shortest test is $\Omega(pn^3)$). If the bound m on the ITU size is larger than n, $m = n + \delta$, then the length is multiplied by a factor p^δ. The p^δ cost for the extra δ states is unavoidable for any specification FSM S.

If there is no reliable reset (i.e. no reset, or there is a reset but may not work properly in the ITU), then the problem is harder. First, it may not be always posible to verify the initial state s_0, i.e. there are FSM's S for which, even if the Tester knows that the ITU M is isomorphic to S, he has no winning strategy that allows him to check that the initial state of M is s_0. Furthermore, telling for a given S whether there is such a winning strategy is a PSPACE complete problem. However, it is always possible to test that the state transition diagram of M conforms to S, and ensure that the ITU (if it passes the test) is at an equivalent state at the end of the test. For $m \leq n$, there is an efficiently constructible randomized strategy that detects with high probability any specific faulty ITU within time $O(pn^3)$, and detects every faulty ITU's within time $O(pn^4 \log n)$ [33]. For most specification FSM's (in a well-defined sense), a test of length within a polylog factor of the number pn of transitions suffices. The existence of a randomized strategy of the above length that wins with high probability against all faulty ITU's, implies that there exists also a deterministic strategy (i.e. a checking sequence) of that length; however, we do not know how to construct it deterministically in polynomial time. These results extend again to ITU's with more states (with the same extra factor p^δ), and they also extend to partially specified specification FSM's.

2.1 Nondeterministic Systems

Suppose that the IUT is a nondeterministic FSM *M,* i.e. some states may have more than one transitions on the same input. Nondeterminism of the IUT reflects a situation where some of the transitions are not controlled by the Tester, and it may be due to a variety of reasons. In this case, the System player has choices during the game. A key issue, which is brought out explicitly in the game formulation, is the following: What should we assume the objective of the System player to be in making these choices? Is it trying to help the Tester arrive at the correct decision (for example, is the System trying to help the Tester find a fault), is it playing against the Tester (trying to hide the fault), or is it an indifferent player, eg. making random legal moves? For example, if the Tester wishes to test whether M conforms to a specification FSM S, and applies an input test sequence τ, and there are many possible responses of the ITU, what does it mean for the ITU to pass the test?

The appropriate role that we should assign to the System player in the context of testing is different than the appropriate role for the purposes of defining correctness of the System. For the system to be regarded as correct, it should behave as expected in all possible cases; i.e. in the context of correctness, the System player in effect cooperates to reveal the fault: if something can possibly go wrong, then we must assume it may go wrong in the field and call M faulty. On the other hand, this does not mean that it will go wrong during the test; if we apply τ in the test, it may well be the case that M will produce a consistent response, and we will not detect a fault. Thus, for the purposes of testing, the System player can be regarded as an adversarial player that is trying to convince the Tester that the IUT is correct.

Let S be a (nondeterministic) specification FSM, and M an (in general nondeterministic) implementation FSM. For an input sequence x, let $\lambda_S(x)$ (resp. $\lambda_M(x)$) be the set of possible output sequences that S (resp. M) produces on input x starting from its initial state. The ITU M is said to *conform* to the specification FSM S if $\lambda_M(x) \subseteq \lambda_S(x)$ for all input sequences $x \in I^*$. In other words, M does not conform to S if there is an input sequence x and *there exists* a path of M which produces an output sequence that could not have been produced by S (although M may have many other paths for the same input sequence x which produce consistent outputs with S).

Suppose that the Tester knows that the ITU is either the specification FSM S or the faulty FSM M that does not conform to S (the issues are similar if there are more possible faulty ITU's). In the testing context, taking the System as an adversarial player leads to the following definition. A (deterministic) testing strategy τ detects the faulty ITU M if it wins the game (decides "Fail") against every strategy of the System player (while of course it passes the ITU S). For example, in the case of a preset testing strategy, where τ is just an input sequence x, the test detects the faulty ITU if *every* path of M on x produces an inconsistent output, i.e., $\lambda_M(x) \cap \lambda_S(x) = \emptyset$. It is possible that an FSM M does not conform to 5, but the Tester does not have a winning strategy, and it is possible also that the Tester has a winning strategy, but not a preset one.

Algorithmically, determining whether a given FSM M conforms to another FSM S amounts to a language containment problem: We can regard S and M as nondeterministic automata over the alphabet $I \times O$ with all states accepting, and let $L(S)$, $L(M)$ be their corresponding languages. Then M conforms to S iff $L(M) \subseteq L(S)$, or equivalently $L(M) \cap \overline{L(S)} = \emptyset$. This is a PSPACE complete-problem. If S as an automaton over $I \times O$ is a deterministic automaton then the condition can be checked in polynomial time by complementing S and intersecting with M using the product construction. The FSM S is called in this case *input-output deterministic* (or *observable*), because even though there may be multiple transitions from a state on a given input, the output determines uniquely the next state.

Determining the existence of a test that detects a faulty machine M is a question of determining the winner in a two-player adversarial game with incomplete information [28]. Furthermore, any such game can be reduced to this particular testing game. The problem is PSPACE-complete for preset tester strategies and EXPTIME-complete for adaptive strategies [2] (this reference considers the equality of the languages of the two FSM's, but containment is similar). In the worst case, the length of the shortest winning strategy (test), if it exists, may be exponential in the size of the FSM's.

A more favorable (and not unreasonable) alternative is to treat the System as an indifferent player. This can be modeled by regarding the ITU as a probabilistic FSM, where all the transitions have nonzero probability. In this case we want to choose a strategy τ for the Tester that maximizes the probability of detecting a nonconforming ITU, and ideally we would like this probability to be 1, in which case we say that τ is an almost surely (a.s.) winning strategy.

If there is an input sequence x for which M has a path that produces an output sequence that is not in $\lambda_S(x)$, then there is a nonzero chance that M will follow that path on input x, and the Tester will detect the fault. That is, if M does not conform to S, then the Tester has nonzero probability of winning. Furthermore, if the ITU has a reliable reset, then the Tester can keep resetting the ITU and applying the same input sequence x repeatedly and drive the probability of detection as close to 1 as desired (athough this may not be the optimal strategy in terms of the efficiency of the test). Thus, if M does not conform to S and there is a reliable reset, then the Tester has an a.s. winning strategy (the probability of winning tends asymptotically to 1 as the length of the game goes to infinity).

Suppose now that there is no reliable reset. Then the Tester may not have an a.s. winning strategy; the existence of such a strategy does not depend on the exact values of the probabilities, it only depends on the transitions of M. The problem of selecting a Tester strategy that maximizes the probability of winning (fault detection) is now an instance of a game of incomplete information against a random player; alternatively, it can be viewed as a partially observable Markov decision process where the objective is to reach a target state. The algorithms are more complicated now than the adversarial game, although the complexity of the existence of a.s. winning strategies turns out to be the same (both for general

games against a random player, and for the testing game in particular): it is PSPACE-complete for preset strategies and EXPTIME-complete for adaptive strategies [2].

3 Property Testing, Model Checking, and Learning

In the previous section we discussed conformance testing of the system with respect to a given specification model. *Model checking* is a complementary approach, where one checks whether a given design model satisfies a required property expressed in some formalism, eg. a temporal logic property [9]. Model checking represents "white box" testing of the properties of a system; white box, in the sense that the system is completely known. Suppose that we have a system M with unknown structure (a black box), and we want to test it to check whether it satisfies a required linear temporal property P (i.e. P is a property of the executions of the system). This is called the *black box checking* problem [26]. Figure 3 shows schematically the relation between conformance testing, model checking, and black box checking.

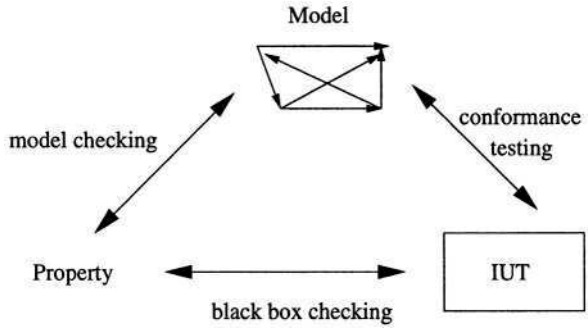


Fig. 3.

Suppose that the ITU is a (unknown) deterministic FSM with a known bound n on its number of states. Let us assume also that the ITU has a reliable reset. One approach is to first experiment with the system to identify exactly what the ITU FSM M is; this is called the Machine identification problem and is in general very expensive (there are machines which require a test of length p^n, where p is the size of the input alphabet [23]). Once we have identified M, we can apply a model checking algorithm to check whether M satisfies the required property P. The drawback of this approach is that it infers the whole structure of the ITU, which may be unnecessary, since all we want is to test if it satisfies a particular property. If the ITU does not satisfy the property, then it may well be possible to detect it (for example produce a counterexample) with more limited testing, without having to discover the whole structure of the ITU. A method

was proposed in [26] which combines conformance testing, model checking and a learning algorithm.

The method is shown schematically in Fig. 4. There is an algorithm due to Angluin [4] for learning FSM's with a reset (up to equivalence) in polynomial time, with the help of a "teacher". (The description in [4] is given in terms of automata, but the same algorithm essentially applies to Mealy machines.) The algorithm makes use of (i) queries (tests) to the FSM, i.e. the learner submits an input sequence to the FSM and receives an output sequence, and (ii) equivalence queries to the teacher, i.e. the learner submits a conjectured FSM S to the teacher, and the teacher either confirms that S is equivalent to M (in which case the algorithm terminates), or provides a counterexample to the equivalence.

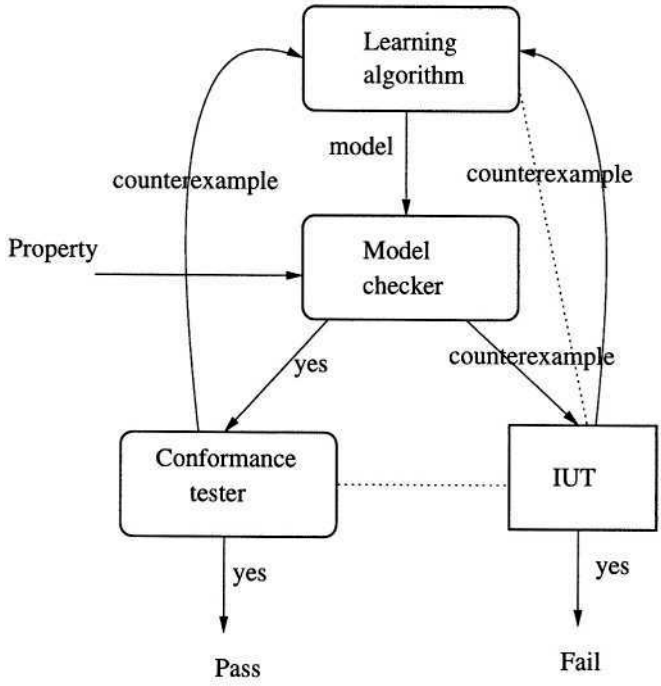


Fig. 4.

The black box checking algorithm combines Angluin's learning algorithm (call it A), the conformance testing algorithm with reset by Vassilevski-Chow (call it VC) [8,32], and any model checking algorithm (call it MC). First, call the learning algorithm A. Whenever A submits an input sequence query, we apply it on the system and return the output sequence to A. After some input queries, A comes up with a conjectured FSM S. Now we run the model checking algorithm on S. If S does not satisfy the property P, then MC returns a counterexample input sequence x. Apply x on the system and check if the output sequence

produced agrees with S. If it does, then the ITU does not satisfy the property P, and the input sequence x provides a counterexample. If the output produced by the ITU differs from S, then x is a counterexample to the equivalence of the ITU with the conjectured FSM S, so we provide it to the learning algorithm A and let it continue. If the model checker MC determines that S satisfies the property P, then we call the conformance testing algorithm VC with the specification FSM S. If VC decides that the ITU does not conform to S and returns a counterexample sequence x, then we provide x again to algorithm A, and let it continue. If VC decides that the ITU conforms to S, then we conclude that the ITU satisfies the property.

The information computed by the algorithms A and VC have a lot in common, so although we presented them as separate procedures, they can share the information and each can continue where the other left off to expedite the computation; we refer to [26] for the details and for the analysis of the running time. It is not necessary to know a priori a bound n on the number of states. We can run the algorithm until an error is found or we exceed the time that we are willing (or able) to spend; in the latter case, the guarrantees on the correctness of the ITU depend on the time spent. In the worst-case the algorithm is still exponential (this is unavoidable); but the expectation is that if there is an error then it will be found more quickly without constructing the whole FSM.

Black box checking represents checking of properties when we know essentially nothing about the structure of the system. Model checking represents the other extreme of white box checking, where we have complete knowledge of the structure. An intermediate case is when we have an approximate, but possibly inaccurate, model S. Then we can employ a similar *adaptive model checking* approach, starting with S, and then iteratively refine it or discover an error in the system [14].

In the absence of a reliable reset, there is a randomized learning algorithm due to Rivest and Schapire which learns an FSM with high probability in polynomial time, using again input queries (tests) to the FSM and equivalence queries to a teacher [29]. As mentioned in the last section, there is also a randomized conformance testing algorithm [33]. We can use the scheme in Fig. 4 and plug in these algorithms for the learning algorithm, and the conformance tester.

4 Testing and Optimization

Since we can always only afford a finite and limited amount of testing, it is important to select the tests in the best possible way to achieve the maximum possible "fault coverage" for the minimum cost of testing. To formalize this question, one needs to specify what is "fault coverage" - what types of faults are the tests supposed to detect - and how we measure the cost of the tests. We discuss in this section the optimization problem for various types of specification machines under the most basic fault coverage criterion, which is the coverage of transitions and states. There is a number of other coverage criteria that have been studied, which we will not discuss here. We will consider first plain FSM's,

and then two important types of succinct representations: extended FSM's and hierarchical FSM's.

Suppose that we have a specification machine S, that is an ordinary deterministic (strongly connected, minimal) FSM, and we want to test an implementation machine M for conformance with S. A checking sequence for S can detect all output and next state faults; although polynomial, however, it may be too long. A *transition tour* is a path that covers all the transitions. Computing the shortest transition tour is the classical Chinese Postman problem, which can be solved in polynomial time [10, 25].

A transition tour checks the correctness of the output of all the transitions of the implementation machine M, provided that the next states are all correct. If it is feasible, we can extend the power of the transition tour by applying some (perhaps limited) checking of the next states of the transitions. There is a number of such methods, all of which have the following structure: Each state s_i has an associated set X_i of (one or more) input sequences, which are meant to verify the state and distinguish it from all the other states. Testing a transition $(s_i, a/b, s_j)$ involves then applying a sequence that transfers the machine from its current state to s_i, then applying a (and expecting to observe b), and then applying a input sequence x_j from X_j (and checking that the expected output sequence is produced); the test sequence has to contain such a segment for each transition $(s_i, a/b, s_j)$ and each member x_j of X_j. There is a choice in the order in which these segments appear in the test sequence, and different orders may lead to different lengths. Finding the shortest test sequence that contains all the segments (as disjoint subsequences) is a Rural Chinese Postman problem. In general it is an NP-hard problem, but in certain cases it can be solved in polynomial time [1].

The problem can be expressed also as an Asymmetric Travelling Salesman path problem (ATSP) with triangle inequality: Construct an instance of the ATSP, which has one node for each segment and a node for the initial state s_0. Define the distance from a segment node u (or the initial node s_0) to another segment node v to be the length of the shortest path in S from the end state of the segment u (resp. from state s_0) to the start state of the segment v plus the length of the segment v. It is easy to see that the distances satisfy the triangle inequality. The length of the shortest test sequence of S that contains all the segments is equal to the length of the shortest path in the ATSP instance that starts at s_0 and visits all the nodes. The ASTP can be approximated within a $\log n$ factor [13]. It is open whether there is a constant factor approximation.

If there is a reset, then instead of considering the test as one continuous sequence, we usually consider it as composed of a set of sequences (i.e. as a test suite), where each sequence starts at the initial state of S and terminates with the resetting of the machine. In this case we usually seek a set of test sequences that cover all the regular (nonreset) transitions of the machine, and we measure the cost by two objectives: (1) the number of test sequences, and (2) the total length of the tests. Typically resetting the machine to run a new test sequence is more expensive than applying a new input symbol (especially in the context of system

tests run in a lab, which involve setting up the equipment for each test case), so the primary objective is objective 1, and the secondary is 2. We can compute in polynomial time a set of test sequences that cover all the (regular) transitions and minimizes the number of tests, while also minimizing the total length among all the minimum cardinality test sets. One can also assign different weights to the two objectives (number and length of tests) and minimize efficiently their linear combination, or compute the trade-off curve between the two objectives.

For some other coverage criteria, we can only optimize the number of tests, but not the length. One such case is the coverage of all the states. In this case, we can minimize the number of tests, but minimizing the total length is NP-hard. If the FSM S (without the reset) is acyclic, then we can minimize also the length, but for strongly connected FSM's, minimization of the length is equivalent to the ASTP on distance metrics induced by directed graphs. This case of the ATSP is also NP-hard and the best known approximation factor is $\log n$.

Another case with a similar behavior is when we want to test only a part of the specification, eg. we want to compute a set of test sequences that cover a given subset of transitions (or a subset of states). In this case also, we can minimize the number of tests, but minimizing the total length is NP-hard.

These methods apply not only to FSM's, but to other graphical models as well. For example, sets of communication scenarios involving a set of communicating processes can be represented by *message sequence chart graphs*. Briefly, a message sequence chart (MSC for short) is a partially ordered (finite) set of message exchanges between a (finite) set of processes. An MSC graph is a directed graph, each of whose nodes is associated with a particular MSC; the graph has a specified initial node. Every path in the graph sepresents a larger message sequence chart, obtained by concatenating the MSC's at the nodes of the path. Thus, the graph represents a (possibly infinite) set of message sequence charts. Message sequence charts are often used to capture typical (sunny day/rainy day) scenarios of communication protocols and distributed systems, and MSC graphs model the global behavior of the system, incorporating in a compact way many multiple possible scenarios. Lucent's uBET tool [17,31] employs this model (and more generally, a hierarchical MSC graph model - see section 4.2) to capture scenario requirements of communicating systems, and uses the abovementioned algorithm to generate tests that cover the edges of the MSC graph using the minimum number of tests with minimum total length.

4.1 Extended Finite State Machines

An extended finite state machine (EFSM) is a FSM extended with a finite set of variables (for example, Boolean variables, counters etc.) Every transition $s_i \rightarrow s_j$ has, in addition to an input a and output b, an associated predicate P on the variables (a guard), and an associated action A on the variables (a transformation on the values of the variables). The transition can take place if the EFSM is in state s_i, receives input a and the current values of the variables satisfy the predicate P; as the effect of the transition, the EFSM outputs symbol b, transforms the values of the variables according to the action A, and moves to state

s_j. The EFSM has typically a designated initial state and initial set of values for the variables. The use of auxhiliary variables is a convenient mechanism, and extended finite state machines are often used in the specification of communication protocols and systems, and underlie languages used for that purpose such as SDL.

An EFSM M is a succinct representation of a corresponding expanded state machine $\hat{M}$ which has one state for each combination of a state of M and vector of variable values (we'll call it a *configuration* to distinguish from the states of M). The initial configuration of $\hat{M}$ consists of the initial state of M and the initial values of the variables. We are interested only in the reachable portion of $\hat{M}$, the set of configurations that are reachable from the initial configuration. If the domains of the variables are finite (for example, Boolean), then $\hat{M}$ is a FSM.

The expanded FSM is usually very large due to the familiar state explosion problem, and hence it is typically not feasible to test all its transitions because the required test suites would be too large. (Note that executing a test sequence on the system is much more costly than simulating the EFSM model for the same sequence, hence the number of tests that can be executed is generally much smaller than the size of the expanded FSM model that we can generate.) A common relaxed coverage requirement then is to find tests that cover all the (reachable) transitions of the EFSM M (instead of the FSM $\hat{M}$). Note that in the expansion process, each transition of the EFSM M gives rise generally to many transitions of $\hat{M}$; covering any one of these transitions (rather than all of them) satisfies the relaxed requirement.

Instead of the full (reachable) expanded FSM $\hat{M}$, we can use its equivalent minimized FSM $\hat{M}_{min}$, which may be much smaller, or even it may be the case that the full expanded machine $\hat{M}$ is infinite but $\hat{M}_{min}$ is finite (this is the case for example for timed automata - EFSM's that include timers). The minimization of the machine preserves the optimality of the test sequences because every path in $\hat{M}$ corresponds to a path in $\hat{M}_{min}$ that covers the same transitions of M, and vice-versa. An efficient algorithm is given in [19] for the direct construction of the reachable minimized machine $\hat{M}_{min}$ from the EFSM M (without first expanding it), provided we have appropriate symbolic representations for sets of configurations.

In some cases it may not be possible to generate the whole $\hat{M}$ (minimized or not) due to its size, but can only generate a portion of it. It is important in this case that the generated portion includes, if possible, all reachable transitions of the EFSM M. Several papers have proposed the use of a model checking tool to generate tests. For each transition of the EFSM, we ask the model checker whether the transition is unreachable; the model checker (if successful) will either affirm that this is the case, or produce a counterexample, i.e. a path that reaches the transition. (Model checking can be used for more general coverage criteria that can be expressed in temporal logic, see eg. [18].) The generated set of paths covers all the reachable transitions, however, it is suboptimal, as there is a separate path for each transition. Instead of using directly these paths as the tests, we can put them all together, along with the other parts of the expanded

FSM explored in the search process, into a graph, which is a subgraph of $\hat{M}$, and then use this graph for test generation.

So, suppose that we have a graph G, with initial node u_0, which is a possibly partial, minimized version of the reachable expanded machine $\hat{M}$. Every edge of G corresponds to a transition of the EFSM M; associate a color with each transition of M, and color the edges of G accordingly. The problem of computing the minimum number of tests that cover all the transitions of M tranlates then to the following problem [21].

Colored Graph Covering (CGC). Given an edge-colored directed graph G with designated initial node u_0, find a minimum number of paths that cover all the colors.

Besides the edges, we could also color the nodes, and we could use sets of colors rather than single colors at each edge/node; the problem does not change substantially. We can also relate the problem to a generalization of the Asymmetric Traveling Salesman Problem, which we'll call Colored ATSP: Suppose that the nodes of the ATSP instance represent branches of the customers of the salesman, and that the salesman does not need to visit all the branches of all the customers, but rather has to visit at least one branch of each customer. The problem is again to find the shortest tour (or path) with this property.

The Colored graph covering problem is the core combinatorial problem for a variety of other types of test coverage criteria requiring the inclusion of representatives from specified classes of configurations or transitions (eg. [12]). For example instead of requiring that the tests include at least one occurence of a transition of the EFSM, we may want to cover at least one occurence of the transition for every possible combination of values for a specified subset of the variables.

The Colored Graph Covering problem (as well as the minimum transition coverage problem for EFSM's) is NP-hard. Furthermore, it embedds the Set Cover problem, hence one cannot approximate the problem within a factor better than $O(\log n)$ unless P=NP. However, we do not know of a $O(\log n)$ approximation algorithm. The adaptation of the greedy algorithm in this case, involves finding a path that covers the maximum number of (so far uncovered) colors, and repeating the process until all colors are covered.

Unfortunately, finding the "most colorful" path is also an NP-hard problem, in fact is is MAX SNP-hard, and thus cannot be approximated arbitrarily close to 1. We do not know if this problem has a constant factor approximation (such an algorithm would yield a $O(\log n)$ approximation to the CGC problem). The best we know so far is an algorithm that finds a path with $O(\sqrt{c})$ colors if the best path has c colors. If the maximum number of colors c on a path is bounded by a constant (the total number of colors may be arbitrarily large however) then we can in fact find an optimal path in essentially linear time in the size of the graph (and in polynomial time if $c = O(\log n)$) [21].

Some of these algorithms were implemented in a tool in Lucent (called Pythia) and applied to several real systems with very good results. In most cases, the first few paths cover a large number of colors, and the remaining

paths which constitute the bulk of the tests cover few additional colors each; the total number of tests are close to lower bounds for the instances.

4.2 Hierarchical State Machines

In order to specify large systems, it is often convenient to do this modularly in a hierarchical fashion. Hierarchical state machines are finite state machines whose states themselves can be other state machines. In more detail, a hierarchical finite state machine (HSM for short) is defined inductively as follows: a HSM of level 1 is simply an ordinary FSM; two of the states are distinguished as the initial (entry) and the final (exit) state (in general, there may be multiple entries and exits, but for simplicity we'll just restrict discussion here to a single entry and exit). Inductively an HSM of level $i > 1$ is a machine with a distinguished entry and exit state, whose other states are ordinary states or are "superstates" mapped to HSM's of smaller level $j < i$; the incoming and outgoing transitions of a supernstate v are connected to the entry and exit states of the corresponding HSM. We refer to the various machines used in the definition of a hierarchical state machine as its components (or modules). Not that many superstates (of the same or different components) can be mapped to the same lower level HSM; this is the benefit of defining a module once and reusing it many times in different contexts in a higher level module.

The ability to build hierarchical models is an essential ingredient of various specification formalisms such as statecharts [15], object-oriented software design methodologies and languages such as UML [6], and formalisms for scenario specifications for distributed systems (cf. the ITU standard Z.120 for high level (hierarchical) message sequence charts). It is available also in several tools. For example, Lucent's tool uBET can be used for the capture and testing of scenario requirements in the form of hierarchical message sequence charts (HMSC's) [31]; these are simply hierarchical graphs where the individual nodes are mapped to message sequence charts. Testmaster, a commercial testing tool that was marketed by Teradyne (then by Empirix) used a hierarchical extended finite state machine model [5].

A hierarchical state machine M is a succinct representation of an ordinary FSM $\hat{M}$ obtained by flattening the hierarchy, i.e., by recursively substituting the lower level machines for the superstates of the higher level machines. In general, the flattened machine can be exponentially larger than the input hierarchical machine, namely the size can be n^d where n is the number of states of a module (a component machine) in the hierarchical specification, and d is the number of levels. Note however that, unlike EFSM's, hierarchical FSM's can be analysed efficiently: we can verify temporal logic properties for them in polynomial time in the size of the succinct specification, without a need to expand the machine [3].

One way to generate tests for a hierarchical FSM M is to construct the expanded (ordinary) FSM $\hat{M}$ and then use one of the FSM methods on $\hat{M}$. For example, we can generate a minimum number of tests that covers all the transitions of $\hat{M}$ (this is the approach taken for example in uBET [31]). This is

a reasonable approach if the number of tests is not too large, and if the size of the expanded FSM is not too large, so that it can be constructed.

In general however, we may run into the familiar state explosion and test explosion problems: Covering every transition of $\hat{M}$ means that we cover every transiton of each module in every possible context, and this can result in general in a huge number of tests. As in the EFSM case, we can relax the transition coverage requirement to require only that each transition of each module is covered at least once; i.e., if several superstates map to the same module then we may choose to cover some transitions of the module at one superstate substitution, and some other transitions at another. This requirement will result in a much smaller covering set of tests. The minimum transition coverage problem for hierarchical FSMs is to compute a minimum number of tests which covers all the transitions of all the modules. This problem is NP-hard for 2 or more levels.

One approach to this problem is to reduce it to the Colored Graph Covering problem, by constructing the flattened FSM $\hat{M}$ and coloring each edge according to the transition of the module to which it corresponds. However, there are two problems with this approach: the expanded graph may be very large, and the guarranteed approximation ratio is not that great.

It is possible to do better, while avoiding the exponential penalty of the flattening. For each depth $d \geq 2$, and hierarchical FSM with d levels, we can generate in polynomial time a suite of tests that cover all transitions of all the modules, and the number of tests is within a factor d of the optimal. Furthermore, the approximation ratio is best possible: for every d, approximating the minimum size by a factor smaller than d is NP-hard [24]. The algorithm uses Linear Programming and network flow techniques. If the module hierarchy (i.e. the containment relationship between the modules) is a tree, then it is possible to guarrantee a somewhat better bound, namely, the rank of the tree; the rank is always upperbounded by the depth, although it may be much smaller depending on the tree (for example the rank of a path of any length is 1).

5 Conlusions

There has been extensive work in the area of testing over the years, both on the theoretical and the practical side. Here we discussed some of the different ways of looking at the problem and the relations with other areas (eg. optimization, games, learning, model checking), and we gave a sample of some of the relevant algorithmic problems and results.

References

1. A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours. *IEEE Trans. on Communication,* vol. 39, no. 11, pp. 1604-15, 1991.
2. R. Alur, C. Courcoubetis, and M. Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. *Proc. 27th Ann. ACM Symp. on Theory of Computing,* pp. 363-372, 1995.

3. R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Trans. on Progr. Languages and Systems,* 23(3):273–303, 2001.
4. D. Angluin. Learning regular sets from queries and counterexamples. *Inform. and Comp.,* 75, pp. 87-106, 1987.
5. L. Apfelbaum. Automated functional test generation. *Proc. IEEE Autotestcon Conference,* 1995.
6. G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide.* Addison Wesley, 1997.
7. E. Brinksma and J. Tretmans. Testing transition systems: An annotated bibliography. *Proc. MOVEP,* Springer Verlag LNCS 2067, pp. 187-195, 2001.
8. T. S. Chow, Testing software design modeled by finite-state machines. *IEEE Trans. on Software Engineering,* vol. SE-4, no. 3, pp. 178-87, 1978.
9. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking,* MIT Press 2000.
10. J. Edmonds and E.L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming,* vol. 5, pp. 88-124, 1973.
11. K. Etessami and M. Yannakakis. From Rule-Based to Automata-Based Testing. *Proc. IFIP Joint Intl. Conf. FORTE XIII-PSTV XX,* Kluwer, pp. 53-68, 2000.
12. G. Friedman, A. Hartman, K. Nagin, and T. Shiran. Projected state machine coverage for software testing. *Proc. ISSTA,* 2002.
13. A. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks* 12, 23-39, 1982.
14. A. Groce, D. Peled, and M. Yannakakis. Adaptive model checking. *Proc. TACAS,* Springer Verlag LNCS 2280, pp. 357-370, 2002.
15. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming,* 8:231-274, 1987.
16. F. C. Hennie. Fault detection experiments for sequential circuits. *Proc. 5th Annual Symp. Switching Circuit Theory and Logical Design,* pp. 95-110, 1964
17. G. Holzmann, D. A. Peled, and M. H. Redberg. Design tools for requirements engineering. *Bell Labs Technical Journal,* 2:86-95, 1997.
18. H. S. Hong, I. Lee, O. Sokolsky, and H. Ural. A temporal logic based theory of test coverage and generation. *Proc. TACAS,* pp. 327-341, 2002.
19. D. Lee and M. Yannakakis, On-line minimization of transition systems. *Proc. 24th Ann. ACM Symp. on Theory of Computing,* pp. 264-274, 1992.
20. D. Lee and M. Yannakakis, Testing finite state machines: state identification and verification. *IEEE Trans. on Computers,* vol. 43, no. 3, pp. 306-320, 1994.
21. D. Lee and M. Yannakakis. Optimization problems from feature testing of communication protocols. In *Proc. Intl. Conf. on Network Protocols,* pages 66–75, 1996.
22. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. *Proc. of IEEE,* 84(8):1090–1123, 1996.
23. E. F. Moore, Gedanken-experiments on sequential machines. *Automata Studies,* Annals of Mathematics Studies, Princeton University Press, no. 34, pp. 129-153, 1956.
24. D. Mosk-Aoyama and M. Yannakakis. Testing Hierarchical Systems via Edge Covering, manuscript. 2004
25. S. Naito and M. Tsunoyama, Fault detection for sequential machines by transitions tours. *Proc. IEEE Fault Tolerant Comput. Symp.,* IEEE Computer Society Press, pp. 238-43, 1981.
26. D. Peled, M. Vardi, and M. Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics,* 7(2), pp. 225-246, 2002.

27. A. Petrenko. Fault model-driven test derivation from finite state models: Annotated bibliography *Proc. MOVEP,* Springer Verlag LNCS 2067, pp. 196-205, 2001.
28. J. Reif. The complexity of two-player games with incomplete information. *J. Computer and System Sciences,* 29, pp. 274-301, 1984.
29. R. L. Rivest and R. E. Schapire, Inference of finite automata using homing sequences. *Proc. 21st Ann. Symp. on Theory of Computing,* pp.411-420, 1989.
30. J. Tretmans. Test generation with inputs, outputs amd repetitive quiescence. *Software - Concepts and Tools,* 17(3), pp. 103-120, 1996.
31. http:/cm.bell-labs.com/cm/cs/what/ubet.
32. M. P. Vasilevskii, Failure diagnosis of automata. *Kibernetika,* no. 4, pp. 98-108, 1973.
33. M. Yannakakis and D. Lee, Testing finite state machines: fault detection. *J. of Computer and System Sciences,* Vol. 50, No. 2, pp. 209-227, 1995.

Deciding Knowledge in Security Protocols Under Equational Theories

Martín Abadi[1*] and Véronique Cortier[2**]

[1] Computer Science Department, University of California at Santa Cruz, USA
[2] Loria, INRIA & CNRS, Nancy, France

Abstract. The analysis of security protocols requires precise formulations of the knowledge of protocol participants and attackers. In formal approaches, this knowledge is often treated in terms of message deducibility and indistinguishability relations. In this paper we study the decidability of these two relations. The messages in question may employ functions (encryption, decryption, etc.) axiomatized in an equational theory. Our main positive results say that, for a large and useful class of equational theories, deducibility and indistinguishability are both decidable in polynomial time.

1 Introduction

Understanding security protocols often requires reasoning about the knowledge of legitimate protocol participants and attackers. As a simple example, let us consider a protocol in which A sends to B a message that consists of a secret s encrypted under a prearranged shared key k. One may argue that, after processing this message, B knows s. More interestingly, one may also argue than an attacker with bounded computing power that does not know k but eavesdrops on the communications between A and B and sees the message does not learn s.

Accordingly, formal methods for the analysis of security protocols rely on definitions of the knowledge of protocol participants and attackers. In those methods, the knowledge of an attacker is used to determine what messages the attacker can send at each point in time—it can send only messages it knows. Moreover, security guarantees can be phrased in terms of the knowledge of the attacker. For example, a guarantee might be that, at the end of a protocol run, the attacker does not know a particular key, or that the attacker does not know whether a certain ciphertext contains the plaintext "true" or "false". For such applications, although the attacker is typically an active entity that can learn by conducting experiments, the definition of knowledge focuses on a particular point in a protocol execution.

Many formal definitions explain the knowledge of an attacker in terms of message deduction (e.g., [17,19,21,20]). Given a set of messages S and another message M, one asks whether M can be computed from S. The messages are represented by expressions,

* Martín Abadi's work was partly supported by the National Science Foundation under Grants CCR-0204162 and CCR-0208800.

** Véronique Cortier's work was partly supported by the RNTL project PROUVE-03V360 and the European project AVISPA IST-2001-39252.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 46–58, 2004.

and correspondingly the computations allowed are symbolic manipulations of those expressions. Intuitively these computations can rely on any step that an eavesdropper who has obtained the messages in S can perform on its own in order to derive *M*. For example, the eavesdropper can encrypt and decrypt using known keys, and it can extract parts of messages.

Despite its usefulness in proofs about protocol behaviors, the concept of message deduction does not always provide a sufficient account of knowledge, and it is worthwhile to consider alternatives. For instance, suppose that we are interested in a protocol that transmits an encrypted boolean value, possibly a different one in each run. We might like to express that this boolean value remains secret by saying that no attacker can learn it by eavesdropping on the protocol. On the other hand, it is unreasonable to say that an attacker cannot deduce the well-known boolean values "true" and "false". Instead, we may say that the attacker cannot distinguish an instance of the protocol with the value "true" from one with the value "false". More generally, we may say that two systems are equivalent when an attacker cannot distinguish them, and we may then express security guarantees as equivalences. The use of equivalences is common in computational approaches to cryptography (e.g., [16]), and it also figures prominently in several formal methods (e.g., [4,18,2]).

Two systems that output messages that an attacker can tell apart are obviously distinguishable. Conversely, in order to establish equivalences between systems, an important subtask is to establish equivalences between the messages that the systems generate (for example, between the encrypted boolean values). These equivalences may be called static equivalences, because they consider only the messages, not the dynamic processes that generate them. Bisimulation proof techniques can reduce process equivalences to static equivalences plus fairly standard bisimulation conditions [2] (see also [3,9]).

In this paper we study the decidability of message deduction and static equivalence. We define a relation $\phi \vdash M$ that means that M can be deduced from ϕ, and a relation $\varphi \approx_s \psi$ that means that φ and ψ are statically equivalent; here ϕ, φ, and ψ are all essentially lists of messages, each with a name, represented by formal expressions. For generating these messages, we allow the application of a wide array of functions—pairing, projections, various flavors of encryption and decryption, digital signatures, one-way hash functions, etc.. Indeed, our results do not make any assumption on any particular cryptographic system beyond fairly general hypotheses on the form of the equational theory that is used for defining the properties of the cryptographic operations. Our main positive results assume only that the equational theory is defined by a convergent rewriting system with a finite number of rules of the form $M \to N$ where N is a proper subterm of M or a constant symbol. Such theories, which we call convergent subterm theories, appear frequently in applications. For them, we obtain that both $\phi \vdash M$ and $\varphi \approx_s \psi$ are decidable, in fact in polynomial time. For other equational theories, even decidable ones, we show that $\phi \vdash M$ and $\varphi \approx_s \psi$ can be undecidable. Moreover, we establish that $\vdash$ can be reduced to $\approx_s$ (not too surprisingly), but that the converse does not hold.

The problem of deciding knowledge is particularly important in the context of algorithms and tools for automated protocol analysis. Often, special techniques are introduced for particular sets of cryptographic operations of interest, on a case-by-case basis. For example, the classic Dolev-Yao result deals with a fixed, limited suite of public-key

operations [15]; more recent decidability results deal with exclusive-or and modular exponentiation (e.g., [10,11,12]); many variants and combinations that arise in practice have not yet been explored. On the other hand, other algorithms and tools (e.g., [6,7, 8]) allow much freedom in the choice of cryptographic operations but their analysis of the knowledge of the attacker is not always guaranteed to terminate. Decidability results under general equational theories have been rare. The most relevant previous work is that of Comon-Lundh and Treinen [13], who have studied the decidability of the deduction problem for a class of equational theories incomparable with ours. (For example, they allow the homomorphism property $\mathsf{enc}(\langle u, v\rangle, k) = \langle \mathsf{enc}(u, k), \mathsf{enc}(v, k)\rangle$ but not the inverse property $I(I(x)) = x$.) Simultaneously with our work (but independently), Delaune and Jacquemard [14] have shown that the deduction problem is decidable for an active attacker and under a class of equational theories which is included in ours. Neither Comon-Lundh and Treinen nor Delaune and Jacquemard considered static equivalence.

The next section, section 2, introduces notations and definitions. Section 3 compares $\vdash$ and $\approx_s$. Section 4 focuses on convergent subterm theories and gives our main decidability results. Section 5 concludes and discusses the possible use of our results for automated analysis of security protocols. Because of space constraints, we omit many technical details; the main ones appear in a research report [1].

2 Basic Definitions

Next we review definitions from previous work. We mostly adopt the definitions of the applied pi calculus [2]. In section 2.1 we give the syntax of expressions. In section 2.2 we explain a representation for the information available to an observer who has seen messages exchanged in the course of a protocol execution. In section 2.3 and 2.4 we present the relations $\vdash$ and $\approx_s$, which (as explained in the introduction) provide two formalizations of the knowledge that the observer has on the basis of that information.

2.1 Syntax

A *signature* Σ consists of a finite set of function symbols, such as enc and pair, each with an arity. Let $\mathsf{ar}(\Sigma)$ be the maximal arity of a function symbol in Σ. A function symbol with arity 0 is a constant symbol.

Given a signature Σ, an infinite set of names $\mathcal{N}$, and an infinite set of variables, the set of *terms* is defined by the grammar:

$L, M, N, T, U, V ::=$	terms
$k, \ldots, n, \ldots, s$	name
x, y, z	variable
$f(M_1, \ldots, M_l)$	function application

where f ranges over the function symbols of Σ and l matches the arity of f. Although names, variables, and constant symbols have similarities, we find it clearer to keep them separate. A term is closed when it does not have free variables (but it may contain names and constant symbols). We write $fn(M)$ for the set of names that occur in the term M. We use meta-variables u, v, w to range over names and variables. The *size* $|T|$ of a term

T is defined by $|u| = 1$ and $|f(T_1, \ldots, T_l)| = 1 + \sum_{i=1}^{l} |T_i|$. The *DAG-size* $|T|_{\mathsf{DAG}}$ is the number of distinct subterms of T.

We equip the signature Σ with an equational theory E, that is, an equivalence relation on terms that is closed under substitutions of terms for variables and closed under application of contexts. We write $M =_E N$ when M and N are closed terms and the equation $M = N$ is in E. We use the symbol == to denote syntactic equality of closed terms. As in these definitions, we often focus on closed terms for simplicity.

2.2 Assembling Terms into Frames

After a protocol execution, an attacker may know a sequence of messages $M_1, \ldots, M_l$. This means that it knows each message but it also knows in which order it received the messages. So it is not enough for us to say that the attacker knows the set of terms $\{M_1, \ldots, M_l\}$. Furthermore, we should distinguish those names that the attacker had before the execution from those that were freshly generated and which may remain secret from the attacker; both kinds of names may appear in the terms.

In the applied pi calculus [2], such a sequence of messages is organized into a *frame* $\nu\tilde{n}\sigma$, where $\tilde{n}$ is a finite set of names (intuitively, the fresh names), and σ is a substitution of the form:

$$\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\} \quad \text{with} \quad dom(\sigma) \stackrel{\text{def}}{=} \{x_1, \ldots, x_l\}.$$

The variables enable us to refer to each M_i, for example for keeping track of their order of transmission. We always assume that the terms M_i are closed. The size of a frame $\phi = \nu\tilde{n}\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\}$ is $|\phi| \stackrel{\text{def}}{=} \sum_{i=1}^{l} |M_i|$.

2.3 Deduction

Given a frame ϕ that represents the information available to an attacker, we may ask whether a given term closed M may be deduced from ϕ. This relation is written $\phi \vdash M$ (following Schneider [21]). It is axiomatized by the rules:

$$\frac{}{\nu\tilde{n}\sigma \vdash M} \text{ if } \exists x \in dom(\sigma) \text{ s.t. } x\sigma = M \qquad \frac{}{\nu\tilde{n}\sigma \vdash s} \; s \notin \tilde{n}$$

$$\frac{\phi \vdash M_1 \quad \cdots \quad \phi \vdash M_k}{\phi \vdash f(M_1, \ldots, M_k)} \; f \in \Sigma \qquad \frac{\phi \vdash M \quad M =_E M'}{\phi \vdash M'}$$

Since the deducible messages depend on the underlying equational theory, we write $\vdash_E$ when E is not clear from the context. Intuitively, the deducible messages are the messages of ϕ and the names which are not protected in ϕ, closed by equality in E and closed by application of functions. We have the following characterization of deduction:

Proposition 1. *Let M be a closed term and $\nu\tilde{n}\sigma$ be a frame. Then $\nu\tilde{n}\sigma \vdash M$ if and only if there exists a term ζ such that $fn(\zeta) \cap \tilde{n} = \emptyset$ and $\zeta\sigma =_E M$.*

As an example, we consider the equational theory of pairing and symmetric encryption. The signature is $\Sigma_1 = \{\mathsf{pair}, \mathsf{enc}, \mathsf{fst}, \mathsf{snd}, \mathsf{dec}\}$. As usual, we write $\langle x, y \rangle$ instead of $\mathsf{pair}(x, y)$. The theory E_1 is defined by the axioms:

$$\mathsf{fst}(\langle x, y \rangle) = x \qquad \mathsf{snd}(\langle x, y \rangle) = y \qquad \mathsf{dec}(\mathsf{enc}(x, y), y) = x.$$

Let $\phi \stackrel{\text{def}}{=} \nu k, s\{\mathsf{enc}(s, k)/x, k/y\}$. Then $\phi \vdash k$ and $\phi \vdash s$. Furthermore, we have $k =_{E_1} y\phi$ and $s =_{E_1} \mathsf{dec}(x, y)\phi$.

2.4 Static Equivalence

Deduction does not always suffice for expressing the knowledge of an attacker, as discussed in the introduction. For example, consider $\phi_1 \stackrel{\text{def}}{=} \nu k\{\mathsf{enc}(0, k)/x, k/y\}$ and $\phi_2 \stackrel{\text{def}}{=} \nu k\{\mathsf{enc}(1, k)/x, k/y\}$, where $0, 1 \in \Sigma$ are constant symbols. The attacker can deduce the same set of terms from these two frames since it knows 0 and 1. But it could tell the difference between these two frames by checking whether the decryption of x with y produces 0 or 1.

We say that two terms M and N are equal in the frame φ for the equational theory E, and write $(M =_E N)\varphi$, if and only if $\varphi = \nu\tilde{n}.\sigma$, $M\sigma =_E N\sigma$, and $\{\tilde{n}\} \cap (fn(M) \cup fn(N)) = \emptyset$ for some names $\tilde{n}$ and substitution σ. Then we say that two frames φ and ψ are *statically equivalent,* and write $\varphi \approx_s \psi$, when $dom(\varphi) = dom(\psi)$ and when, for all terms M and N, we have $(M =_E N)\varphi$ if and only if $(M =_E N)\psi$. We write $\approx_{sE}$ when E is not clear from the context.

In our example, we have $(\mathsf{dec}(x, y) =_{E_1} 0)\phi_1$ but not $(\mathsf{dec}(x, y) =_{E_1} 0)\phi_2$. Therefore, $\phi_1 \not\approx_s \phi_2$ although $\nu k\{\mathsf{enc}(0, k)/x\} \approx_s \nu k\{\mathsf{enc}(1, k)/x\}$.

3 Comparison of Deduction and Static Equivalence

We compare equality, deduction, and static equivalence from the point of view of decidability. There is little hope that deduction or static equivalence would be decidable when equality itself is not. (We note however that, for some artificial, especially designed equational theories, deduction may be decidable while equality is undecidable.) Therefore, we focus on equational theories for which equality is at least decidable.

3.1 $\vdash$ May Be Undecidable

Unfortunately, the decidability of equality is not sufficient for the decidability of deduction and static equivalence. As evidence, let us consider the decidable equational theory E_2 defined by:

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$
$$[x_1, y_1] \cdot [x_2, y_2] = [x_1 \cdot x_2, y_1 \cdot y_2]$$
$$f([x \cdot y, x \cdot y]) = f([x, x])$$

According to these equations, the symbol $\cdot$ is associative and distributes over the symbol $[\,]$, and any term of the form $f(M, M)$ can be collapsed into any term $f(M', M')$ where

M' is a prefix of M. This equational theory enables us to encode thePost Correspondence Problem (PCP) into the deduction problem.

Proposition 2. *The deduction problem for* E_2 ($\vdash_{E_2}$) *is undecidable.*

ThePCP is: given a finite number of pairs of words $(u_i, v_i)_{1 \leq i \leq n}$ on the alphabet $A \subset \mathcal{N}$, does there exists a sequence $s_1 \cdots s_k \in \{1..n\}^*$ such that: $u_{s_1} \cdots u_{s_k} = v_{s_1} \cdots v_{s_k}$? We map the PCP input $(u_i, v_i)_{1 \leq i \leq n}$ to the substitution $\sigma = \{[u_i, v_i]/x_i\}$. Then we can verify that there exists a solution to the PCP if and only if there exists a letter $a \in A$ such that $(\nu \Sigma)\sigma \vdash_{E_2} T([a, a])$.

3.2 $\vdash$ Reduces to $\approx_s$

Next we show that deduction may be reduced to static equivalence. For this purpose, we add the familiar equation $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$. (We have not studied what happens without this equation, since it is so common in applications.)

Proposition 3. *Let E be an equational theory over some signature* Σ. *Let* 0,1 *be two constants,* dec *and* enc *be two binary function symbols that are not in* Σ.
We define $\Sigma' \stackrel{def}{=} \Sigma \uplus \{0, 1, \mathsf{enc}, \mathsf{dec}\}$ *and* $E' \stackrel{def}{=} E \uplus \{\mathsf{dec}(\mathsf{enc}(x, y), y) = x\}$. *Let* $\phi = \nu \tilde{n}\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}\}$ *be a frame and M be a closed term. Then* $\phi \vdash_E M$ *if and only if*

$$\nu \tilde{n}\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}, ^{\mathsf{enc}(0,M)}/_{x_{l+1}}\} \not\approx_{sE'} \nu \tilde{n}\{^{M_1}/_{x_1}, \ldots, ^{M_l}/_{x_l}, ^{\mathsf{enc}(1,M)}/_{x_{l+1}}\}.$$

We derive that if $\approx_s$ is decidable for $E \uplus \{\mathsf{dec}(\mathsf{enc}(x, y), y) = x\}$, then $\vdash$ is decidable for E (with at most the same complexity).

3.3 $\approx_s$ Does Not Reduce to $\vdash$ in General

The converse is not true: $\vdash$ may be decidable while $\approx_s$ is not. Indeed, we can encode an undecidable problem into the static equivalence problem in such a way that the deduction problem remains decidable.

Proposition 4. *There exists an equational theory such that* $\approx_s$ *is undecidable while* $\vdash$ *is decidable.*

We consider the following construction: Given two deterministic Turing machines $M_1 = (Q, A, q_0, Q_f, \delta_1)$ and $M_2 = (Q, A, q_0, Q_f, \delta_2)$ with the same control states, where $\delta_1, \delta_2 : Q \times A \to Q \times A \times \{L, R\}$, we construct the machine $\mathcal{M}(M_1, M_2) = (Q, A, q_0, Q_f, \delta)$ where $\delta : \{1, 2\} \times Q \times A \to Q \times A \times \{L, R\}$ such that $\delta(1, q, a) = \delta_1(q, a)$ and $\delta(2, q, a) = \delta_2(q, a)$. At each step, the machine $\mathcal{M}(M_1, M_2)$ plays a transition of either M_1 or M_2. Since the machines M_1 and M_2 are deterministic, a run of the machine $\mathcal{M}(M_1, M_2)$ on a word w may be described by a word s of $\{1, 2\}^*$, which gives the list of choices made by $\mathcal{M}(M_1, M_2)$ at each step. $\mathcal{M}(M_1, M_2), w \xrightarrow{s}$ denotes the machine (with its current tape) after the sequence of choices s on the word w. We assume that the local control state is written on the tape.

Proposition 5. *The following problem is undecidable.*
Input: *Two machines* $\mathcal{M}(M_1, M_2)$ *and* $\mathcal{M}(M_1', M_2')$ *and a word* w *of* A^*.
Output: *Does the following property hold for* $\mathcal{M}(M_1, M_2)$ *and* $\mathcal{M}(M_1', M_2')$*: for any sequences* $s_1, s_2 \in \{1,2\}^*$, $\mathcal{M}(M_1, M_2), w \xrightarrow{s_1}$ *and* $\mathcal{M}(M_1, M_2), w \xrightarrow{s_2}$ *have the same tape if and only if* $\mathcal{M}(M_1', M_2'), w \xrightarrow{s_1}$ *and* $\mathcal{M}(M_1', M_2'), w \xrightarrow{s_2}$ *have the same tape?*

We reduce this undecidable problem to the $\approx_s$ problem under an equational theory E_3 such that $\vdash$ remains decidable. The intuitive idea of our encoding is that a frame ϕ represents a machine of the form $\mathcal{M}(M_1, M_2)$, a term M represents a sequence of choices such that $M\phi$ represents the tape of the machine (and the number of choices) after this sequence of choices. Then, for two "machines" ϕ and ϕ', it is undecidable whether there exists two sequences of choices M_1, M_2 such that $(M_1 =_{E_3} M_2)\phi$ and $(M_1 \neq_{E_3} M_2)\phi'$, *i.e.,* whether $\phi \not\approx_s \phi'$.

On the other hand, it is possible to decide whether there exists a sequence of choices M such that $M\phi =_{E_3} N$ (*i.e.,* whether $\phi \vdash N$) for a given term N. Indeed, the term N contains the number of choices, so it is sufficient to test any sequence of choices of length equal to this number of choices.

4 Deciding Knowledge Under Convergent Subterm Theories

In order to obtain decidability results for both $\vdash$ and $\approx_s$, we restrict attention to *subterm theories,* defined by a finite set of equations of the form $M = N$ where N is a proper subterm of M or a constant symbol. In section 4.1, we motivate and introduce a convergence condition on subterm theories. Convergent subterm theories are quite common in applications, as we illustrate with examples in section 4.2. We present our main decidability results for these theories in section 4.3.

4.1 Convergence

The definition of subterm theories is almost vacuous on its own. Even equality may be undecidable for subterm theories. Any equational theory defined by a finite set of equations $M = M'$ with variables can be encoded as a subterm theory, with the two equations:

$$\mathsf{Whichever}(M, M') = M \qquad \mathsf{Whichever}(M, M') = M'$$

for each original equation $M = M'$. In light of this encoding, we should add the assumption that, by orienting the equations that define a subterm theory from left to right, we obtain a convergent rewriting system:

Definition 1. *A equational theory E, defined by a finite set of equations* $\bigcup_{i=1}^{n}\{M_i = N_i\}$ *where* $fn(M_i) = fn(N_i) = \emptyset$*, is a* convergent subterm theory *if the set of rewriting rules* $r(E) \stackrel{def}{=} \bigcup_{i=1}^{n}\{M_i \to N_i\}$ *is convergent and if each* N_i *is a proper subterm of* M_i *or a constant. We write* $U \to V$ *if U and V are closed terms and U may be rewritten to V (in one step) using a rule of* $r(E)$.

As usual, if $r(E)$ is convergent then for all terms U, V, we have $U =_E V$ if and only if $U{\downarrow} = V{\downarrow}$, where $U{\downarrow}$ and $V{\downarrow}$ are the normal forms of U and V.

We write $\to_E$ instead of $\to$ when the equational theory is not clear from the context.

4.2 Examples

Important destructor-constructor rules like those for pairing, encryption, and signature may be expressed in subterm theories (typically convergent ones):

$$\mathsf{fst}(< x, y >) = x \qquad \mathsf{dec}(\mathsf{enc}(x, y), y) = x$$
$$\mathsf{snd}(< x, y >) = y \qquad \mathsf{check}(x, \mathsf{sign}(x, \mathsf{sk}(y)), \mathsf{pk}(y)) = \mathsf{ok}$$

Additional examples can be found in previous work (e.g., [2,8]). Convergent subterm theories also enable us to capture sophisticated but sensible properties, as in:

$$E_4 : \{I(I(x)) = x, I(x) \times x = 1, x \times I(x) = 1\},$$
$$E_5 : \{h(h(x)) = x\},$$
$$E_6 : \{\mathsf{enc}(\mathsf{enc}(x, y), y) = x\}.$$

The theory E_4 models an inverse function. The theory E_5 models a hash function that is idempotent on small inputs (since the hash of a hash gives the same hash). The theory E_6 represents an encryption function that also decrypts: the encryption of a plaintext, twice with the same key, returns the plaintext.

4.3 Decidability Results

For convergent subterm theories, both $\vdash$ and $\approx_s$ become decidable. Let E be a convergent subterm theory given by $\bigcup_{i=1}^{n}\{M_i = N_i\}$, and $c_E = \max_{1 \le i \le n}(|M_i|, \mathsf{ar}(\Sigma) + 1)$.

Theorem 1. *For any frames ϕ and ϕ', for any closed term M, we can decide $\phi \vdash M$ and $\phi \approx_s \phi'$ in polynomial time in $|\phi|$, $|\phi'|$, and $|M|$.*

The end of this section is devoted to outlining the proof of the theorem.

Step 1 of the proof: saturating a frame ϕ. We first associate with each frame ϕ the set of subterms of messages in ϕ that may be deduced from ϕ by applying only small contexts. We prove that this set can be computed in polynomial time. In addition, we show that each term in this set has a "representation" whose DAG-size is polynomial.

Definition 2. *Let $\phi = \nu\tilde{n}\{M_1/x_1, \ldots, M_k/x_k\}$ be a frame. Let $\mathsf{st}(\phi)$ be the set of subterms of the M_i's. The saturation $\mathsf{sat}(\phi)$ of ϕ is the minimal set such that:*

1. *for every $1 \le i \le k$, $M_i \in \mathsf{sat}(\phi)$,*
2. *if $M_1, \ldots, M_k \in \mathsf{sat}(\phi)$ and $C[M_1, \ldots, M_k] \to M$, where C is a context, $|C| \le c_E$, $fn(C) \cap \tilde{n} = \emptyset$, and $M \in \mathsf{st}(\phi)$ then $M \in \mathsf{sat}(\phi)$,*
3. *if $M_1, \ldots, M_k \in \mathsf{sat}(\phi)$ and $f(M_1, \ldots, M_k) \in \mathsf{st}(\phi)$, then $f(M_1, \ldots, M_k) \in \mathsf{sat}(\phi)$.*

Proposition 6. *Let ϕ be a frame, $\phi = \nu\tilde{n}\sigma$.*

1. *The set $\mathsf{sat}(\phi)$ can be computed in time $\mathcal{O}(|\phi|^{\max(\mathsf{ar}(\Sigma), c_E)+2})$.*

2. *For every* $M \in \mathsf{sat}(\phi)$, *there exists a term* ζ_M *such that* $fn(\zeta_M) \cap \tilde{n} = \emptyset$, $|\zeta_M|_{\mathsf{DAG}} \leq (c_E + 1)|\phi|$, *and* $\zeta_M \sigma =_E M$. *The term* ζ_M *is called a* recipe *of* M *and is chosen arbitrarily between the possible terms verifying these properties.*

The set $\mathsf{sat}(\phi)$ is obtained by saturating the set $\{M_1, \ldots, M_k\}$ by applying the rules 2 and 3 of definition 2. Since $\mathsf{sat}(\phi) \subseteq \mathsf{st}(\phi)$, this set is saturated in at most $|\phi|$ steps. At each step, we have to compute:

- Every closed term of the form $C[M_1, \ldots, M_k]$ (up to renamings in C), where $|C| \leq c_E$ and the M_i's are already in the set, and check if it is an instance of some left-hand side of a rule. Thus we need at most $\mathcal{O}(|\phi|^{c_E+1})$ computations.
- Every term $f(M_1, \ldots, M_k)$ that is also in $\mathsf{st}(\phi)$. Thus we have to construct at most $|\Sigma||\phi|^{\mathsf{ar}(\Sigma)}$ terms.

Since each step requires at most $\mathcal{O}(|\phi|^{\max(\mathsf{ar}(\Sigma), c_E+1)})$ computations and since there are at most $|\phi|$ steps, $\mathsf{sat}(\phi)$ may be computed in time $\mathcal{O}(|\phi|^{\max(\mathsf{ar}(\Sigma), c_E)+2})$. For the second part of proposition 6, we already know by proposition 1 that each term M of $\mathsf{sat}(\phi)$ has a representation ζ_M such that $fn(\zeta_M) \cap \tilde{n} = \emptyset$ and $\zeta_M \sigma =_E M$. By construction of $\mathsf{sat}(\phi)$, the recipes may be chosen so that:

1. $\zeta_M = x_i$ if $\sigma(x_i) = M$,
2. $\zeta_M = C[\zeta_{M_1}, \ldots, \zeta_{M_k}]$ with $M_i \in \mathsf{sat}(\phi)$ if M is obtained by the rule 2,
3. $\zeta_M = f(\zeta_{M_1}, \ldots, \zeta_{M_k})$ with $M_i \in \mathsf{sat}(\phi)$ if M is obtained by the rule 3.

Since there are at most $|\mathsf{sat}(\phi)| \leq |\phi|$ recipes, the maximal DAG-size of a recipe of a term in $\mathsf{sat}(\phi)$ is $(c_E + 1)|\phi|$.

Step 2 of the proof: Introducing a finite set of equalities to characterize a frame. With each frame ϕ, we associate a set of equalities $\mathsf{Eq}(\phi)$ (finite modulo renaming) such that two frames are equivalent if and only if they satisfy the equalities from each other's set: ϕ' satisfies the equalities $\mathsf{Eq}(\phi)$ and ϕ satisfies the equalities $\mathsf{Eq}(\phi')$. We assume fixed the set of recipes corresponding to the terms of $\mathsf{sat}(\phi)$.

Definition 3. *Let* $\phi = \nu \tilde{n} \sigma$ *be a frame. The set* $\mathsf{Eq}(\phi)$ *is the set of equalities*

$$C_1[\zeta_{M_1}, \ldots, \zeta_{M_k}] = C_2[\zeta_{M'_1}, \ldots, \zeta_{M'_l}]$$

such that $(C_1[\zeta_{M_1}, \ldots, \zeta_{M_k}] =_E C_2[\zeta_{M'_1}, \ldots, \zeta_{M'_l}])\phi$, $|C_1|, |C_2| \leq c_E$, *and the* M_i *and* M'_i *are in* $\mathsf{sat}(\phi)$. *If* ϕ' *is a frame such that* $(M =_E N)\phi'$ *for every* $(M = N) \in \mathsf{Eq}(\phi)$, *we write* $\phi' \models \mathsf{Eq}(\phi)$.

Two crucial lemmas show that it is sufficient to consider these equalities:

Lemma 1. *Let* $\phi = \nu \tilde{n} \sigma$ *and* $\phi' = \nu \tilde{n'} \sigma'$ *be two frames such that* $\phi' \models \mathsf{Eq}(\phi)$. *For all contexts* C_1, C_2 *such that* $(fn(C_1) \cup fn(C_2)) \cap \tilde{n} = \emptyset$, *for all terms* $M_i, M'_i \in \mathsf{sat}(\phi)$, *if* $C_1[M_1, \ldots, M_k] == C_2[M'_1, \ldots, M'_l]$, *then* $(C_1[\zeta_{M_1}, \ldots, \zeta_{M_k}] =_E C_2[\zeta_{M'_1}, \ldots, \zeta_{M'_l}])\phi'$.

Lemma 2. *Let $\phi = \nu\tilde{n}\sigma$ be a frame. For every context C_1 such that $fn(C_1) \cap \tilde{n} = \emptyset$, for every $M_i \in \mathsf{sat}(\phi)$, for every term T such that $C_1[M_1, \ldots, M_k] \rightarrow_E T$, there exist a context C_2 such that $fn(C_2) \cap \tilde{n} = \emptyset$, and terms $M'_i \in \mathsf{sat}(\phi)$, such that $T == C_2[M'_1, \ldots, M'_l]$ and for every frame $\phi' \models \mathsf{Eq}(\phi)$, $(C_1[\zeta_{M_1}, \ldots, \zeta_{M_k}] =_E C_2[\zeta_{M'_1}, \ldots, \zeta_{M'_l}])\phi'$.*

How these lemmas are used to prove the decidability of deduction and static equivalence is explained in steps 3 and 4 of the proof, respectively.

Step 3 of the proof: decidability of $\vdash$. Here we show that any message deducible from a frame ϕ is actually a context over terms in $\mathsf{sat}(\phi)$.

Proposition 7. *Let $\phi = \nu\tilde{n}\sigma$ be a frame, M be a closed term and $M\downarrow$ its normal form. Then $\phi \vdash M$ if and only if there exist C and $M_1, \ldots, M_k \in \mathsf{sat}(\phi)$ such that $fn(C) \cap \tilde{n} = \emptyset$ and $M\downarrow == C[M_1, \ldots, M_k]$.*

If $M\downarrow == C[M_1, \ldots, M_k]$ with $fn(C) \cap \tilde{n} = \emptyset$, then $M =_E C[\zeta_{M_1}, \ldots, \zeta_{M_k}]\sigma$, by construction of the ζ_{M_i}'s. Thus, by proposition 1, $\phi \vdash M$. Conversely, if $\phi \vdash M$, then by proposition 1, there exists ζ such that $fn(\zeta) \cap \tilde{n} = \emptyset$ and $M =_E \zeta\sigma$. Thus $M\downarrow == (\zeta\sigma)\downarrow$. Applying recursively lemma 2, we obtain that $(\zeta\sigma)\downarrow == C[M_1, \ldots, M_k]$ for some $M_1, \ldots, M_k \in \mathsf{sat}(\phi)$ and C such that $fn(C) \cap \tilde{n} = \emptyset$.

We derive that $\phi \vdash M$ can be decided by checking whether $M\downarrow$ is of the form $C[M_1, \ldots, M_k]$ with $M_i \in \mathsf{sat}(\phi)$. Given a term M, $M\downarrow$ can be computed in polynomial time. Once $\mathsf{sat}(\phi)$ is computed (in polynomial time by proposition 6), checking whether there exist C and $M_1, \ldots, M_k \in \mathsf{sat}(\phi)$ such that $fn(C) \cap \tilde{n} = \emptyset$ and $M\downarrow == C[M_1, \ldots, M_k]$ may be done in time $\mathcal{O}(|M||\phi|^2)$. We conclude that $\phi \vdash M$ is decidable in polynomial time.

Step 4 of the proof: decidability of $\approx_s$.

Proposition 8. *For all frames ϕ and ϕ', we have $\phi \approx_s \phi'$ if and only if $\phi \models \mathsf{Eq}(\phi')$ and $\phi' \models \mathsf{Eq}(\phi)$.*

By definition of static equivalence, if $\phi \approx_s \phi'$ then $\phi \models \mathsf{Eq}(\phi')$ and $\phi' \models \mathsf{Eq}(\phi)$. Conversely, assume now that $\phi' \models \mathsf{Eq}(\phi)$ and consider M, N such that there exist $\tilde{n}, \sigma$ such that $\phi = \nu\tilde{n}\sigma$, $(fn(M) \cup fn(N)) \cap \tilde{n} = \emptyset$ and $(M =_E N)\phi$. Then $M\sigma =_E N\sigma$, so $(M\sigma)\downarrow == (N\sigma)\downarrow$. Let $T = (M\sigma)\downarrow$. Applying recursively lemma 2, we obtain that there exist $M_1, \ldots, M_k \in \mathsf{sat}(\phi)$ and C_M such that $fn(C_M) \cap \tilde{n} = \emptyset$ and

$$T == C_M[M_1, \ldots, M_k] \text{ and } M\sigma' =_E C_M[\zeta_{M_1}, \ldots, \zeta_{M_k}]\sigma'.$$

Since $T == (N\sigma)\downarrow$, we obtain similarly that there exist $M'_1, \ldots, M'_l \in \mathsf{sat}(\phi)$ and C_N such that $fn(C_N) \cap \tilde{n} = \emptyset$ and

$$T == C_N[M'_1, \ldots, M'_l] \text{ and } N\sigma' =_E C_N[\zeta_{M'_1}, \ldots, \zeta_{M'_l}]\sigma'.$$

Moreover, since $C_M[M_1, \ldots, M_k] == C_N[M'_1, \ldots, M'_l]$, we derive from lemma 1 that $C_M[\zeta_{M_1}, \ldots, \zeta_{M_k}]\sigma' =_E C_N[\zeta_{M'_1}, \ldots, \zeta_{M'_l}]\sigma'$, thus $(M =_E N)\phi'$. Conversely, if $(M =_E N)\phi'$ and $\phi \models \mathsf{Eq}(\phi')$, we can prove that $(M =_E N)\phi$. We conclude $\phi \approx_s \phi'$.

Therefore, given ϕ and ϕ', to decide whether $\phi \approx_s \phi'$ we construct $\mathsf{sat}(\phi)$ and $\mathsf{sat}(\phi')$. This can be done in polynomial time by proposition 6. For each term M of $\mathsf{sat}(\phi)$ or $\mathsf{sat}(\phi')$, the term ζ_M has a polynomial DAG-size. Then, for all contexts C_1, C_2 such that $|C_1|, |C_2| \leq c_E$, for all $M_i, M'_i \in \mathsf{sat}(\phi)$, we check whether $(C_1[\zeta_{M_1}, \ldots, \zeta_{M_k}] =_E C_2[\zeta_{M'_1}, \ldots, \zeta_{M'_l}])\phi$ and $(C_1[\zeta_{M_1}, \ldots, \zeta_{M_k}] =_E C_2[\zeta_{M'_1}, \ldots, \zeta_{M'_l}])\phi'$. There are at most $\mathcal{O}((|\phi|^{c_E})^2)$ equalities in $\mathsf{Eq}(\phi)$ (up to renamings of the names in C_1 and C_2). Each term of the form $C_1[\zeta_{M_1}, \ldots, \zeta_{M_k}]\phi$ has a polynomial DAG-size. The equality of two terms represented by DAGs can be checked in polynomial time: we do not need to expand the DAGs to test for equality. We conclude that $\phi \approx_s \phi'$ can be decided in polynomial time in $|\phi|$ and $|\phi'|$.

Although this proof is effective, the complexity bounds that we obtain from it appear rather high. For example, for the equational theory E_1 of section 2.3, we can obtain that $\phi \vdash M$ is decidable in time $\mathcal{O}(|M|^3|\phi|^7)$. It should be possible to do much better.

5 Conclusion

This paper investigates decidability questions for message deducibility and static equivalence, two formal representations for knowledge in the analysis of security protocols. This investigation yields a few somewhat negative results, for example that static equivalence cannot always be reduced to message deducibility. On the other hand, the main results are strong, positive ones: both message deducibility and static equivalence are decidable in polynomial time under a large and useful class of equational theories.

These positive results suggest some directions for further research in protocol analysis. In the general case of infinite-state protocols, our algorithms could be integrated into analysis tools; substantial work on optimizations may however be required. For finite-state protocols, various security properties are decidable under specific equational theories (e.g., [5]). Perhaps our results can serve as the starting point for a generalization to a broad class of equational theories. This generalization may be easy if one restricts attention to passive attackers (eavesdroppers): since the capabilities of eavesdroppers are limited to deducing and comparing messages, our decidability results may apply fairly directly. The case with active attackers is clearly more difficult and interesting; as mentioned in the introduction, Delaune and Jacquemard have recently proved that the deduction problem is still decidable for a subclass of convergent subterm theories. It remains to study whether this work could be extended to establish process equivalences (such as testing equivalences [4]).

Acknowledgments. We are grateful to Michael Rusinowitch for helpful discussions.

References

1. M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. Technical Report RR-5169, INRIA, April 2004. An up-to-date version will be kept at `http://www.loria.fr/~cortier/publis.html`.

2. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01),* pages 104–115, January 2001.
3. M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing,* 5(4):267–303, Winter 1998.
4. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation,* 148(1):1–70, Jan. 1999.
5. R. M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In C. Palamidessi, editor, *CONCUR 2000: Concurrency Theory (11th Int. Conference),* volume 1877 of *LNCS,* pages 380–394. Springer Verlag, Aug. 2000.
6. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14),* pages 82–96, June 2001.
7. B. Blanchet. From secrecy to authenticity in security protocols. In M. Hermenegildo and G. Puebla, editors, *9th Int. Static Analysis Symposium (SAS'02),* volume 2477 of *LNCS,* pages 342–359. Springer Verlag, Sept. 2002.
8. B. Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy,* May 2004, to appear.
9. M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Proceedings of the Fourteenth Annual IEEE Symposium on Logic in Computer Science,* pages 157–166, July 1999.
10. Y. Chevalier, R. Kuester, M. Rusinowitch, and M. Turani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In P. K. Pandya and J. Radhakrishnan, editors, *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference,* volume 2914 of *LNCS,* pages 124–135. Springer Verlag, 2003.
11. Y. Chevalier, R. Kuester, M. Rusinowitch, and M. Turani. An NP decision procedure for protocol insecurity with xor. In *Proceedings of the 18th Annual IEEE Symposium on Logic In Computer Science (LICS'03),* pages 261–270,2003.
12. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proceedings of the 18th Annual IEEE Symposium on Logic In Computer Science (LICS'03),* pages 271–280, 2003.
13. H. Comon-Lundh and R. Treinen. Easy intruder deductions. Technical Report LSV-03-8, Laboratoire Spécification et Vérification, ENS de Cachan, France, 2003.
14. S. Delaune and F. Jacquemard. Narrowing-based constraint solving for the verification of security protocols. Technical Report LSV-04-8, Laboratoire Spécification et Vérification, ENS de Cachan, France, April 2004.
15. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory,* IT-29(12):198–208, Mar. 1983.
16. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences,* 28:270–299, Apr. 1984.
17. R. Kemmerer, C. Meadows, and J. Millen. Three system for cryptographic protocol analysis. *Journal of Cryptology,* 7(2):79–130, Spring 1994.
18. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security, pages* 112–121, 1998.
19. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems,* volume 1055 of *LNCS,* pages 147–166. Springer Verlag, 1996.
20. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security,* 6(1–2):85–128,1998.

21. S. Schneider. Security properties and CSP. In *IEEE Symposium on Security and Privacy,* pages 174–187, 1996.

Representing Nested Inductive Types Using W-Types

Michael Abbott[1], Thorsten Altenkirch[2], and Neil Ghani[1]

[1] Department of Mathematics and Computer Science, University of Leicester
michael@araneidae.co.uk, ng13@mcs.le.ac.uk
[2] School of Computer Science and Information Technology, Nottingham University
txa@cs.nott.ac.uk

Abstract. We show that strictly positive inductive types, constructed from polynomial functors, constant exponentiation and arbitrarily nested inductive types exist in any Martin-Löf category (extensive locally cartesian closed category with W-types) by exploiting our work on container types. This generalises a result by Dybjer (1997) who showed that non-nested strictly positive inductive types can be represented using W-types. We also provide a detailed analysis of the categorical infrastructure needed to establish the result.

1 Introduction

Inductive types play a central role in programming and constructive reasoning. From an intuitionistic point of view we can understand strictly positive inductive types (SPITs) as well-founded trees, which may be infinitely branching. The language of SPITs is built from polynomial types and exponentials, enriched by a constructor μ for inductive types. In this language we can conveniently construct familiar types such as the natural numbers, $\mathbb{N} \equiv \mu X.1 + X$; binary trees, $\mathrm{BTree} \equiv \mu X.1 + X \times X$; lists parameterised over a type List $A \equiv \mu X.1 + A \times X$; ordinals, $\mathrm{Ord} \equiv \mu X.1 + X + X^{\mathbb{N}}$; and finitely branching trees as the fixpoint of Lists, $\mathrm{FTree} \equiv \mu Y.\mathrm{List}\, Y = \mu Y.\mu X.1 + X \times Y$. Categorically, μ corresponds to taking the initial algebra of a given functor.

The grammar of SPITs can be easily defined inductively, see definition 6.1. However, we would like to have a simple semantic criterion which guarantees the existence of SPITs. Dybjer (1997) shows that inductive types over strictly positive operators constructed using only polynomials in a single type variable and fixed exponentiation can be constructed in extensional Type Theory using W-types, the type of well-founded trees introduced in Martin-Löf (1984). However, Dybjer (1997) does not consider any nesting of inductive types, e.g. the example FTree is not covered by his definition. Here we present a more general result which shows that nested inductive types can be constructed using only W-types and we analyse the categorical framework in more detail.

An important ingredient in our construction is the insight that SPITs give rise to containers, which we have investigated in Abbott et al. (2003) and which are the topic of Abbott (2003). The basic notion of a *container* is a dependent pair of types $A \vdash B$ creating a functor $T_{A \triangleright B} X \equiv \sum a : A.\, X^{B(a)}$. A morphism of containers $(A \vdash B) \to (C \vdash D)$ is a pair of morphisms $(u : A \to C,\ f : u^* D \to B)$. With this definition of a category $\mathcal{G}$ of containers we can construct a full and faithful functor $T : \mathcal{G} \to [\mathbb{C}, \mathbb{C}]$.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 59–71, 2004.

However, when constructing fixed points it is also necessary to take account of containers with parameters, so we define $T : \mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ for each parameter index set I. For the purposes of this paper the index set I can be regarded as a finite set, but this makes little practical difference to the development.

It is easy to show that containers are closed under sums and products and constant exponentiation, see Abbott et al. (2003); this is also done in Dybjer (1997) for containers in one variable. W-types are precisely the initial algebras of containers in one variable (theorem 3.6), hence constructing inductive types over a single variable SPITs is straightforward and already covered (in part) by Dybjer's work. However, the general case for nested types corresponds to showing that containers are closed under initial algebras. The problem boils down (proposition 4.1) to solving an equation on families of types up to isomorphism, which is achieved in proposition 5.1.

The work presented here also overcomes a shortcoming of Abbott et al. (2003): there we constructed initial algebras of containers using the assumption that the ambient category is locally finitely presentable. Alas, this assumption rules out many interesting examples of categories, in particular realisability models such as **ω-sets.** This is fixed here, since we only have to require that the category has all W-types, i.e. initial algebras of container functors, which can be easily established for realisability models. Since dependent types and inductive types are the core of Martin-Löf's Type Theory, we call categories with this structure *Martin-Löf categories,* see definition 3.7. Dybjer and Setzer (1999, 2001) present general schemes for inductive (and inductive-recursive) definitions but they do not present a reduction to a combinator like W-types. Moreover, they also use universes extensively.

Recently Gambino and Hyland (2004) have put our results in a more general context and indeed their theorem 12 generalises our proposition 5.1 to dependently typed containers, which they call dependent polynomial functors. Similarly, their theorem 14 is closely related to our proposition 4.1. We also learnt from their work that this construction is related to the proof in Moerdijk and Palmgren (2000) that W-types localise to slice categories.

2 Definitions and Notation

This paper uses the dependent internal language of a locally cartesian closed category $\mathbb{C}$: see Streicher (1991), Hofmann (1994), Jacobs (1999) and Abbott (2003) for details. The key idea is regard an object $B \in \mathbb{C}/A$ as a *family* of objects of $\mathbb{C}$ indexed by elements of A, and to regard A as the *context* in which B regarded as a *type dependent on A* is defined.

Elements of A will be represented by morphisms $f : U \to A$ in $\mathbb{C}$, and *substitution* of f for A in B is implemented by pulling back B along f to $f^*B \in \mathbb{C}/U$. We start to build the internal language by writing $a : A \vdash B(a)$ to express B as a type *dependent* on values in A, and then the result of substitution of f is written as $u : U \vdash B(fu)$. We will treat $B(a)$ as an alias for B and $B(fu)$ as an alias for f^*B, and we'll write $a : A \vdash B(a)$ or even just $A \vdash B$ for $B \in \mathbb{C}/A$—**variables** will be omitted from the internal language where practical for conciseness.

Note that substitution by pullback extends to a functor $f^* : \mathbb{C}/A \to \mathbb{C}/U$: for conciseness of presentation we will assume that substitution corresponds precisely to a choice of pullback, but for a more detailed treatment of the issues involved see Hofmann (1994) and Abbott (2003).

Terms of type $A \vdash B$ correspond to *global elements* of B, which is to say morphisms $t : 1 \to B$ in $\mathbb{C}/A$. In the internal language we write $a : A \vdash t(a) : B(a)$ for such a morphism in $\mathbb{C}$. We will write t for $t(a)$ where practical, again omitting a variable when it can be inferred. Given object $A \vdash B$ and $A \vdash C$ we will write $A \vdash f : B \to C$ for a morphism in $\mathbb{C}/A$, and similarly $A \vdash f : B \cong C$ for an isomorphism.

The morphism in $\mathbb{C}$ associated with $B \in \mathbb{C}/A$ will be written as $\pi_B : \sum_A B \to A$ (the *display map* for B); the transformation $B \mapsto \sum_A B$ becomes a left adjoint functor $\sum_A \dashv \pi_B^*$, where pulling back along π_B plays the role of *weakening* with respect to a variable $b : B(a)$ in context $a : A$. In the type theory we'll write $\sum_A B \in \mathbb{C}$ as $1 \vdash \sum a : A.\, B(a)$, or more concisely $\vdash \sum_A B$, with elements $\vdash (t, u) : \sum a : A.\, B(a)$ corresponding to elements $\vdash t : A$ and $\vdash u : B(t)$.

More generally, all of the constructions described here localise: given an arbitrary context $\Gamma \in \mathbb{C}$ and an object $A \in \mathbb{C}/\Gamma$ we can use the isomorphism $(\mathbb{C}/\Gamma)/A \cong \mathbb{C}/\sum_\Gamma A$ to interpret $\Gamma, a : A \vdash B(a)$ both as a morphism $\pi_B : \sum_A B \to A$ in $\mathbb{C}/\Gamma$ and as $\pi_B : \sum_A B \to \sum_\Gamma A$ in $\mathbb{C}$, and $\sum$ extends to provide a left adjoint to every substitution functor. We will write $\Gamma, a : A, b : B(a) \vdash C(a, b)$ or just $\Gamma, A, B \vdash C$ as a shorthand for $\Gamma, (a, b) : \sum_A B \vdash C(a, b)$.

Local cartesian closed structure on $\mathbb{C}$ allows right adjoints to weakening $\pi_A^* \dashv \prod_A$ to be constructed for every $\Gamma \vdash A$ with type expression $\Gamma \vdash \prod a : A.\, B(b)$ for $\Gamma \vdash \prod_A B$ derived from $\Gamma, A \vdash B$. Finally the *equality type* A, $A \vdash \mathrm{Eq}_A$ is represented as an object of $\mathbb{C}/A \times A$ by the diagonal morphism $\delta_A : A \to A \times A$, and more generally $\Gamma, A, A \vdash \mathrm{Eq}_A$. Given parallel morphisms u, v into A the equality type has the key property that an element of $\mathrm{Eq}(u, v) = (u, v)^* \mathrm{Eq}_A$ exists precisely when $u = v$ as morphisms of $\mathbb{C}$.

For coproducts in the internal language to behave properly, in particular for containers to be closed under products, we require that $\mathbb{C}$ have *disjoint* coproducts: the pullback of distinct coprojections $\kappa_i : A_i \to \sum_{i \in I} A_i$ into a coproduct is always the initial object 0. When this holds the functor $\mathbb{C}/A + B \to (\mathbb{C}/A) \times (\mathbb{C}/B)$ taking $A + B \vdash C$ to $(A \vdash \kappa^* C,\ B \vdash \kappa'^* C)$ is an equivalence: write $- \dagger -$ for the inverse functor. Thus given $A \vdash B$ and $C \vdash D$ (with display maps π_B and π_D) we write $A + C \vdash B \dagger D$ for their disjoint sum; this satisfies two identities: $\sum_{A+C}(B \dagger D) \cong \sum_A B + \sum_C D$ and $\pi_{B \dagger D} = \pi_B + \pi_D$ (modulo the preceding isomorphism).

Given a (finite) index set I define $[\mathbb{C}^I, \mathbb{C}^J]$ to be the category of *fibred* functors and natural transformations $\mathbb{C}^I \to \mathbb{C}$ where the fibre of $\mathbb{C}^I$ over $\Gamma \in \mathbb{C}$ is the I-fold product $(\mathbb{C}/\Gamma)^I$. Of course, when $J = 1$ we will write this as $[\mathbb{C}^I, \mathbb{C}]$.

Basic Properties of Containers

We summarise here the development of containers in Abbott et al. (2003).

Definition 2.1. *Given an index set I define the* category of containers $\mathcal{G}_I$ *as follows:*

- *Objects are pairs* $(A \in \mathbb{C}, B \in (\mathbb{C}/A)^I)$; *write this as* $(A \triangleright B) \in \mathcal{G}_I$

– *A morphism* $(A \triangleright B) \to (C \triangleright D)$ *is a pair* (u, f) *for* $u : A \to C$ *in* $\mathbb{C}$ *and* $f : (u^*)^I D \to B$ *in* $(\mathbb{C}/A)^I$.

Note that the alternative of defining an $n+1$-**ary** container as an indexed family of n-**ary** containers is equivalent to this definition (Abbott, 2003, proposition 4.1.1).

A container $(A \triangleright B) \in \mathcal{G}_I$ can be written using type theoretic notation as

$$\vdash A \qquad i : I, a : A \vdash B_i(a) \ .$$

A morphism $(u, f) : (A \triangleright B) \to (C \triangleright D)$ can be written in type theoretic notation as

$$u : A \longrightarrow C \qquad i : I, a : A \vdash f_i(a) : D_i(ua) \longrightarrow B_i(a) \ .$$

Finally, each $(A \triangleright B) \in \mathcal{G}_I$, thought of as a syntactic presentation of a datatype, generates a fibred functor $T_{A \triangleright B} : \mathbb{C}^I \to \mathbb{C}$ which is its semantics.

Definition 2.2. *Define the* container construction functor $T : \mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ *as follows. Given* $(A \triangleright B) \in \mathcal{G}_I$ *and* $X \in \mathbb{C}^I$ *define*

$$T_{A \triangleright B} X \equiv \sum a : A. \prod_{i \in I} X_i^{B_i(a)} \ ,$$

and for $(u, f) : (A \triangleright B) \to (C \triangleright D)$ *define* $T_{u,f} : T_{A \triangleright B} \to T_{C \triangleright D}$ *to be the natural transformation* $T_{u,f} X : T_{A \triangleright B} X \to T_{C \triangleright D} X$ *thus:*

$$(a, g) : T_{A \triangleright B} X \vdash T_{u,f} X (a, g) \equiv (u(a), (g_i \cdot f_i)_{i \in I}) \ .$$

The following proposition follows more or less immediately by the construction of T.

Proposition 2.3 (Abbott et al., 2003, proposition 3.3). *For each container* $F \in \mathcal{G}_I$ *and each container morphism* $\alpha : F \to G$ *the functor* T_F *and natural transformation* T_α *are fibred over* $\mathbb{C}$. □

By making essential use of the fact that the natural transformations in $[\mathbb{C}^I, \mathbb{C}]$ are fibred we can show that T is full and faithful.

Theorem 2.4 (ibid., theorem 3.4). *The functor* $T : \mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ *is full and faithful.* □

This theorem gives a particularly simple analysis of polymorphic functions between container functors. For example, it is easy to observe that there are precisely n^m polymorphic functions $X^n \to X^m$: the data type X^n is the container $(1 \triangleright n)$ and hence there is a bijection between polymorphic functions $X^n \to X^m$ and functions $m \to n$. Similarly, any polymorphic function $\mathrm{List}\, X \to \mathrm{List}\, X$ can be uniquely written as a function $u : \mathbb{N} \to \mathbb{N}$ together with for each natural number $n : \mathbb{N}$, a function $f_n : un \to n$.

It turns out that each $\mathcal{G}_I$ inherits products and coproducts from $\mathbb{C}$, and that T preserves them:

Proposition 2.5 (ibid., propositions 4.1, 4.2). *If* $\mathbb{C}$ *has products and coproducts then* $\mathcal{G}_I$ *has products and coproducts preserved by* T. □

Given containers $F \in \mathcal{G}_{I+1}$ and $G \in \mathcal{G}_I$ we can compose their images under T to construct the functor

$$T_F[T_G] \equiv (\mathbb{C}^I \xrightarrow{(\mathrm{id}_{\mathbb{C}^I}, T_G)} \mathbb{C}^I \times \mathbb{C} \cong \mathbb{C}^{I+1} \xrightarrow{T_F} \mathbb{C}) \ .$$

This composition can be lifted to a functor $-[-]: \mathcal{G}_{I+1} \times \mathcal{G}_I \to \mathcal{G}_I$ as follows. For a container in $\mathcal{G}_{I+1}$ write $(A \triangleright B, E) \in \mathcal{G}_{I+1}$, where $B \in (\mathbb{C}/A)^I$ and $E \in \mathbb{C}/A$ and define:

$$(A \triangleright B, E)[(C \triangleright D)] \equiv \left(a{:}A, f{:}C^{E(a)} \triangleright \left(B_i(a) + \sum e{:}E(a).\ D_i(fe) \right)_{i \in I} \right) \ .$$

In other words, given type constructors $F(\vec{X}, Y)$ and $G(\vec{X})$ this construction defines the composite type constructor $F[G](\vec{X}) \equiv F(\vec{X}, G(\vec{X}))$.

Proposition 2.6 (ibid., proposition 6.1). *Composition of containers commutes with composition of functors thus:* $T_F[T_G] \cong T_{F[G]}$. □

This shows how composition of containers captures the composition of container functors. More generally, it is worth observing that a composition of containers of the form $- \circ - : \mathcal{G}_I \times \mathcal{G}_J^I \to \mathcal{G}_J$ reflecting composition of functors $\mathbb{C}^J \to \mathbb{C}^I \to \mathbb{C}$ can also be defined making containers into a bicategory with 0-cells the index sets I and the category of homs from I to J given by the container category $\mathcal{G}_I^J$ (Abbott, 2003, proposition 4.4.4).

3 Initial Algebras and W-Types

In this section we discuss the construction of initial algebras for container functors and the principles in the ambient category $\mathbb{C}$ used to construct them.

Initial algebras can be regarded as the fundamental building blocks used to introduce recursive datatypes into type theory. Initial algebras define "well founded" structures, which can be regarded as the expression of terminating processes.

Definition 3.1. *An* algebra *for a functor* $F: \mathbb{C} \to \mathbb{C}$ *is an object* $X \in \mathbb{C}$ *together with a morphism* $h: FX \to X$*; refer to X as the* carrier *of the algebra. An* algebra morphism $(X, h) \to (Y, k)$ *is a morphism* $f: X \to Y$ *satisfying the identity* $f \cdot h = k \cdot Ff$. *An* initial algebra *for F is then an initial object in the category of algebras and algebra morphisms.*

The following result tells us that initial algebras for a functor F are *fixed points* of F, and indeed the initial algebra is often called the least fixed point.

Proposition 3.2 (Lambek's Lemma). *Initial algebras are isomorphisms.* □

The following useful result about initial algebras tells us that initial algebras with parameters extend to functors, and so can be constructed "pointwise".

Proposition 3.3. *Given a functor* $F: \mathbb{D} \times \mathbb{C} \to \mathbb{C}$ *if each endofunctor* $F(X, -)$ *on* $\mathbb{C}$ *has an initial algebra* $(GX, \alpha X)$ *then G extends to a functor* $G: \mathbb{D} \to \mathbb{C}$ *and* α *to a natural transformation* $\alpha: F[G] \to G$. □

We can now define an operation μ constructing the least fixed point of a functors. If we regard a functor $F : \mathbb{D} \times \mathbb{C} \to \mathbb{C}$ as a type constructor $F(X, Y)$ then we can can regard the fixed points defined below as types.

Definition 3.4. *Given a functor* $F : \mathbb{D} \times \mathbb{C} \to \mathbb{C}$ *regarded as a type constructor* $F(X, Y)$ *define* $\mu Y.F(X, Y)$ *to be the initial algebra of the functor* $F(X, -)$.

To extend this definition of μ-**types** to containers observe that for containers $F \in \mathcal{G}_{I+1}$ and $G \in \mathcal{G}_I$ the operation $G \mapsto F[G]$, with $T_{F[G]}X \cong T_F(X, T_G X)$ is an endofunctor on $\mathcal{G}_I$. Thus given $F \in \mathcal{G}_{I+1}$ we will write μF for the initial algebra of $F[-] : \mathcal{G}_I \to \mathcal{G}_I$.

We will show in this paper that the functor $\mu : \mathcal{G}_{I+1} \to \mathcal{G}_I$ exists, and that the initial algebra of a container functor is a container functor.

W-Types

In Martin-Löf's Type Theory (Martin-Löf, 1984; Nordström et al., 1990) the building block for inductive constructions is the W-type. Given a family of constructors $A \vdash B$ the type $\mathsf{W}a : A.\, B(a)$ (or $\mathsf{W}_A B$) should be regarded as the type of "well founded trees" constructed by regarding each $a : A$ as a constructor of arity $B(a)$.

The standard presentation of a W-type is through one type forming rule, an introduction rule and an elimination rule, together with an equation. As the type theoretic development in this paper focuses entirely on categorical models, we take W-types to be *extensionally* defined. Indeed, extensional Type Theory as presented in Martin-Löf (1984) represents the canonical example of a Martin-Löf category.

Definition 3.5. *A type system* has W-types *iff it has a type constructor*

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash \mathsf{W}_A B} \qquad \text{(W-type)}$$

together with a constructor term

$$\Gamma,\, a : A,\, f : (\mathsf{W}_A B)^{B(a)} \vdash \mathrm{sup}(a, b) : \mathsf{W}_A B \qquad \text{(sup)}$$

and an elimination rule

$$\frac{\begin{array}{l}\Gamma,\, \mathsf{W}_A B \vdash C \\ \Gamma,\, a : A,\, f : (\mathsf{W}_A B)^{B(a)},\, g : \prod b : B(a).\, C(fb) \vdash h(a, f, g) : C(\mathrm{sup}(a, f))\end{array}}{\Gamma,\, w : \mathsf{W}_A B \vdash \mathrm{wrec}_h(w) : C(w)} \qquad \text{(wrec)}$$

satisfying the equation for variables $a : A$ *and* $f : (\mathsf{W}_A B)^{B(a)}$*:*

$$\mathrm{wrec}_h(\mathrm{sup}(a, f)) = h(a, f, \mathrm{wrec}_h \cdot f)\ .$$

Note that the elimination rule together with equality types ensures that wrec_h is unique. It is easy to see that the rule (wrec) implies that each $\mathsf{W}_A B$ is an initial algebra for $T_{A \triangleright B}$, and indeed the following theorem (see, for example, Abbott, 2003, theorem 5.2.2) allows us to identify W-types and initial algebras of containers.

Theorem 3.6. *W-types are precisely the initial algebras of container functors in one parameter:*

$$\mathsf{W}_A B \cong \mu X.\, \sum_A X^B = \mu X.\, T_{A \triangleright B} X \ . \qquad \square$$

We consider that this notion summarises the essence of Martin-Löf's Type Theory from a categorical perspective, hence the following definition.

Definition 3.7. *A* Martin-Löf category *is an extensive locally cartesian closed category with an initial algebra for every container functor (i.e. W-types).*

We know that W-types exist in toposes with natural numbers objects (Moerdijk and Palmgren, 2000, proposition 3.6) and in categories which are both locally cartesian closed and locally presentable (Abbott et al., 2003, theorem 6.8).

4 Initial Algebras of Containers

One consequence of theorem 3.6 is that in the presence of W-types we can immediately construct μ-types for containers in one parameter. However, the construction of a μ-type for a container in multiple parameters is a more delicate matter and will require the introduction of some additional definitions.

Let $F : \mathbb{C}^{I+1} \to \mathbb{C}$ be a container in multiple parameters, which we can write as

$$F(X,Y) \equiv T_{S \triangleright P,Q}(X,Y) = \sum s{:}S.\ \left(\prod_{i \in I} X_i^{P_i(s)}\right) \times Y^{Q(s)} = \sum_S \left(\prod_I X^P \times Y^Q\right).$$

The task is to compute $(A \triangleright B)$ such that $T_{A \triangleright B} X \cong \mu Y.\, F(X,Y)$. Clearly

$$A \cong T_{A \triangleright B} 1 \cong \mu Y.\ F(1,Y) \cong \mu Y.\ \sum s{:}S.\ Y^{Q(s)} \cong \mathsf{W}_S Q \ ,$$

but the construction of $\mathsf{W}_S Q \vdash B$ is more tricky.

In the rest of this paper we will ignore the index set I and write X^P for $\prod_I X^P$. In particular, this means that the family $B \in (\mathbb{C}/\mathsf{W}_S Q)^I$ will be treated uniformly (as if $I = 1$). It is a straightforward exercise to generalise the development to arbitrary index sets. We will therefore take

$$F(X,Y) \equiv \sum_S (X^P \times Y^Q) \ .$$

To simplify the algebra of types we will write $S, A^Q \vdash P + \sum_Q \epsilon^* B$ as an abbreviation for the type expression (where ϵ is the evaluation map $A^Q \times Q \to A$):

$$s{:}S,\ f{:}A^{Q(s)} \vdash\ P(s) + \sum q{:}Q(s).\ B(fq) \ .$$

For conciseness, write the initial algebra on $A = \mathsf{W}_S Q$ as $\psi : \sum_S A^Q \to A$.

Proposition 4.1. *Given the notation above, if* $\mathsf{W}_S Q \vdash B$ *is equipped with an fibred family of isomorphisms:*

$$S, A^Q \vdash\ \varphi : P + \sum_Q \epsilon^* B \cong \psi^* B$$

then $T_{A \triangleright B} X \cong \mu Y.\, F(X,Y)$.

Proof. First we show that each $T_{A \triangleright B} X$ is an $F(X, -)$ algebra thus:

$$\begin{aligned} F(X, T_{A \triangleright B} X) &= \sum_S \left(X^P \times \left(\sum_A X^B \right)^Q \right) \cong \sum_S \left(X^P \times \sum_{AQ} \prod_Q X^{\epsilon^* B} \right) \\ &\cong \sum_S \sum_{AQ} \left(X^P \times \prod_Q X^{\epsilon^* B} \right) \cong \sum_S \sum_{AQ} X^{P + \sum_Q \epsilon^* B} \\ &\overset{\varphi^{-1}}{\cong} \sum_S \sum_{AQ} X^{\psi^* B} \overset{(\psi, \mathrm{id})}{\cong} \sum_A X^B = T_{A \triangleright B} X \ . \end{aligned}$$

With variables $s : S$, $g : X^{P(s)}$ and $h : \left(\sum_A X^B \right)^{Q(s)}$ note that we can decompose h into components $\pi \cdot h : A^{Q(s)}$ and $\pi' \cdot h : \prod q : Q(s). X^{B(\pi h q)}$ and so the algebra morphism $in : F(X, T_{A \triangleright B} X) \to T_{A \triangleright B} X$ can be conveniently written as

$$in(s, g, h) = (\psi(s, \pi \cdot h), [g; \pi' \cdot h] \cdot \varphi^{-1}) \ ;$$

conversely, given variables $s : S$, $f : A^{Q(s)}$ and $k : X^{B(\psi(s,f))}$, similarly note that $k \cdot \varphi \cdot \kappa'$ can be regarded as a term of type $\prod q : Q(s). X^{B(fq)}$ and so we can write

$$in^{-1}(\psi(s, f), k) = (s,\ k \cdot \varphi \cdot \kappa,\ (f, k \cdot \varphi \cdot \kappa')) \ .$$

To show that *in* is an *initial* $F(X, -)$-algebra we need to construct from any algebra $\alpha : F(X, Y) \to Y$ a unique map $\overline{\alpha} : T_{A \triangleright B} X \to Y$ satisfying the algebra morphism equation $\overline{\alpha} \cdot in = \alpha \cdot F(X, \overline{\alpha})$:

$$\begin{array}{ccc} F(X, T_{A \triangleright B} X) & \xrightarrow{in} & T_{A \triangleright B} X \\ \downarrow{\scriptstyle F(X, \overline{\alpha})} & & \downarrow{\scriptstyle \overline{\alpha}} \\ F(X, Y) & \xrightarrow[\alpha]{} & Y \end{array} \ .$$

The map $\overline{\alpha}$ can be transposed to a term $A \vdash \widetilde{\alpha} : X^B \Rightarrow Y$ which we will construct by induction on $A = \mathsf{W}_S Q$. Given $s : S$, $f : A^{Q(s)}$ and $k : X^{B(\psi(s,f))}$ construct $g \equiv k \cdot \varphi \cdot \kappa : X^{P(s)}$ and $h \equiv k \cdot \varphi \cdot \kappa' : \prod q : Q(s). X^{B(fq)}$. In this context define $H(s, f, \beta)(k) \equiv \alpha(s, g, \beta(h))$ and compute

$$\begin{aligned} \widetilde{\alpha}(\psi(s, f))(k) &= \overline{\alpha}(\psi(s, f), k) = \overline{\alpha} \cdot in \cdot (s, g, (f, h)) \\ &= \alpha \cdot F(X, \overline{\alpha}) \cdot (s, g, (f, h)) = \alpha(s, g, \overline{\alpha} \cdot (f, h)) \\ &= \alpha(s, g, (\widetilde{\alpha} \cdot f)(h)) = H(s, f, \widetilde{\alpha} \cdot f)(k) \ . \end{aligned}$$

This shows that $\widetilde{\alpha} = \mathbf{wrec}_H$ and thus that $T_{A \triangleright B} X$ is an $F(X, -)$-initial algebra. □

Note that we can discover from the proposition above that B is defined uniquely up to isomorphism (since $\mu Y. F(X, Y)$ is unique). The intuitive reason for this is that B corresponds to the type of paths in a finite tree, and consequently there cannot be any infinite paths. The structure of the functor $X \mapsto P + \sum_Q \epsilon^* X$ respects the structure of the initial algebra ψ, which forces B to be unique. Compare this with Wraith's theorem (Johnstone, 1977, theorem 6.19), for the special case $A = \mathbb{N}$.

Of course, it remains to prove the hypothesis of the theorem above, that a family $A \vdash B$ with the given isomorphism φ exists; we do this below in proposition 5.1.

5 Constructing a Fixed Point over an Initial Algebra

Proposition 4.1 relies on the hypothesis that the functor $X \mapsto P + \sum_Q \epsilon^* X$ has a fixed point "over" the initial algebra $\psi : T_{S \triangleright Q} A \to A$, or in other words there exists a B such that $P + \sum_Q \epsilon^* B \cong \psi^* B$. This fixed point does indeed exist, as a *subtype* of a W-type.

Proposition 5.1. *For each fixed point $\psi : T_{S \triangleright Q} A \cong A$ there exists an object $A \vdash B$ such that there is an isomorphism:*

$$S, A^Q \vdash P + \sum_Q \epsilon^* B \cong \psi^* B \ .$$

Proof. Write $S, A^Q \vdash \varphi : P + \sum_Q \epsilon^* B \to \psi^* B$ for the isomorphism that we wish to construct. As already noted, we cannot directly appeal to W-types to construct this fixed point, so the first step is to create a fixed point equation that we *can* solve. Begin by "erasing" the type dependency of B and construct (writing $\sum_Q Y \cong Q \times Y$, etc)

$$\begin{aligned} \widehat{B} &\equiv \mu Y . \sum_S \sum_{A^Q} (P + Q \times Y) \cong \mu Y . \left(\sum_S (A^Q \times P) + \left(\sum_S (A^Q \times Q) \right) \times Y \right) \\ &\cong \mathrm{List} \left(\sum_S (A^Q \times Q) \right) \times \sum_S (A^Q \times P) \ ; \end{aligned}$$

there is no problem in constructing arbitrary lists in $\mathbb{C}$ so $\widehat{B}$ clearly exists.

The task now is to select the "well-formed" elements of $\widehat{B}$. A list in $\widehat{B}$ can be thought of as a putative path through a tree in $\mu Y . T_{S \triangleright P,Q}(X, Y)$; we want $B(a)$ to be the set of all valid paths to X-substitutable locations in the tree.

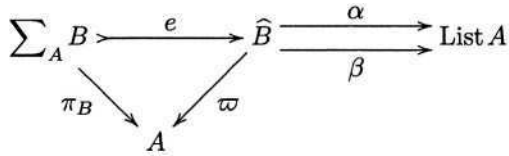
An element of $\widehat{B}$ can be conveniently written as a list followed by a tuple thus

$$([(s_0, f_0, q_0), \ldots, (s_{n-1}, f_{n-1}, q_{n-1})], (s_n, f_n, p))$$

for $s_i : S$, $f_i : A^{Q(s_i)}$, $q_i : Q(s_i)$ and $p : P(s_n)$. The condition that this is a well formed element of $B(\psi(s_0, f_0))$ can be expressed as the n equations

$$f_i(q_i) = \psi(s_{i+1}, f_{i+1}) \quad \text{for } i < n$$

which can be captured as an equaliser diagram



where α, β and ϖ are defined inductively on $\widehat{B}$ as follows (and $\pi_B \equiv \varpi \cdot e$):

$$\begin{aligned} \alpha(\mathsf{nil}, p') &= \mathsf{nil} & \alpha(\mathsf{cons}((s, f, q), l), p') &= \mathsf{cons}(fq, \alpha(l, p')) \\ \varpi(\mathsf{nil}, (s, f, p)) &= \psi(s, f) & \varpi(\mathsf{cons}((s, f, q), l), p') &= \psi(s, f) \\ \beta(\mathsf{nil}, p') &= \mathsf{nil} & \beta(\mathsf{cons}(b, l), p') &= \mathsf{cons}(\varpi(l, p'), \beta(l, p')) \ . \end{aligned}$$

The property that $b : \widehat{B}$ is an element of B can be written $b : B(\varpi b)$ and using the equations above we can establish:

$$(\mathsf{nil}, (s, f, p)) : B(\psi(s, f)) \tag{1}$$

$$fq = \varpi(l, p') \wedge (l, p') : B(fq) \implies (\mathsf{cons}((s, f, q), l), p') : B(\psi(s, f)) \ . \tag{2}$$

The converse to (2) also holds, since $(\mathsf{cons}((s, f, q), l), p') : B(\psi(s, f)) \iff \mathsf{cons}(fq, \alpha(l, p')) = \mathsf{cons}(\varpi(l, p'), \beta(l, p')) \iff fq = \varpi(l, p') \wedge (l, p') : B(fq)$.

The isomorphism $\widehat{\varphi} : \sum_S \sum_{A^Q} (P + Q \times \widehat{B}) \cong \widehat{B}$ can now be used to construct the isomorphism φ for B. Writing an element of $\sum_S \sum_{A^Q} (P + Q \times \widehat{B})$ as $(s, f, \kappa p)$ or $(s, f, \kappa'(q, b))$, the function $\widehat{\varphi}$ can be computed thus:

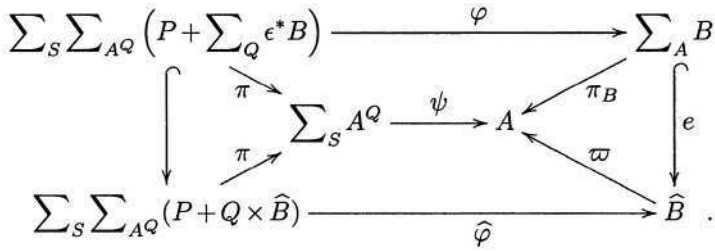
$$\begin{array}{ccc} \sum_S \sum_{A^Q} (P + Q \times \widehat{B}) & \overset{\widehat{\varphi}}{\cong} & \begin{array}{c} \mathrm{List}(\sum_S (A^Q \times Q)) \\ \times \sum_S (A^Q \times P) \end{array} = \widehat{B} \\ (s, f, \kappa p) & \longleftrightarrow & (\mathsf{nil}, (s, f, p)) \\ (s, f, \kappa'(q, (l, p'))) & \longleftrightarrow & (\mathsf{cons}((s, f, q), l), p') \ . \end{array}$$

To show that $\widehat{\varphi}$ restricts to a morphism $\varphi : P + \sum_Q \epsilon^* B \to \psi^* B$ we need to show for each $s : S$ and $f : A^Q$ that $x : (P(s) + \sum q : Q(s).\, B(fq))$ implies $\widehat{\varphi}(s, f, x) : B(\psi(s, f))$.

When $x = \kappa p$ we immediately have $\widehat{\varphi}(s, f, \kappa p) = (\mathsf{nil}, (s, f, p)) : B(\psi(s, f))$ by (1) above. Now let $(s, f, \kappa'(q, (l, p')))$ be given with $(l, p') : B(fq)$ (which means, in particular, that $\varpi(l, p') = fq$) and consider the equation $\widehat{\varphi}(s, f, \kappa'(q, (l, p'))) = (\mathsf{cons}((s, f, q), l), p')$, then by (2) this is also in $B(\psi(s, f))$. Thus $\widehat{\varphi}$ restricts to

$$s : S, f : A^{Q(s)} \vdash \varphi_{s,f} : P(s) + \sum q : Q(s).\ B(fq) \longrightarrow B(\psi(s, f)) \ .$$

We have, in effect, constructed φ making the diagram below commute:



To show that φ is an isomorphism we need to show that $\widehat{\varphi}^{-1}$ restricts to an inverse to φ. As before we can analyse $b : B(\psi(s, f))$ into two cases, and show that in both cases $\widehat{\varphi}^{-1} b : P(s) + \sum q : Q(s).\, B(fq)$.

When $b = (\mathsf{nil}, (s, f, p))$ then $\widehat{\varphi}^{-1} b = (s, f, \kappa p)$ which can be regarded as an element of $P(s)$. When $b = (\mathsf{cons}((s, f, q), l), p')$ and so $\widehat{\varphi}^{-1} b = (s, f, \kappa'(q, (l, p')))$ it is enough to observe that $b : B(\psi(s, f))$ implies $(l, p') : B(fq)$ and hence $\widehat{\varphi}^{-1} b$ arises from an element of $\sum q : Q(s).\, B(fq)$. □

Combining 4.1 and 5.1 we obtain as a corollary:

Corollary 5.2. *If $\mathbb{C}$ has W-types then containers are closed under the construction of μ-types.* □

6 Strictly Positive Inductive Types

We now have enough machinery in place to observe that all strictly positive types can be described as containers.

Definition 6.1. *A strictly positive inductive type (SPIT) in n variables (Abel and Altenkirch, 2000) is a type expression (with type variables $X_1, \ldots, X_n$) built up inductively according to the following rules:*

- *if K is a constant type (with no type variables) then K is a SPIT;*
- *each type variable X_i is a SPIT;*
- *if F, G are SPITs then so are $F + G$ and $F \times G$;*
- *if K is a constant type and F a SPIT then $K \Rightarrow F$ is a SPIT;*
- *if F is a SPIT in $n+1$ variables then $\mu X.F$ is a SPIT in n variables (for X any type variable).*

Note that the type expression for a SPIT F can be interpreted as a functor $F : \mathbb{C}^n \to \mathbb{C}$, and indeed we can see that each strictly positive type corresponds to a container in $\mathcal{G}_n$.

Let strictly positive types F, G be represented by containers $(A \triangleright B)$ and $(C \triangleright D)$ respectively, then the table below shows the correspondence between strictly positive types and containers.

$$K \mapsto (K \triangleright 0) \qquad X_i \mapsto (1 \triangleright (\delta_{i,j})_{j \in I})$$

$$F + G \mapsto (A + C \triangleright B \dot{+} D) \qquad F \times G \mapsto (a{:}A,\, c{:}C \triangleright B(a) \times D(c))$$

$$K \Rightarrow F \mapsto \left(f{:}A^K \triangleright \sum k{:}K.\ B(fk)\right)$$

As we have seen in this paper the construction of fixed points can be described in a uniform way. Let F be represented by $(S \triangleright P, Q) \in \mathcal{G}_{I+1}$, then for each fixed point $\psi : T_{S \triangleright Q} A \cong A$ of $T_{S \triangleright Q}$ we have constructed in proposition 5.1 an isomorphism over ψ, written here as $A \vdash B_A$, of the form

$$s{:}S,\ f{:}A^{Q(s)} \vdash \varphi{:}P(s) + \sum q{:}Q(s).\ B_A(fs) \longrightarrow B_A(\psi(s,f))\ ;$$

we can now define

$$\mu Y.\ F \mapsto (\mathsf{W}_S Q \triangleright B_{\mathsf{W}_S Q})\ .$$

Our development can be summarised by the following:

Theorem 6.2. *All strictly positive inductive types can be represented within a Martin-Löf category.*

Proof. This is a consequence of corollary 5.2 and the discussion above. □

7 Discussion and Further Work

An important extension of the work presented here is to include coinductive types, $\nu X.F$, corresponding to terminal coalgebras, to cover non-well founded data structures such as streams (Stream $A = \nu X.A \times X$), which are used extensively in lazy functional programming. We have also established (see Abbott, 2003, p. 78 and Abbott et al., 2004), that Martin-Löf categories are closed under ν-types—this can be reduced to constructing the dual of W-types which we dub M-types.

Another interesting extension would be to consider inductive and coinductively defined families (such as vectors or simply typed λ-terms). Again, we conjecture that it should be possible to represent those within Martin-Löf categories. This result would provide further evidence establishing that these categories provide a convenient and concise base for intuitionistic Type Theory.

References

M. Abbott. *Categories of Containers.* PhD thesis, University of Leicester, 2003.

M. Abbott, T. Altenkirch, and N. Ghani. Categories of containers. In *Proceedings of Foundations of Software Science and Computation Structures,* volume 2620 of *Lecture Notes in Computer Science,* 2003.

M. Abbott, T. Altenkirch, and N. Ghani. Representing strictly positive types. Presented at APPSEM annual meeting, invited for submission to Theoretical Computer Science, 2004.

A. Abel and T. Altenkirch. A predicative strong normalisation proof for a λ-calculus with interleaving inductive types. In *Types for Proof and Programs, TYPES '99,* volume 1956 of *Lecture Notes in Computer Science,* 2000.

P. Dybjer. Representing inductively defined sets by wellorderings in Martin-Löf's type theory. *Theoretical Computer Science,* 176:329–335, 1997.

P. Dybjer and A. Setzer. A finite axiomatization of inductive-recursive definitions. In *Typed Lambda Calculus and Applications,* pages 129–146, 1999.

P. Dybjer and A. Setzer. Indexed induction-recursion. *Lecture Notes in Computer Science,* 2183, 2001.

N. Gambino and M. Hyland. Wellfounded trees and dependent polynomial functors. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs (TYPES 2003),* Lecture Notes in Computer Science, 2004.

M. Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *CSL,* pages 427–441, 1994.

B. Jacobs. *Categorical Logic and Type Theory.* Number 141 in Studies in Logic and the Foundations of Mathematics. Elsevier, 1999.

P. T. Johnstone. *Topos Theory.* Academic Press, 1977.

P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, Napoli, 1984.

I. Moerdijk and E. Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic,* 104:189–218, 2000.

B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory.* Number 7 in International Series of Monographs on Computer Science. Oxford University Press, 1990.

T. Streicher. *Semantics of Type Theory*. Progress in Theoretical Computer Science. Birkhäuser Verlag, 1991.

Algorithms for Multi-product Pricing

Gagan Aggarwal*, Tomás Feder**, Rajeev Motwani***, and An Zhu†

Computer Science Department, Stanford University, Stanford, CA 94305.
{gagan,rajeev,anzhu}@cs.stanford.edu, tomas@theory.stanford.edu

Abstract. In the information age, the availability of data on consumer profiles has opened new possibilities for companies to increase their revenue via data mining techniques. One approach has been to strategically set prices of various products, taking into account the profiles of consumers. We study algorithms for the multi-product pricing problem, where, given consumer preferences among products, their budgets, and the costs of production, the goal is to set prices of multiple products from a single company, so as to maximize the overall revenue of the company. We present approximation algorithms as well as negative results for several variants of the multi-product pricing problem, modeling different purchasing patterns and market assumptions.

1 Introduction

Through interaction with online consumers, e-commerce websites can gather data reflecting consumer preferences. Such data allows significant revenue increases through strategic price setting via sophisticated analytical tools. While the airline and hotel industry were the traditional users of revenue management [9,14], corporations in other verticals in retail and manufacturing have recently started employing intelligent pricing strategies to boost their bottom lines. For instance, Dell quotes different prices to different market segments for the same product, enabling Dell to increase its market share and profitability [10]. Other documented examples include Ford [2] and ShopKo Stores [11].

Motivated by the availability of such data, Rusmevichientong, Van Roy, and Glynn formulated the non-parametric multi-product pricing problem [13]. In multi-product pricing, given demands and production costs, the problem is to determine an assignment of prices to products that maximizes overall revenue or profit [6,7]. When faced with choice between multiple substitutable products, consumers may be indifferent between two or more products and substitute one product for another. The problem of modeling substitutability among products and determining optimal pricing in that context remains a challenging open problem in this research area. As noted in [13], in order to capture substitutability, in most models, consumer demand functions are generally assumed to take on specific parametric forms [3]. However, these parametric forms may not reflect

* Supported in part by a SGF fellowship from Stanford and NSF Grant EIA-0137761.
** 268 Waverley St., Palo Alto, CA 94301.
*** Supported in part by NSF Grant IIS-0118173 and EIA-0137761, an Okawa Foundation Research Grant, and grants from Microsoft and Veritas.

† Supported in part by a GRPW fellowship from Bell Labs, Lucent Technologies, and NSF Grant EIA-0137761.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 72–83, 2004.

the true demand function accurately. We adopt the non-parametric approach proposed in [13].

The non-parametric approach employs large quantities of consumer data in order to reduce reliance on rigid parametric models, where the consumer data is obtained from Internet sites. A concrete example is the General Motors' Auto Choice Advisor website [4]. This website is set up to advise potential purchasers of cars (of all brands) on products that meet their requirements. The website asks the users various questions about their preferences with respect to cars, such as model, budget, primary use, etc., and then recommends some cars that satisfy their requirements. Thus, GM has access to large quantities of data that reflect consumer preferences. This data can be used by GM to optimize the pricing of their own vehicles. Based on this scenario, Rusmevichientong et al. [12,13] introduced the non-parametric multi-product pricing model, which we describe next.

Consider the problem of pricing n different products $\mathcal{P}$ indexed by $A = \{1, \ldots, n\}$, based on a set of data samples, $(R_1, B_1), \ldots, (R_m, B_m)$, each associated with one of m different consumers that have visited an e-commerce website. For each j, let $R_j = \pi(A)$ be the preference ordering of the j^{th} consumer over all n products, where products ranked lower are preferred over products ranked higher. Further, let b_{ij} be the *budget* of consumer j for product i, i.e. the maximum amount consumer j is willing to spend for product i. Also, let $B_j = \{b_{ij}, \forall 1 \leq i \leq n\}$ denote the sequence of budgets of the j^{th} consumer. We assume consumers are consistent, i.e., the order of their budgets for various products obeys their preference ordering. Each sample (R_j, B_j) represents all available information about the j^{th} consumer. We further assume that, given that the products are priced at $P = \{P_1, \ldots, P_n\}$, the j^{th} consumer will purchase the lowest-ranked product in their preference list which she can afford, i.e., they will buy product k, where $k = \arg\min_{B_{ij} \geq P_i} R_j(i)$. This model of consumer behavior is termed *Rank-Buying*. In addition, we assume that there is a *Price-Ladder (PL)* constraint, i.e., we know the relative ordering of the prices of various products. Such a constraint can usually be deduced from market studies/policies and manufacturing costs. Assuming each consumer buys only one product, the objective is to set the product prices so as to maximize the revenue.

Rusmevichientong et al. [12,13] studied this basic model and presented various heuristics. However, they left open the problem of designing algorithms with provable quality guarantees. In this paper, we present a PTAS for this problem. In addition, we consider some interesting variants of the original model:

1. Given the prices, suppose the consumer will buy the most expensive (least expensive) product that she can afford. These variants, called the *Max-Buying* (respectively, *Min-Buying*) models, were proposed in [12,13]. Assuming a price ladder, the Min-Buying model has been shown to be solvable in polynomial time using dynamic programming [12,13].
2. In another variant, referred to as *No-Price-Ladder* (*NPL*), we do not assume a price ladder on the products.
3. In yet another variant, we are given costs of production, and the goal is to maximize the profit rather than the revenue.

4. We also consider the model where there are a limited number of copies of each product – inventory/production constraints might lead to such a condition.

We present algorithms and negative results for various combinations of these variations on the original model. Our results are summarized in the following table. The first column lists the particular model being considered in terms of price ladder or no price ladder, the buying behavior, and limited versus unlimited copies of products. The second column gives upper and lower bounds on the approximation bounds achievable in polynomial time. The last column gives extensions of the model where the algorithm remains applicable.

Model	Upper [Lower] Bounds	Extensions
PL & Rank-Buying	PTAS	Max-Buying instead of Rank-Buying
PL & Max-Buying & Limited-Copies	4	Consumers arrive online
NPL & Max-Buying	1.59 [16/15]	Maximize Profit instead of Revenue
NPL & Min-Buying	$\log m$ [$1+\epsilon$]	Upper bound holds for all models

The rest of the paper is organized as follows. In Sect. 2, we present the PTAS and the 4-approximation algorithm for the Rank-Buying with Price-Ladder problem and related models. In Sect. 3, we discuss the No-Price-Ladder model. We present the 1.59-approximation algorithm and the hardness result for the Max-Buying with No-Price-Ladder problem, and the hardness result for the case of Min-Buying with No-Price-Ladder. Section 4 presents the $O(\log m)$-approximation algorithm that works for all models and discusses some related issues. Finally, we conclude with some open problems in Sect. 5.

2 Rank/Max Buying with Price-Ladder

We first show that when there is a Price-Ladder constraint, the Rank-Buying model can be reduced to the Max-Buying model[1].

Lemma 1. *We can transform an instance I in the Rank-Buying with Price-Ladder model to an equivalent instance I' in the Max-Buying with Price-Ladder model.*

Proof. In the instance *I*, consider a consumer j and any two products i and i' such that $P_i \leq P_{i'}$ and $R_j(i) < R_j(i')$, where $R_j(i)$ denotes the position of product i in j's preference list, with the most preferred product ranked lowest. For such a pair of products, if the consumer could afford both products, Max-Buying would lead the consumer to buy i', while Rank-Buying would let her buy i, a product which is cheaper and more preferred. In order to reduce Rank-Buying to Max-Buying, we would like to eliminate all such product pairs without altering the solution to the instance. We note that for such product pairs i, i', since the budgets are assumed to be in the same order as the rank, the

[1] Note that the reduction is valid only in this setting, and may not work for other extensions and variations in this paper.

budget for i is higher than the budget for i'. Since $P_i \leq P_{i'}$, the consumer can afford i, whenever she can afford i', and since the consumer buys by rank, she would buy i rather than i'. Thus, we see that the consumer never buys i'. So we can reduce the budget $b_{i'j}$ to 0, without affecting the outcome in the Rank-Buying model. By repeating this for every product pair with conflicting rank and price orders, we can create an equivalent instance I' in which the rank order (equivalently, budget order) conforms to the price order for each consumer. Consequently, Max-Buying gives the same outcome as Rank-Buying on this new instance I', which in turn gives the same outcome as Rank-Buying on the original instance I. □

We now present a PTAS for the Max-Buying model, which along with the above transformation, will give us a PTAS for the Rank-Buying model. We begin by noting that given any solution assigning prices to products, we can transform the solution to one in which the prices are taken from the set of distinct budget values, without decreasing the revenue generated by the solution. This general observation holds for all models studied in this paper.

Assume that the products are listed in the order of decreasing prices (as specified by the Price-Ladder), i.e., $P_1 \geq P_2 \geq \cdots \geq P_n$. We first relax the problem in two ways:

1. Let $\mathcal{B} = \max_{i,j} B_{ij}$. We discretize the search space of possible prices into values of the form $d_i = \mathcal{B}/s^i$, where $s > 1$ will be chosen later.
2. We relax the constraint that a consumer can purchase at most one product. Instead, we allow the consumer to buy multiple products. However, if j buys a product at price p, then she is not allowed to buy any other product with price p', where $p \leq p' < s^k p$, where the integer k is chosen later.

Consider the modified instance. By the first relaxation, we lose a factor of s; by the second relaxation, we lose a factor of $1 + \frac{1}{s^k} + \frac{1}{s^{2k}} + \frac{1}{s^{3k}} + \cdots \leq \frac{1}{1-s^{-k}}$. Combining the two error factors gives a factor of $\frac{s}{1-s^{-k}}$, which is minimized at $s = (k+1)^{1/k}$, where it equals $(k+1)^{\frac{1}{k}}(1+\frac{1}{k}) = 1 + \frac{\log k}{k}(1+o(1))$. This approximation factor is 4 for $k = 1$ and can be made $1+\epsilon$, for any constant ϵ, by taking a suitably large k.

We next show how to obtain the optimal solution to this modified problem by dynamic programming. Define $F(i, x_i, x_{i-1}, x_{i-2}, \ldots, x_{i-k+1})$ to be the maximum revenue generated by only considering products with prices ranging from d_0 (=$\mathcal{B}$) to d_i, with x_j being the last product with price set to d_j or higher. Note that in order to respect the price ladder, x_j must precede x_{j+1} or be equal to it in the price ladder. To compute $F(i+1, x_{i+1}, x_i, \ldots, x_{i-k+2})$, we enumerate through each choice of x_{i-k+1} (the number of choices is at most x_{i-k+2}). Let $C(x_{i+1}, x_i, \ldots, x_{i-k+2}, x_{i-k+1})$ denote the number of consumers that satisfy the following two conditions:

- For $i-k+2 \leq j \leq i$, if prices of all products $x_{j-1}+1, \ldots, x_j$ are set to d_j, the consumer will not be able to afford any of these products.
- If the price of products $x_i+1, \ldots, x_{i+1}$ was set to d_{i+1}, the consumer will be able to afford at least one of these products.

Define $G(i+1, x_{i+1}, x_i, \ldots, x_{i-k+1}) = F(i, x_i, \ldots, x_{i-k+1}) + d_{i+1} C(x_{i+1}, x_i, \ldots, x_{i-k+1})$. In other words, $G(i+1, x_{i+1}, \ldots, x_{i-k+2}, x_{i-k+1})$

is the sum of $F(i, x_i, \ldots, x_{i-k+1})$ and the payoff generated by consumers who buy products at price d_{i+1}, while ensuring that they have not bought products of price d_j, for $i-k+2 \le j \le i$. We obtain the following recurrence:

$$F(i+1, x_{i+1}, \ldots, x_{i-k+2}) = \max_{0 \le x_{i-k+1} \le x_{i-k+2}} G(i+1, x_{i+1}, \ldots, x_{i-k+2}, x_{i-k+1}).$$

This leads to the following theorem.

Theorem 1. *The Max-Buying with Price-Ladder problem has a PTAS. It can be approximated within a factor of* $(k+1)^{\frac{1}{k}}(1+\frac{1}{k}) = 1 + \frac{\log k}{k}(1+o(1))$, *in time* $\binom{n+k-1}{k} n^2 m^2 = O(n^{k+2}m^2)$ *and space* $O(n^{k+1}m)$. *With an additional approximation factor of* $1 + \frac{1}{\text{poly}(n)}$, *the complexity can be improved to time* $O(n^{k+1}\log n)$ *and space* $O(n^k \log n)$.

Proof. The optimal solution will be the best value of $F(l, \ldots)$, where d_l is the smallest price considered. The number of choices for the arguments of $F(i, x_i, x_{i-1}, \ldots, x_{i-k+1})$ is ran for the value of i (since there are at most $O(nm)$ distinct B_{ij}'s, and it is easy to see that there exists an optimal solution where the set of distinct prices is a subset of the set of distinct budget values) and $\binom{n+k-1}{k}$ for the x_j's. Each value of F requires $O(nm)$ computation time and $O(1)$ space, giving the stated bounds. If we restrict the smallest allowed price to $\frac{\mathcal{B}}{n\text{poly}(n)}$, we incur an additional $(1+\frac{1}{\text{poly}(n)})$-approximation factor, but reduce the number of choices for i from nm to $O(\log n)$, giving the better time and space bounds. □

In Appendix A, we present another algorithm which requires only linear space, but has a worse (super-polynomial) time bound.

We now consider the Max-Buying with Price-Ladder model with the additional constraint that there are only N_i copies of the i^{th} product. Since the number of copies of the products are limited, we need to redefine the optimal revenue generated by a setting of prices. Given a price setting and an order in which the consumers arrive, we assume that when a consumer arrives, she buys the most expensive product she can afford that is still available. We define the maximum revenue for a setting of prices as the payoff generated by the best arrival order of consumers (or the best allocation of the limited copies of products to consumers in the offline setting). In the more realistic case of consumers arriving in an arbitrary order, the payoff generated is within factor 2 of the best possible allocation as shown by the following lemma.

Lemma 2. *Let OPT denote the revenue generated by the best allocation. If* $\mathcal{R}(\pi)$ *denotes the revenue generated by arrival order* π *of consumers, then* $\mathcal{R}(\pi) \ge \frac{1}{2}OPT \quad \forall \pi$.

Proof. Let $A(i)$ (respectively, $B(i)$) be the set of consumers in the optimal (respectively, π) allocation who buy product i at price P_i. Consider those products i for which we have $|A(i)| > |B(i)|$. Since some copies of product i are still left over at the end of the π-allocation, the consumers in $A(i) - B(i)$ must all have bought some product with price at least P_i. So we charge the extra revenue generated by product i under optimal allocation to the revenue generated by these consumers under allocation π. Since each consumer gets charged at most once, the extra revenue generated by optimal allocation is no more than the total revenue generated by π, and hence the lemma. We note that this bound of 2 is tight. □

Theorem 2. *The Limited-Copies variant can be approximated to be a factor of 4, giving an 8-approximation algorithm for the online case.*

Proof. We use the same relaxation techniques as in the PTAS above. We set $k = 1$, which leads to the relaxation where a consumer is allowed to buy one product at every price d_i. The prices differ from each other by powers of 2. We set up a recursion for $F(i, x_i)$ (we enumerate over all possible values of $x_{i-1} \leq x_i$, adding the payoff from setting the price of products $x_{i-1}+1, \ldots, x_i$ to d_i to the optimal payoff $F(i-1, x_{i-1})$ from prices d_{i-1} and higher, and take the maximum over all these values to get $F(i, x_i)$). This gives us an approximation factor of 4. The reason higher values of k do not work lies in the difficulty of recursion in the dynamic programming. If we try to set up a recursion for $F(i, x_i, x_{i-1})$ instead of $F(i, x_i)$, the values x_i and x_{i-1} are not enough (in the limited copies scenario) to determine the products available at price d_{i-1}, and hence not enough to determine the set of consumers that buy products with price d_i. □

The Max-Rank model can be extended to take into account the competitors' products and the PTAS works in that case as well. In addition to our products $A = \{1, \ldots, n\}$, the competitors produce the remaining products $\overline{A} = \{n+1, \ldots, N\}$. Each consumer has a complete ranking list and budgets for all N products. In addition, the prices of the competitors' products $P_{n+1}, \ldots, P_N$ are specified. Again, each consumer buys the lowest-ranked product that she can afford. If a consumer buys a competitors' product, then we get no revenue from that consumer. The objective is to maximize the revenue generated by our products. We can reduce any instance that involves competitors' products to an instance limited only to our products. For each consumer j, we can find the lowest-ranked competitors' product that she can afford, say C_j. If any of our products is ranked higher than C_j in j's preference list, then j will never buy that product. On the other hand, if she can afford a product ranked lower than C_j, then she will buy it. Thus, it is sufficient and necessary to modify j's preference list to include only those products that are ranked lower than C_j. This model assumes that the competitors do not respond to our pricing decisions. The detailed proof of the following lemma can be found in [12].

Lemma 3. *Any problem instance in the Max-Rank with Price-Ladder model that includes competitors' products can be reduced to one without any competitors' products, without changing the value of the optimal solution.*

3 The No-Price-Ladder Model

We now study a model where no ordering constraints are specified on the prices of products *(No-Price-Ladder)*. We first study the Max-Buying with No-Price-Ladder problem and give a 1.59-approximation algorithm for the problem. We also show that it is NP-hard to approximate this problem better than 16/15. Then, we discuss the Min-Buying with No-Price-Ladder model. The Min-Buying with Price-Ladder problem can be solved optimally by using dynamic programming [13]. However, removing the Price-Ladder constraint makes the problem hard.

3.1 An Algorithm for Max-Buying with No-Price-Ladder

The unordered nature of prices renders the previous approach ineffective. Instead, we use linear programming and rounding techniques to get a 1.59-approximation algorithm for this variant. We will also show how to derandomize this algorithm. Consider the following linear program:

$$\begin{aligned}
\textbf{Maximize}: \quad & \textstyle\sum_p y_{jp} p & & \\
\textbf{subject to}: \quad & \textstyle\sum_p x_{ip} = 1 & \forall i & \quad (1)\\
& y_{jp} \le \textstyle\sum_{p \le B_{ij}} x_{ip} & \forall j,p & \quad (2)\\
& \textstyle\sum_p y_{jp} \le 1 & \forall j & \quad (3)\\
& 0 \le x_{ip} \le 1 & & \\
& 0 \le y_{jp} \le 1 & &
\end{aligned}$$

Here x_{ip} indicates that product i is assigned price p and y_{jp} indicates that consumer j buys a product at price p. Clearly, if x_{ip} and y_{jp} take values in {0,1}, the objective function is exactly the revenue we are maximizing. Instead, we relax the constraints on x_{ip} and y_{jp} to allow fractional solutions. We round the fractional optimal solution to obtain an integer solution by assigning product i price p with probability x_{ip}.

Theorem 3. *The Max-Buying with No-Price-Ladder problem can be approximated in polynomial time within a factor* $\frac{e}{e-1} < 1.59$.

Proof. We introduce some notation first. Let $s_{jp} = \sum_{p \le B_{ij}} x_{ip}$ be the total amount of (fractional) products priced at p which consumer j can afford. Let $t_{jp} = \sum_{p \le p'} s_{jp'}$ be the total amount of products priced p or higher which consumer j can afford to buy. Let $z_{jp} = \sum_{p \le p'} y_{jp'}$, the total amount of products consumer j buys at a price p or higher.

We thus have the following relations: $y_{jp} \le s_{jp}$ and $z_{jp} \le t_{jp}$. Now, the probability that consumer j buys at price at least p is $q_{jp} = 1 - \prod_i (1 - \sum_{p \le p' \le B_{ij}} x_{ip'})$. Recalling the definition of t_{jp}, we have that $q_{jp} \ge 1 - (1 - \frac{t_{jp}}{n})^n > 1 - e^{-t_{jp}} \ge \min(t_{jp}, 1)(1 - e^{-1})$.

We now look at the optimal fractional solution. Consumer j buys at price at least p with fractional value exactly $r_{jp} = \min(t_{jp}, 1)$. This implies that q_{jp} majorizes $(1 - e^{-1}) r_{jp}$. Thus, we get that the expected value of our rounded solution is at least a $(1 - e^{-1})$ fraction of that of the optimal fractional solution, thus giving us an approximation factor of 1.59.

The algorithm can be derandomized by replacing any two $0 < x_{ip} < 1$ and $0 < x_{ip'} < 1$ by $x_{ip} + \epsilon$ and $x_{ip'} - \epsilon$, for a suitable ϵ. Since the expected payoff is a linear function of ϵ, either any positive ϵ makes the payoff nondecreasing, or else, any negative ϵ ensures the payoff does not decrease. We may select such an ϵ with the appropriate sign to obtain $x_{ip} = 0$ or $x_{ip'} = 0$. Repeatedly performing this transformation ensures $x_{ip} = 0$ or $x_{ip} = 1$ for all x_{ip}. □

We remark that this linear programming formulation can be extended to the model where the goal is to maximize profit instead of revenue. For each product i, let $c(i)$ denote the fixed manufacturing cost. We redefine x_{ip} to indicate whether product i is

assigned profit p, and y_{jp} to indicate whether consumer j buys a product with profit p. We substitute inequality (2) in our linear programming formulation with the following: $y_{jp} \leq \sum_{p:p+c(i)\leq B_{ij}} x_{ip}$. The rest of the analysis goes through as before.

3.2 Negative Result for Max-Buying with No-Price-Ladder

Consider a special case of the Max-Buying with No-Price-Ladder problem where each consumer j specifies a set S_j of products she is interested in. In addition, her budget $b_{ij} = B_j$ for $i \in S_j$, and $b_{ij} = 0$ otherwise. Also, $B_j \in \{a, b\}$, for all j. We call this the *Uniform-two-budget* problem. We show below that even this special case of the Max-Buying with No-Price-Ladder problem is MAX SNP-hard.

Theorem 4. *The Uniform-two-budget problem with Max-Buying and No-Price-Ladder cannot be approximated better than* 16/15 *unless P = NP. There exists a polynomial time algorithm to approximate it within* 1/0.78.

Proof. If there are only two distinct budget values $a > b$, then the only prices used in an optimal solution are a and b. A consumer with budget a will always spend b, and may or may not spend the additional $a - b$. For every product i, consider a boolean variable x_i with $x_i = 1$ if i has price a and $x_i = 0$ if i has price b. Since we are considering the Max-Buying setting, a consumer j with budget a will pay the additional $a - b$ if the disjunction of x_i for the products in S_j holds, while a consumer j with budget b will pay b if the disjunction of the $\overline{x_i}$ for the products in S_j holds. The problem is thus an instance of MAX-SAT with disjunctions of positive literals having weight $a - b$ and disjunctions of negative literals having weight b. Since the MAX-SAT problem can be approximated within 1/0.78 [1], this gives us an algorithm for solving the Uniform-two-budget case with an approximation factor of 1/0.78.

For the hardness result, we reduce MAX-3SAT, which is hard to approximate within $\frac{8}{7} - \epsilon$ for any $\epsilon > 0$ (see [5]), to our problem. Consider an instance of MAX-3SAT. Replace clauses that have both positive and negative literals by two clauses that have only positive or only negative literals, such that the original clause is satisfied if both the new clauses are satisfied. For example, the clause $x \vee y \vee \overline{z}$ is replaced by clauses $x \vee y \vee t$ and $\overline{t} \vee \overline{z}$. Since the number of clauses at most doubles, the modified problem is hard to approximate within $\frac{16}{15} - \epsilon$. Now we reduce this modified instance to an instance of our problem. Let n be the number of clauses. We create consumers corresponding to every clause, and the literals in each clause correspond to the products of interest to the consumers (i.e. S_j). For a product, setting the corresponding variable to be *true* (respectively, *false*) corresponds to setting the price to a (respectively, b). For the positive clauses, there is one consumer with $B_j = a = n$, while for the negative clauses, we have n identical consumers, each with budget $B_j = b = 1$. A solution for the pricing instance corresponds naturally to a solution for the MAX-3SAT instance. The only difference in the objective function values arises from the fact that even if the consumers with budget a do not pay a, they pay at least b. Let n_a be the number of consumers with budget a. A price setting where n_{ab} of the n_a consumers with budget a spend $b = 1$ has a contribution of n_{ab} from these consumers to the total payoff. Since setting the price of all products to $a = n$ leads to a payoff of $n_a n$, and $n_{ab} \leq n_a$, in the optimal solution, the

fraction of the total payoff contributed by these n_{ab} consumers is at most $1/n$, which is negligible. Thus any $16/15 - \epsilon$ approximate solution for this multi-product pricing instance leads to a $16/15 - \epsilon$ approximate solution for the MAX-SAT instance. Thus, the Uniform-two-budget case of the Max-Buying with No-Price-Ladder problem cannot be approximated better than 16/15 unless $P = NP$. □

We now give a hardness result for the Min-Buying with No-Price-Ladder problem.

3.3 Min-Buying with No-Price-Ladder

We consider the Uniform-two-budget case of the problem, where each consumer j specifies a set S_j of products she is interested in, and her budget $b_{ij} = B_j$ for $i \in S_j$, and $b_{ij} = 0$ otherwise. Also, $B_j \in \{a, b\}$, for all j.

Theorem 5. *The Uniform-two-budget case of the Min-Buying with No-Price-ladder problem is NP-hard to approximate within* $1 + \epsilon$, *for some constant* $\epsilon > 0$.

Proof. We do a reduction from the following MAX-CSP: Consider the MAX-SAT problem consisting of the following two types of clauses, conjunction of only positive literals and disjunction of only negative literals.

We first show that this version of MAX-CSP is MAX SNP-hard. We achieve this goal by first showing that it is NP-hard. Then, the results from Khanna, Sudan, and Williamson [8] will imply that it is MAX SNP-hard (given that it is NP-hard). Specifically, we show that the following version of MAX-CSP is NP-hard: The conjunctions contain only singleton x_i's and the disjunctions are of the form $\overline{x_i} \vee \overline{x_j}$ with two literals. We first note that there exists an optimal solution to this MAX-CSP which maximizes the number of x_i set to 1 while satisfying all $\overline{x_i} \vee \overline{x_j}$ clauses. If a solution does not satisfy all disjunctive clauses, we can convert it into a equally good or better solution as follows: Satisfy all the disjunctive clauses by negating either of the literals for any unsatisfied disjunctive clause. Now, if we have a vertex i for each x_i and view the disjunctions $\overline{x_i} \vee \overline{x_j}$ as edges (i, j), then this MAX-CSP is equivalent to the the maximum independent set problem, which is NP-hard to solve.

Given an instance of this MAX-CSP, we reduce it to an instance of the Min-Buying with No-Price-Ladder problem as follows. There are two distinct budget values $a > b$ with $a = 2b$. Corresponding to each variable x_i, we have a product i. For a product, setting the price to a (respectively, b) corresponds to setting the corresponding variable to be *true* (respectively, *false*). Corresponding to a conjunctive clause, we have a consumer with budget a interested in the products appearing in the clause. Similarly, corresponding to a disjunctive clause, we have a consumer with budget b interested in the products appearing in the clause. Since we are in the Min-Buying setting, a consumer with budget a will always pay b, and will pay the an additional b if the conjunction of the x_i for the products it is interested in holds. A consumer with budget b will pay b if the disjunction of the $\overline{x_i}$ for the products it is interested in holds. If the maximum number of satisfiable clauses is at least a constant fraction of the total number of clauses (which can be ensured by adding some dummy clauses to the instance), then the MAX-SNP-hardness of the MAX-CSP problem implies MAX-SNP-hardness of this case of the pricing problem.

□

4 General Algorithms for All Models

In this section, we present an algorithm and hardness result applicable to all six models – {Rank-Buying, Max-Buying, Min-Buying} with {Price-Ladder, No-Price-Ladder}. Recall that the number of products is n while the number of consumers is m.

Theorem 6. *Consider all six models: Rank/Max/Min-Buying with Price-Ladder/No-Price-Ladder. An algorithm which is allowed to assign only k distinct prices to the products cannot have an approximation ratio better than $\frac{H_m}{k}$ with respect to an unrestricted optimal solution.*

Proof. To show this lower bound, we create an instance with $n = m$. Consider a situation where a consumer j is interested only in product j and has budget m/j for it and 0 for all other products. In the optimal solution, product i is priced m/i. Thus, in the optimal solution, consumer j spends his budget m/j, and the total payoff is mH_m. Now consider a solution which assigns only k distinct prices to the products. Products priced at any single price m/j can be afforded by at most the first j consumers, thus giving a payoff of $\leq m$ for that price. Thus the total payoff with k distinct prices is at most km. This gives the H_m/k lower bound. □

When $k = 1$, the above bound is tight. Let $B_j^{max} = \max_i B_{ij}$. Assume that the consumers are ordered by their maximum budgets, B_j^{max}, in decreasing order. If we set the prices of all products to B_j^{max}, then the first j consumers will be able to afford some product they are interested in, and pay price B_j^{max} for it (irrespective of the policy governing consumer behavior in case of ties). Thus, the payoff generated will be jB_j^{max}. If $j' = \arg\max_j jB_j^{max}$, then we set the price of all products to be $B_{j'}^{max}$. For this single-price algorithm, we get the following theorem.

Theorem 7. *The single price algorithm provides an H_m-approximation for revenue maximization.*

Proof. We note that the revenue generated by an optimal solution $OPT \leq \sum_j B_j^{max}$, since each consumer j spends at most B_j^{max}, her maximum budget over all products. Let $\mathcal{R}$ be the revenue generated by the single price algorithm. Then, $\mathcal{R} = j' B_{j'}^{max} \geq jB_j^{max}$ for all j. Thus, $OPT \leq \sum_j B_j^{max} \leq \sum_j \mathcal{R}/j = \mathcal{R}H_m$. □

5 Conclusion and Open Problems

In this paper, we studied the non-parametric multi-product pricing problem. We presented a PTAS for the realistic Rank-Buying with Price-Ladder model, thus providing a useful pricing tool to companies with access to customer profiles. We also present approximation algorithms and complexity analysis for various extensions and variants of the model.

Many problems still remain open. The complexity of the Rank-Buying with Price-Ladder problem is unresolved. It will be interesting to extend the results for profit maximization to the Rank-Buying with Price-Ladder problem. One can also study other

extensions of the multi-product pricing problem. A possible extension might be to consider a *budget range* for each consumer – each consumer has a lower as well as upper bound on the amount they are willing to spend to buy a product. Another model of (especially) game theoretic interest is Max-Gain-Buying, where each consumer buys the product that gives it the maximum *gain,* which is defined to be the difference between the utility (budget) and price of a product.

References

1. T. Asano, and D. Williamson. "Improved approximation algorithms for MAX SAT." In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms,* pp. 96–115, 2000.
2. P. Coy. "The Power of Smart Pricing." *Business Week,* April 10, 2000
3. G. Dobson and S. Kalish. "Heuristics for Pricing and Positioning a Product-line Using Conjoint and Cost Data." *Management Science,* vol. 39, no. 2, pp. 160–175,1993.
4. General Motors. *The Auto Choice Advisor web site.* `http://www.autochoiceadvisor.com.`
5. J. Håastad. "Some Optimal Inapproximability results." In *Proceedings of the 28th ACM Symposium on the Theory of Computing,* pp. 1–10,1997.
6. W. Hansom and K. Martin. "Optimal Bundle Pricing." *Management Science,* vol. 36, pp. 155–174,1990.
7. J.R. Hauser and P. Simmie. "Profit Maximizing Perceptual Positions: An Integrated Theory for the Selection of Product Features and Price." *Management Science,* vol. 27, no. 1, pp. 33–56, 1981.
8. S. Khanna, M. Sudan, and D. Williamson. "A Complete Classification of the Approximability of Maximization Problems Derived from Boolean Constraint Satisfaction." In *Proceedings of the 29th ACM Symposium on the Theory of Computing,* pp. 11–20,1997.
9. J. I. McGill, and G. Van Ryzin. "Revenue Management: Research Overview and Prospects." *Transportation Science,* vol. 33, no. 2, pp. 233–256,1999.
10. G. McWilliams. "Lean Machine: How Dell Fine Tunes Its PC Pricing to Gain Edge in a Slow Market." *Wall Street Journal,* June 8, 2001.
11. A. Merrick. "Priced to Move: Retailers Try to Get Leg Up on Markdowns With New Software." *Wall Street Journal,* August 7, 2001
12. P. Rusmevichientong. "A Non-Parametric Approach to Multi-Product Pricing: Theory and Application." PhD Thesis, 2003.
13. P. Rusmevichientong, B. Van Roy, and P. Glynn. "A Non-Parametric Approach to Multi-Product Pricing." Submitted to *Operations Research,* 2002.
14. L. R. Weatherford, and S. E. Bodily. "A Taxonomy and Research Overview of Perishable-Asset Revenue Management: Yield Management, Overbooking, and Pricing." *Management Science,* vol. 40, no. 5, pp. 831-844, 1992.

A Alternate Algorithm for Price-Ladder with Rank/Max-Pricing

Theorem 8. *For the Max-Buying with Price-Ladder model, an approximation ratio of* $1+\epsilon(1+o(1))$ *can be achieved in time* $O(n^{\frac{1}{\epsilon}\log\log\log n})$ *and space* $O(n)$ *for any* $\epsilon > 0$ *by choosing* $\frac{\log n}{\epsilon}(1+o(1))$ *distinct prices.*

Proof. For this, we guess the number of consumers buying at a certain price, instead of guessing the products priced at a certain price as in Theorem 1.

We first modify the instance so that every product is of interest to at most one consumer. In the original instance, if m_i consumers are interested in the some product i, we create m_i new products with each of the m_i consumers interested in one of these products in the new instance. We specify the price order of these new products as follows: the m_i products are constrained to have prices in an order that is the reverse of the budget order of the corresponding consumers for product i. Thus, an optimal solution to the new instance will assign the same price to all these m_i products, and hence, an optimal solution to the new instance can easily be converted to a solution for the original one.

We may assume that the prices used in the solution are of the form $\mathcal{B}/(1+\mu)^i$, where $\mathcal{B}$ is the highest budget, and no smaller than $\delta\mathcal{B}/n$, by incurring a loss of $((1+\mu)(1+\delta))$ in the approximation factor. This restriction results in at most $k = \frac{\log n}{\mu}(1+o(1))$ possible prices. Let $\mathcal{S}$ be the maximum revenue generated using only a single price as in Theorem 7. We further restrict the solution space such that every chosen price gives a revenue of at least $\delta\mathcal{S}/k$, by losing an approximation factor of $1+\delta$. In addition, we only consider solutions where the number of consumers that buy at a price p to be of the form $(1+\delta)^i$, with a further loss of $1+\delta$ in the approximation factor. This leaves at most $\frac{\log k}{\delta}(1+o(1))$ possible choices for the number of consumers that buy at a given price. Overall, the approximation factor is $(1+\mu)(1+\delta)^3$.

The total number of choices of how many consumers will pay which price is thus $L = (\frac{\log k}{\delta})^k = (\frac{\log\log n - \log\mu}{\delta})^{\frac{\log n}{\mu}} = n^{\frac{1}{\mu}(\log(\log\log n - \log\mu) - \log\delta)}$. If we choose $\mu + 3\delta = \mu(1+o(1))$ and $\mu = \epsilon(1+o(1))$, then the approximation factor is $1+\epsilon(1+o(1))$ and $L = n^{\frac{1}{\epsilon}\log\log\log n}$.

Consider any of these L choices, which specifies the number of consumers N_p that buy at any (rounded) price p. This gives us a projected revenue of $\mathcal{R} = \sum_p N_p p$. We next try to find a solution that generates a revenue of at least $\mathcal{R}$. We use a greedy strategy: consider the products one by one in decreasing order of price. Start with the highest price p, set the price of the next product to p, until there are N_p consumers buying at price p. We repeat for the next lower (rounded) price and so on. If there exists an unrounded solution with at least N_p consumers buying at (rounded) price p, then this procedure will always be able to find N_p consumers buying at price p. We can see this as follows: the unrounded solution has more than N_p consumers buying at price p. Since our greedy solution has to pick fewer consumers at each price, the set of products available to be priced at p is always a superset of the set of products priced p in the unrounded solution. Thus, the greedy algorithm would never run out of products or consumers. As argued earlier, we can easily modify this solution for the modified instance, where each product is of interest to only one consumer, to get a solution for the original instance without any loss in revenue. □

Exponential Lower Bounds for the Running Time of DPLL Algorithms on Satisfiable Formulas

Michael Alekhnovich[1]*, Edward A. Hirsch[2]**, and Dmitry Itsykson[3]

[1] Institute for Advanced Study, Princeton, USA,
misha@ias.edu
[2] Steklov Institute of Mathematics, St. Petersburg, Russia,
hirsch@pdmi.ras.ru
[3] St.Petersburg State University, St.Petersburg, Russia,
dmitrits@mail.ru

Abstract. DPLL algorithms form the largest family of contemporary algorithms for SAT (the propositional satisfiability problem) and are widely used in applications. The recursion trees of DPLL algorithm executions on unsatisfiable formulas are equivalent to tree-like resolution proofs. Therefore, lower bounds for tree-like resolution (known since 1960s) apply to them. However, these lower bounds say nothing about their behavior on *satisfiable* formulas. Proving exponential lower bounds for them in the most general setting is impossible without proving $\mathbf{P} \neq \mathbf{NP}$; thus, in order to prove lower bounds one has to restrict the power of branching heuristics. In this paper, we give exponential lower bounds for two families of DPLL algorithms: *generalized myopic* algorithms (that read up to $n^{1-\epsilon}$ of clauses at each step and see the remaining part of the formula without negations) and *drunk* algorithms (that choose a variable using any complicated rule and then pick its value at random).

1 Introduction

SAT solving heuristics. The propositional satisfiability problem *(SAT)* is one of the most well-studied **NP**-complete problems. In this problem, one is asked whether a Boolean formula in conjunctive normal form (a conjunction of *clauses,* which are disjunctions of *literals,* which are variables or their negations) has an assignment that satisfies all its clauses. Despite the $\mathbf{P} \neq \mathbf{NP}$ conjecture, there is a lot of algorithms for SAT (motivated, in particular, by its importance for applications). DPLL algorithms (defined below) are based on the most popular

* Supported by CCR grant NCCR-0324906.
** Supported in part by RAS program of fundamental research "Research in principal areas of contemporary mathematics", RFBR grant #02-01-00089, and by Award No. RM1-2409-ST-02 of the U.S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 84–96, 2004.

approach that originates in the papers by Davis, Putnam, Logemann and Loveland [9,8]. Very informally, these algorithms use a "divide-and-conquer" strategy: they split a formula into two subproblems by fixing a value of some literal, then they recursively process the arising formulas. These algorithms received much attention of researchers both from theory and practice and are heavily used in the applications.

Lower bounds for Resolution and the running time of DPLL algorithms. Propositional proof systems form one of the simplest and the most studied model in propositional calculus. Given a formula F, a propositional proof system allows to show that F is unsatisfiable. For example, using the well-known *resolution rule* $\frac{x \vee C_1;\ \neg x \vee C_2}{C_1 \vee C_2}$ one can non-deterministically build a *resolution refutation of* F, which may be used as a certificate of unsatisfiability for the formula F. It is well-known that the size of the minimum tree-like resolution refutation and the running time of DPLL algorithms are polynomially related. Therefore, (sub)exponential lower bounds for tree-like resolution (starting with Tseitin's bounds [14] and finishing with quite strong bounds of [12]) imply that any DPLL algorithm should take exponentially long to prove that the corresponding formulas are unsatisfiable. However, these results say nothing in the case of *satisfiable* formulas. There are several reasons why the performance may differ on satisfiable and unsatisfiable instances:

- Experiments show that contemporary SAT solvers are able to solve much larger satisfiable formulas than unsatisfiable ones [13].
- Randomized one-sided error algorithms fall out of scope, since they do not yield proofs of unsatisfiability.
- If a DPLL algorithm is provably efficient (i.e. takes polynomial time) on some class of formulas, then one can interrupt the algorithm running on a formula from this class after sufficiently large number of steps if it has not found a satisfying assignment. This will give a certificate of unsatisfiability that can be much smaller than the minimum tree-like resolution refutation.

Previously known lower bounds for satisfiable formulas. Despite the importance of the problem, only few works have addressed the question of the worst-case running time of SAT algorithms on satisfiable formulas. There has been two papers [10,4] on (specific) local search heuristics; as to DPLL algorithms all we know are the bounds of [11,1,2].

In the work of Nikolenko [11] exponential lower bounds are proved for two specific DPLL algorithms (called `GUC` and `Randomized GUC`) on specially tailored satisfiable formulas.

Achlioptas, Beame, and Molloy [1] prove the hardness of random formulas in 3-CNF with n variables and cn $(c < 4)$ clauses for three specific DPLL algorithms (called `GUC`, `UC`, and `ORDERED-DLL`). It is an open problem to prove that these formulas are satisfiable (though it is widely believed they are). Recently, the same authors [2] have proved an *unconditional* lower bound on satisfiable random formulas in 4-CNF for `ORDERED-DLL`. The latter result states that

ORDERED-DLL takes exponential time with *constant* (rather than exponentially close to 1) probability.

Our contribution. Proving such bounds for DPLL algorithms in a greater generality is the ultimate goal of the present paper. We design two families of satisfiable formulas and show lower bounds for two general classes of algorithms which are much less restricted than those studied before.

The first class of formulas simply encodes a linear system $Ax = b$ that has a unique solution over $\mathbb{GF}_2$, where A is a "good" expander. We prove that any *generalized myopic* DPLL algorithm that has a local access to the formula (i.e., can read up to $n^{1-\epsilon}$ clauses at every step) with high probability has to make an exponential number of steps before it finds a satisfying assignment.

In our second result we describe a general way to cook a satisfiable formula out of any unsatisfiable formula hard for tree-like resolution so that the resulting formula is hard for any *drunk* DPLL algorithm that chooses a variable in an arbitrarily complicated way and then tries both its values in a random order.

The proofs of auxiliary statements are omitted due to the space restrictions; please refer to the full version of the paper for the details.

2 Preliminaries

Let x be a Boolean variable ranging over {0,1}. A *literal* of x is either x or $\neg x$. A *clause* is a disjunction of literals (considered as a set). A *formula* in this paper refers to a Boolean formula in conjunctive normal form, i.e., a conjunction of clauses (a formula is considered as a multiset). A formula in k-*CNF* contains clauses of size at most k. We will use the notation $\mathbf{Vars}(\mathcal{O})$ to denote the set of variables occurring in any object $\mathcal{O}$ (a clause, a formula, etc.).

An *elementary substitution* $v := \varepsilon$ just chooses a Boolean value $\varepsilon \in \{0, 1\}$ for a variable v. A *substitution* (also called a *partial assignment*) is a set of elementary substitutions for different variables. The result of applying a substitution ρ to a formula F (denoted by $F[\rho]$) is a new formula obtained from F by removing the clauses containing literals satisfied by ρ and removing the opposite literals from other clauses.

For a non-negative integer n, let $[n] = \{1, 2, \ldots, n\}$. For a vector $v = (v_1, ..., v_m)$ and index set $I \subseteq [m]$ we denote by v_I the subvector with coordinates chosen according to I. For a matrix A and a set of rows $I \subseteq [m]$ we use the notation A_I for the submatrix of A corresponding to these rows. In particular, we denote the ith row of A by A_i and identify it with the set $\{j \mid A_{ij} = 1\}$. The cardinality of this set is denoted by $|A_i|$.

DPLL algorithms: general setting. A DPLL algorithm is a recursive algorithm. At each step, it simplifies the input formula F (without affecting its satisfiability), chooses a variable v in it and makes two recursive calls for the formulas $F[v := 1]$ and $F[v := 0]$ in some order; it outputs "Satisfiable" iff at least one of the recursive calls says so (note that there is no reason to make the second call if

Algorithm $\mathcal{A}$.
Input: formula F in CNF.
Output: "Satisfiable" or "Unsatisfiable".

1. Simplify F using *simplification rules.*
2. If F is empty, return "Satisfiable".
3. If F contains the empty clause, return "Unsatisfiable".
4. Choose a variable v using *Heuristic A.*
5. Choose a Boolean value ε using *Heuristic B.*
6. If $\mathcal{A}(F[v := \varepsilon])$ returns "Satisfiable", return "Satisfiable".
7. If $\mathcal{A}(F[v := \neg\varepsilon])$ returns "Satisfiable", return "Satisfiable".
8. Return "Unsatisfiable".

Fig. 1. A DPLL algorithm.

the first one was successful). The recursion proceeds until the formula trivializes, i.e., it becomes empty (hence, satisfiable) or one of the clauses becomes empty (hence, the formula is unsatisfiable).

A DPLL *recursion tree* is a binary tree (a node may have zero, one, or two children) in which nodes correspond to the intermediate subproblems that arise after the algorithm makes a substitution, edges correspond to the recursive calls on the resulting formulas. The computation of a DPLL algorithm thus can be considered as depth-first traverse of the recursion tree from the left to the right; in particular, the rightmost leaf always corresponds to the satisfying assignment (if any), the overall running time is proportional to the size of the tree. For a node v in the computation tree by ρ_v we denote the partial assignment that was set prior to visiting v, thus the algorithm at v works on the subformula $F[\rho_v]$.

A DPLL algorithm is determined by its simplification rules and two heuristics: Heuristic A that chooses a variable and Heuristic B that chooses its value to be examined first. A formal description is given in Fig. 1. Note that if **P = NP** and Heuristic B is not restricted, it can simply choose the correct values and the algorithm will terminate quickly. Therefore, in order to prove unconditional lower bounds one has to restrict the simplification rules and heuristics and prove the result for the restricted model. In this paper, we consider two models: generalized myopic algorithms and drunk algorithms. Both models extend the original algorithm of [8], which uses the unit clause and pure literal rules and no non-trivial Heuristics A and B.

Drunk algorithms. Heuristic A of a drunk algorithm can be arbitrarily complicated (even non-recursive). This is compensated by the simplicity of Heuristic B: it chooses 0 or 1 at random. The simplification rules are

Unit clause elimination. If F contains a clause that consists of a single variable v (or its negation $\neg v$), replace F by $F[v := 1]$ (resp., $F[v := 0]$).

Pure literal elimination. If a variable v occurs in F only positively (resp., negatively), replace F by $F[v := 1]$ (resp., $F[v := 0]$).

Subsumption. If the formula F contains a clause that contains another clause as a subset, delete the larger clause.

Note that `Randomized GUC` with pure literal elimination considered in [11] is a drunk algorithm (that does not use subsumption).

Generalized myopic algorithms. Both heuristics are restricted w.r.t. the parts of formula that they can read (this can be viewed as accessing the formula via an oracle). Heuristic A can read

- $K(n)$ clauses of the formula (where n is the number of variables in the original input formula and $K(n) = n^{1-\epsilon}$ is a function with $\epsilon > 0$);
- the formula with negation signs removed;
- the number of occurrences of each literal.

Heuristic B may use the information obtained by Heuristic A. The information revealed about the formula can be used in the subsequent recursive calls (but not in other branches of the recursion tree).

The only simplification rule is pure literal elimination. Also the unit clause elimination can be easily implemented by choosing the proper variable and value. In particular, heuristics `ORDERED-DLL`, `GUC` and `UC` considered in [1] yield generalized myopic algorithms. Note that our definition indeed generalizes the notion of myopic algorithms introduced in [3].

Formally, the heuristics are unable to read all clauses containing a variable if this variable is too frequent. However, it is possible to restrict our hard formulas (that we use for proving our exponential lower bound) so that every variable occurs $O(\log n)$ times.

Expanders. An expander is a bounded-degree graph that has many neighbors for every sufficiently small subset of its nodes. Similarly to [5], we use a more general notion of expander as an $m \times n$ matrix. There are two notions of expanders: expanders and boundary expanders. The latter notion is stronger as it requires the existence of unique neighbors. However, every good expander is also a boundary expander.

Definition 1. *For a set of rows $I \subseteq [m]$ of an $m \times n$ matrix A, we define its* boundary $\partial_A I$ *(or just ∂I) as the set of all $j \in [n]$ (called* boundary elements*) such that there exists exactly one row $i \in I$ that contains j. We say that A is an* (r, s, c)-boundary expander *if*

1. $|A_i| \leq s$ *for all* $i \in [m]$, *and*
2. $\forall I \subseteq [m]\ (|I| \leq r \Rightarrow |\partial I| \geq c \cdot |I|)$.

Matrix A is an (r, s, c)-expander *if condition 2 is replaced by*

$2'$. $\forall I \subseteq [m]$ $(|I| \leq r \Rightarrow |\bigcup_{i \in I} A_i| \geq c \cdot |I|)$.

We define the boundary and boundary elements of equation(s) in a linear system $Ax = b$ similarly to those of rows in a matrix A.

While several probabilistic and explicit constructions of expanders are known (see, e.g., [5]), in our construction of hard satisfiable formulas we need expanders with an additional property (refer to the full version of this paper for a formal proof of their existence).

Theorem 1. *For every sufficiently large* n, *there exists an* $n \times n$ *non-degenerate matrix* $A^{(n)}$ *such that* $A^{(n)}$ *is an* $(n/\log^{14} n, 3, 25/13)$-*expander.*

Definition 2 ([6]). *Let* $A \in \{0,1\}^{m \times n}$. *For a set of columns* $J \subseteq [n]$ *define the following inference relation* $\vdash_J$ *on the sets* $[m]$ *of rows of A:*

$$I \vdash_J I_1 \iff |I_1| \leq r/2 \ \wedge \ \partial_A(I_1) \subseteq \left[\bigcup_{i \in I} A_i \cup J\right]. \tag{1}$$

That is, we allow to derive rows of A from already derived rows. We can use these newly derived rows in further derivations. (for example, we can derive new rows from $I \cup I_1$*). Let the* closure $\mathrm{Cl}(J)$ *of J be the set of all rows which can be inferred via* $\vdash_J$ *from the empty set.*

Lemma 1 ([6, Lemma 3.16]). *For any set J with* $|J| \leq (cr/2)$, $|\mathrm{Cl}(J)| \leq r/2$.

We also need another (stronger) closure operation the intuitive sense of which is to extract a good expander out of a given matrix by removing rows and columns.

Definition 3. *For an* $A \in \{0,1\}^{m \times n}$ *and a subset of its columns* $J \subseteq [n]$ *we define an inference relation* $\vdash'_J$ *on subsets of rows of A:*

$$I \vdash'_J I_1 \iff |I_1| \leq r/2 \ \wedge \ \left|\partial_A(I_1) \setminus \left[\bigcup_{i \in I} A_i \cup J\right]\right| < c/2|I_1| \tag{2}$$

Given a set of rows I and a set of columns J consider the following cleaning *step:*

- *If there exists a nonempty subset of rows* I_1 *such that* $I \vdash'_J I_1$, *then*
  - *Add* I_1 *to I.*
  - *Remove all rows corresponding to* I_1 *from A.*

Repeat the cleaning step as long as it is applicable. Fix any particular order on the sets to exclude ambiguity, initialize $I = \emptyset$ *and denote the resulting content of I at the end by* $\mathrm{Cl}^{\mathrm{e}}(J)$.

Lemma 2. *Assume that A is an arbitrary matrix and J is a set of its columns. Let $I' = \mathrm{Cl}^e(J)$, $J' = \bigcup_{i \in \mathrm{Cl}^e(J)} A_i$. Denote by $\hat{A}$ the matrix that results from A by removing the rows corresponding to I' and columns to J'. If $\hat{A}$ is non-empty than it is an $(r/2, 3, c/2)$-boundary expander.*

Lemma 3. *If $|J| < cr/4$, then $|\mathrm{Cl}^e(J)| < 2c^{-1}|J|$.*

3 A Lower Bound for Generalized Myopic Algorithms

In this section, we prove an exponential lower bound on the running time of generalized myopic algorithms on satisfiable formulas. The proof strategy is as follows: we take a full-rank $n \times n$ 0/1-matrix A having certain expansion properties and construct a uniquely satisfiable Boolean formula Φ expressing the statement $Ax = b$ (modulo 2) for some vector b. Then we prove that if one obtains an unsatisfiable formula from Φ using a reasonable substitution, the resulting formula is hard for tree-like resolution (the proof is similar to that of [7]). Finally, we show that changing several bits in the vector b, while changes the satisfying assignment, does not affect the behavior of a generalized myopic algorithm that did not reveal these bits, which implies it encounters a hard unsatisfiable formula on its way to the satisfying assignment.

Theorem 1 defines an $n \times n$ matrix A and parameters $r = n/\log^{14} n$, $c' = 25/13$. Denote $c = 2c' - 3$ (thus A is an $(r, 3, c)$-boundary expander). We fix A, r, c, c' until the end of this section.

Definition 4. *Let b be a vector from $\{0,1\}^n$. Then $\Phi(b)$ is the formula expressing the equality $Ax = b$ (modulo 2), namely, every equation $a_{ij_1}x_{j_1} + a_{ij_2}x_{j_2} + a_{ij_3}x_{j_3} = b_i$ is transformed into the 4 clauses on $x_{j_1}, x_{j_2}, x_{j_3}$ satisfying all its solutions. Sometimes we identify an equation with the corresponding clauses. We also identify $j \in J$ (where J is a set of columns of A) with the variable x_j.*

Remark 1. The formula $\Phi(b)$ has several nice properties that we use in our proofs. First, it has exactly one satisfying assignment (since $\mathrm{rk}\, A = n$). It is also clear that a myopic DPLL algorithm has no chance to apply pure literal elimination to it, because for any substitution ρ, the formula $\Phi(b)[\rho]$ never contains a pure literal unless this pure literal is contained in a unit clause. Moreover, the number of occurrences of a literal in $\Phi(b)[\rho]$ always equals the number of occurrences of the opposite literal (recall that a formula is a *multiset* of clauses); again the only exception is literals occurring in unit clauses.

Definition 5. *A substitution ρ is said to be* locally consistent *w.r.t. the linear system $Ax = b$ if and only if ρ can be extended to an assignment on X which satisfies the equations corresponding to $\mathrm{Cl}(\rho)$, namely, $A_{\mathrm{Cl}(\rho)}x = b_{\mathrm{Cl}(\rho)}$.*

Lemma 4. *Let A be an $(r,3,c)$-boundary expander, $b \in \{0,1\}^n$, and ρ be a locally consistent partial assignment. Then for any set of rows $I \subset [n]$ with $|I| \leq r/2$, ρ can be extended to an assignment x which satisfies the subsystem $A_I x = b_I$.*

Proof. Assume for the contradiction that there exists set I for which ρ cannot be extended to satisfy $A_I x = b_I$; choose the minimal such I. Then $\partial_A(I) \subseteq \mathrm{Vars}(\rho)$, otherwise one could remove an equation with boundary variable in $\partial_A(I) \setminus \mathrm{Vars}(\rho)$ from I. Thus, $\mathrm{Cl}(\rho) \supseteq I$, which contradicts Definition 5. □

We need the following lemma which is a straightforward generalization of [7].

Lemma 5. *For any matrix A which is an $(r,3,c)$-boundary expander and any vector $b \notin \mathrm{Im}(A)$ any resolution proof of the system $Ax = b$ must have size $2^{\Omega(cr)}$.*

Recall that the hard formula $\Phi(b)[\rho]$ in Definition 4 is constructed using an $(r,3,c)$-boundary expander A.

Lemma 6. *If a locally consistent substitution ρ s.t. $|\mathrm{Vars}(\rho)| \leq cr/4$ results in an unsatisfiable formula $\Phi(b)[\rho]$ then every generalized myopic DPLL algorithm would take $2^{\Omega(r)}$ time on $\Phi(b)[\rho]$.*

Proof. The work of any DPLL algorithm on an unsatisfiable formula can be translated to tree-like resolution refutation so that the size of the refutation is the working time of the algorithm. Thus, it is sufficient to show that the minimal tree-like resolution refutation size of $\Phi(b)[\rho]$ is large. Denote by $I = \mathrm{Cl}^e(\rho)$, $J = \bigcup_{i \in I} A_i$. By Lemma 3 $|I| \leq r/2$. By Lemma 4 ρ can be extended to another partial assignment ρ' on variables x_J, s.t. ρ' satisfies every linear equation in $A_I x = b_I$. The restricted formula $(Ax = b)|_{\rho'}$ still encodes an unsatisfiable linear system, $A'x = b'$, where matrix A' results from A by removing rows corresponding to I and variables corresponding to J. By Lemma 2, A' is an $(r/2,3,c/2)$-boundary expander and Lemma 5 now implies that the minimal tree-like resolution refutation of the Boolean formula corresponding to the system $A'x = b'$ has size $2^{\Omega(r)}$. □

Theorem 2. *For every deterministic generalized myopic DPLL algorithm $\mathcal{A}$ that reads at most $K = K(n)$ clauses per step, $\mathcal{A}$ stops on $\Phi(b)$ in $2^{o(r)}$ steps with probability $2^{-\Omega(r/K)}$ (taken over b uniformly distributed on $\{0,1\}^n$).*

Corollary 1. *Let $\mathcal{A}$ be any (randomized) generalized myopic DPLL algorithm that reads at most $K = K(n)$ clauses per step. $\mathcal{A}$ stops on $\Phi(b)$ in $2^{o(n \log^{-14} n)}$ steps with probability $2^{-\Omega(K^{-1} n \log^{-14} n)}$ (taken over random bits used by the algorithm and over b uniformly distributed on $\{0,1\}^n$).*

Proof (of Theorem 2). The proof strategy is to show that during its very first steps the algorithm does not get enough information to guess a correct substitution with non-negligible probability. Therefore, the algorithm chooses an incorrect substitution and has to examine an exponential-size subtree by Lemma 6.

Without loss of generality, we assume that our algorithm is a *clever myopic* algorithm. We define a clever myopic algorithm w.r.t. matrix A as a generalized myopic algorithm (defined as in Section 2) that

- has the following ability: whenever it reveals occurrences of the variables x_J (at least one entry of each) it can also read all clauses in $\mathrm{Cl}(J)$ for free and reveal the corresponding occurrences;
- never asks for a number of occurrences of a literal (syntactical properties of our formula imply that $\mathcal{A}$ can compute this number itself: the number of occurrences outside unit clauses does not depend on the substitutions that $\mathcal{A}$ has made; all unit clauses belong to $\mathrm{Cl}(J)$);
- always selects one of the revealed variables;
- never makes stupid moves: whenever it reveals the clauses C and chooses the variable x_j for branching it makes the right assignment $x_j = \epsilon$ in the case when C semantically imply $x_j = \epsilon$ (this assumption can only save the running time).

Proposition 1. *After the first* $\lfloor \frac{cr}{6K} \rfloor$ *steps a clever myopic algorithm reads at most* $r/2$ *bits of* b.

Proof. At each step the algorithm makes K clause queries, asking for $3K$ variable entries. This will sum up to $3K(cr/(6K))$ variables which will result by Lemma 1 in at most $r/2$ revealed bits of b. □

Proposition 2. *During the first* $\lfloor \frac{cr}{6K} \rfloor$ *steps the current partial assignment made by a clever myopic algorithm is locally consistent (in particular, the algorithms does not backtrack).*

Proof. Follows by repeated application of Lemma 4. □

Assume now that b chosen at random is hidden from $\mathcal{A}$. Whenever an algorithm reads the information about a clause corresponding to the linear equation $A_i x = b_i$ it reveals the ith bit of b. Let us observe the situtation after the first $\lfloor \frac{cr}{6K} \rfloor$ steps of $\mathcal{A}$, i.e., the $\lfloor \frac{cr}{6K} \rfloor$-th vertex v in the leftmost branch in the DPLL tree of the execution of $\mathcal{A}$. By Proposition 1 the algorithm reads at most $r/2$ bits of b. Denote by $I_v \subset [m]$ the set of the revealed bits, and by R_v the set of the assigned variables, $|R_v| = \lfloor \frac{cr}{6K} \rfloor$. The idea of the proof is that $\mathcal{A}$ cannot guess the true values of x_{R_v} by observing only r bits of b. Denote by ρ_v the partial assignment to the variables in R_v made by $\mathcal{A}$. Consider the event $E = \{(A^{-1}b)_{R_v} = \rho_v\}$ (recall that our probability space is defined by the 2^n possible values of b). This event holds if and only if the formula $\Phi(b)|_{\rho_v}$ is satisfiable. For $I \subset [n], R \subset [n], \epsilon \in \{0,1\}^I, \rho \in \{0,1\}^R$ we want to estimate the

conditional probability $\Pr[E \mid I_v = I,\ R_v = R,\ b_{I_v} = \epsilon,\ \rho_v = \rho\,]$. If we show that this conditional probability is small (irrespectively of the choice of I, R, ϵ, and ρ), it will follow that the probability of E is small.

We use the following projection lemma that is proved in the full version of the paper. Intuitively it says that if A is a good expander and $\mathcal{L}$ is a small subsystem of $Ax = b$ then the set of solutions of $\mathcal{L}$ being projected on any set of variables $\hat{X} \subseteq X$ is either empty or sufficiently large.

Lemma 7. *Assume that an $n \times n$ matrix A is an $(r, 3, c')$-expander, $X = \{x_1, \ldots, x_n\}$ is a set of variables, $\hat{X} \subseteq X$, $|\hat{X}| < r$, $b \in \{0,1\}^m$, and $\mathcal{L} = \{\ell_1, \ldots, \ell_k\}$ (where $k < r$) is a tuple of linear equations from the system $Ax = b$. Denote by L the set of assignments to the variables in $\hat{X}$ that can be extended to X to satisfy $\mathcal{L}$. If L is not empty then it is an affine subspace of $\{0,1\}^{\hat{X}}$ of dimension greater than $|\hat{X}|\left(\frac{1}{2} - \frac{14-7c'}{2(2c'-3)}\right)$.*

Choose $\mathcal{L} = \{A_i x = \epsilon_i\}_{i \in I}$, $X = \mathrm{Vars}(\mathcal{L})$, $\hat{X} = R$, $|\hat{X}| = \lfloor cr/(6K) \rfloor$, recall that $c' = 25/13$. Denote by L the set of locally consistent assignments to the variables in R, i.e., L is the projection of all assignments satisfying $A_I x = \epsilon_I$ on R. Then Lemma 7 says that $\dim L > \frac{2}{11}|R|$. Define $(\hat{b})_i = \epsilon_i$ for $i \in I$ and $(\hat{b})_i = b_i$ otherwise. Note that $\hat{b}$ has the distribution of b when we fix $I_v = I$ and $b_I = \epsilon$. The vector $\hat{b}$ is independent from the event

$$E_1 = [I_v = I \wedge R_v = R \wedge b_{I_v} = \epsilon \wedge \rho_v = \rho].$$

This is because in order to determine whether E_1 holds it is sufficient to observe the bits b_I only. Clearly, $(A^{-1}\hat{b})_R$ is distributed uniformly on L (note that A is a bijection), thus

$$\begin{aligned}
&\Pr[E \mid I_v = I,\ R_v = R,\ b_{I_v} = \epsilon,\ \rho_v = \rho\,] \\
&= \Pr[(A^{-1}\hat{b})_R = \rho \mid I_v = I,\ R_v = R,\ b_{I_v} = \epsilon,\ \rho_v = \rho\,] \\
&= \Pr[(A^{-1}\hat{b})_R = \rho\,] \le 2^{-\dim L} < 2^{-\frac{2}{11}|R|} \le 2^{-\frac{cr}{1000K}}.
\end{aligned}$$

However, if E does not happen then by Lemma 6 it takes time $2^{\Omega(r)}$ for $\mathcal{A}$ to refute the resulting unsatisfiable system (note that by Proposition 2 the assignment ρ_v is locally consistent). □

4 A Lower Bound for Drunk Algorithms

In this section, we prove an exponential lower bound on the running time of drunk algorithms (described in Sect. 2) on satisfiable formulas. The proof strategy is as follows: we take a known hard unsatisfiable formula G and construct a new satisfiable formula that turns into G if the algorithm chooses a wrong value for some variable. Since for several tries the algorithms errs at least once with high probability, it follows that the recursive procedure is likely to be called on G and hence will take an exponential time.

In what follows, we give the construction of our hard satisfiable formulas (citing the construction of hard unsatisfiable formulas), then give two (almost trivial) formal statements for the behavior of DPLL algorithms on hard unsatisfiable formulas, and, finally, prove the main result of this section.

Since the size of recursion tree for an unsatisfiable formula does not depend on the random choices of a drunk algorithm, we can assume that our algorithm has the smallest possible recursion tree for every unsatisfiable formula. We call such an algorithm an *"optimal"* drunk algorithm.

Our formulas are constructed from known hard unsatisfiable formulas. For example, we can take hard unsatisfiable formulas from [12].

Theorem 3 ([12], Theorem 1). *For each $k \geq 3$ there exist a positive constant $c_k = O(k^{-1/8})$, a function $f(x) = \Omega(2^{x(1-c_k)})$ and a sequence of unsatisfiable formulas G_n in k-CNF (for each l, G_l uses exactly l variables) such that all tree-like resolution proofs of G_n have size at least $f(n)$.*

Corollary 2. *The recursion tree of the execution of a drunk DPLL algorithm on the formula G_n from Theorem 3 (irrespectively of the random choices made by the algorithm) has at least $f(n)$ nodes.*

Definition 6. *Let us fix n. We call an unsatisfiable formula F (we do* not assume *that F contains n variables)* hard *if the recursion tree of the execution of (every) "optimal" drunk algorithm on F has at least $f'(n) = (f(n)-1)/2$ nodes, where f is the function appearing in Theorem 3.*

Definition 7. *We consider formulas of the form $H_n(x_1^{(1)}, \ldots, x_n^{(n)}) = G^{(1)} \wedge G^{(2)} \wedge \cdots \wedge G^{(n)}$, where $G^{(i)}$ is the formula in CNF of n variables.*[1] *$x_1^{(i)}, \ldots, x_n^{(i)}$ (for all $i \neq j$, the sets of variables of the formulas $G^{(i)}$ and $G^{(j)}$ are disjoint) defined as follows. Take a copy of the hard formula from Theorem 3; call its variables $x_j^{(i)}$ and the formula $\widetilde{G}^{(i)}$. Then change the signs*[2] *of some literals in $\widetilde{G}^{(i)}$ (this is done by replacing all occurrences of a positive literal l with $\neg l$ and, simultaneously, of the negative literal $\neg l$ with l) so that the recursion tree of the execution of (every) "optimal" drunk algorithm on $\widetilde{G}^{(i)}[\neg x_j^{(i)}]$ is not smaller than that on $\widetilde{G}^{(i)}[x_j^{(i)}]$ (hence, $\widetilde{G}^{(i)}[\neg x_j^{(i)}]$ is hard). Use the (modified) formula $\widetilde{G}^{(i)}$ to construct the formula*[3] *$(\widetilde{G}^{(i)} \vee x_1^{(i)}) \wedge (\widetilde{G}^{(i)} \vee x_2^{(i)}) \wedge \cdots \wedge (\widetilde{G}^{(i)} \vee x_n^{(i)})$ and simplify it using the simplification rules; the obtained formula is $G^{(i)}$.*

[1] It is possible that some of these variables do not appear in the formula; therefore, formally, a formula is a pair: a formula and the number of its variables.

[2] We change signs of literals only to simplify the proof of our result; one can think that the algorithm is actually given the input formula without the change.

[3] We use $G \vee x$ to denote a formula in CNF: x is added to each clause of G, and the clauses containing $\neg x$ are deleted.

Lemma 8. *Assume that G is a hard formula, and F has exactly one satisfying assigment. Let the sets of variables of F and G be disjoint. Then $F \wedge G$ is hard.*

Lemma 9. *The formula $G^{(i)}[\neg x_j^{(i)}]$ is hard.*

Theorem 4. *The size of the recursion tree of the execution of a drunk DPLL algorithm on input H_n is less than $f'(n)$ with probability at most 2^{-n}.*

Proof. The unique satisfying assignment to H_n is $x_j^{(i)} = 1$. Note that $H_n[\neg x_j^{(i)}]$ contains an unsatisfiable subformula $G^{(i)}[\neg x_j^{(i)}]$.

Consider the splitting tree of our algorithm on input H_n. It has exactly one leaf corresponding to the satisfying assignment. We call node w on the path corresponding to the satisfying assignment *critical,* if Heuristic A chooses a variable $x_m^{(i)}$ for this node and this is the first time a variable from $G^{(i)}$ is chosen along this path. A *critical subtree* is the subtree corresponding to the unsatisfiable formula resulting from substituting a "wrong" value in a critical node.

By Lemmas 8 and 9 the size of a critical subtree is at least $f'(n)$ (note that the definition of a critical node implies that the corresponding subformula $G^{(i)}$ is untouched in it and hence its child contains a hard subformula $G^{(i)}[\neg x_j^{(i)}]$; it is clear that the simplification rules could not touch $G^{(i)}$ before the first assignment to its variables).

The probability of choosing the value $x_j^{(i)} = 0$ equals $\frac{1}{2}$. There are n critical nodes on the path leading to the satisfying assignment; therefore the probability that the algorithm does not go into any critical subtree equals 2^{-n}. Note that if it ever goes into a critical subtree, it has to examine all its nodes, and there are at least $f'(n)$ of them. □

5 Discussion

Various generalizations of the notions of myopic and drunk algorithms would guide to natural extensions of our results. However, note that merging the notions into one is not easy: if Heuristic A is not restricted, it can feed information to Heuristic B even if it is not enabled directly (for example, it can choose variables that are to be assigned 1 while they persist). Therefore, Heuristic B must have oracle access that would hide syntactical properties of the formula like which of the two opposite literals is positive or which variable has the smallest number.

Acknowledgments. The authors are grateful to Eli Ben-Sasson for helpful discussions and to anonymous referees for numerous comments that improved the quality of this paper.

References

1. D. Achlioptas, P. Beame, and M. Molloy. A sharp threshold in proof complexity. *JCSS,* 2003.
2. D. Achlioptas, P. Beame, and M. Molloy. Exponential bounds for DPLL below the satisfiability threshold. In *SODA'04,* 2004.
3. D. Achlioptas and G. B. Sorkin. Optimal myopic algorithms for random 3-SAT. In *FOCS'00,* 2000.
4. M. Alekhnovich and E. Ben-Sasson. Analysis of the random walk algorithm on random 3-CNFs. Manuscript, 2002.
5. M. Alekhnovich, E. Ben-Sasson, A. Razborov, and A. Wigderson. Pseudorandom generators in propositional complexity. In *FOCS'00,* 2000. Journal version is to appear in *SIAM J. Comp.*
6. M. Alekhnovich and A. Razborov. Lower bounds for the polynomial calculus: non-binomial case. In *FOCS'01,* 2001.
7. E. Ben-Sasson and A. Wigderson. Short proofs are narrow — resolution made simple. *JACM,* 48(2): 149–169, 2001.
8. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Comm. ACM,* 5:394–397, 1962.
9. M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM,* 7:201–215, 1960.
10. E. A. Hirsch. SAT local search algorithms: Worst-case study. *JAR,* 24(1/2) :127–143, 2000.
11. S. I. Nikolenko. Hard satisfiable formulas for DPLL-type algorithms. *Zapiski nauchnyh seminarov POMI,* 293:139–148, 2002. English translation is to appear in *Journal of Mathematical Sciences.*
12. P. Pudlák and R. Impagliazzo. A lower bound for DLL algorithms for k-SAT. In *SODA'00,* 2000.
13. L. Simon, D. Le Berre, and E. A. Hirsch. The SAT 2002 Competition. To appear in *AMAI,* 2002.
14. G. S. Tseitin. On the complexity of derivation in the propositional calculus. *Zapiski nauchnykh seminarov LOMI,* 8:234–259, 1968. English translation of this volume: Consultants Bureau, N.Y., 1970, pp. 115–125.

Linear and Branching Metrics for Quantitative Transition Systems*

Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga

Department of Computer Engineering, Universitity of California, Santa Cruz, USA

Abstract. We extend the basic system relations of trace inclusion, trace equivalence, simulation, and bisimulation to a quantitative setting in which propositions are interpreted not as boolean values, but as real values in the interval [0, 1]. Trace inclusion and equivalence give rise to asymmetrical and symmetrical *linear distances,* while simulation and bisimulation give rise to asymmetrical and symmetrical *branching distances.* We study the relationships among these distances, and we provide a full logical characterization of the distances in terms of quantitative versions of LTL and μ-calculus. We show that, while trace inclusion (resp. equivalence) coincides with simulation (resp. bisimulation) for deterministic boolean transition systems, linear and branching distances do not coincide for deterministic quantitative transition systems. Finally, we provide algorithms for computing the distances, together with matching lower and upper complexity bounds.

1 Introduction

Quantitative transition systems extend the usual transition systems, by interpreting propositions as numbers in [0,1], rather than as truth values. Quantitative transition systems arise in a wide range of contexts. They provide models for optimization problems, where the propositions can be interpreted as rewards, costs, or as the use of resources such as power and memory. They also provide models for discrete-time samplings of continuous systems, where the propositions represent the values of continuous variables at discrete instants of time. We extend the classical relations of trace inclusion, trace equivalence, simulation, and bisimulation to a quantitative setting, by defining linear and branching *distances*[1]. Considering distances, rather than relations, is particularly useful in the quantitative setting, as it leads to a theory of system approximations [5, 16, 1], enabling the quantification of how closely a concrete system implements a specification.

We define two families of distances: *linear distances,* which generalize trace inclusion and equivalence, and *branching distances,* which generalize (bi)simulation. We relate these distances to the quantitative version of the two well-known specification languages LTL and μ-calculus, showing that the distances measure to what extent the logic can tell one system from the other.

* This research was supported in part by the NSF CAREER grant CCR-0132780, the NSF grant CCR-0234690, and the ONR grant N00014-02-1-0671.

[1] In this paper, we use the term "distance" in a generic way, applying it to quantities that are traditionally called pseudometrics and quasi-pseudometrics [7].

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 97–109, 2004.

Our starting point for linear distances is the distance $\|\sigma - \rho\|_\infty$ between two traces σ and ρ, which measures the supremum of the difference in predicate valuations at corresponding positions of σ and ρ. To lift this trace distance to a distance over states, we define $ld^s(s,t) = \sup_{\sigma \in Tr(s)} \inf_{\rho \in Tr(t)} \|\sigma - \rho\|_\infty$, where $Tr(s)$ and $Tr(t)$ are the set of traces from s and t, respectively. The distance $ld^s(s,t)$ is asymmetrical, and is a quantitative extension of trace containment: if $ld^s(s,t) = b$, then for all traces σ from s, there is a trace ρ from t such that $\|\sigma - \rho\|_\infty \leq b$. In particular, $Tr(s) \subseteq Tr(t)$ iff $ld^s(s,t) = 0$. We define a symmetrical version of this distance by $\overline{ld}^s(s,t) = \max\{ld^s(s,t), ld^s(t,s)\}$, yielding a distance that generalizes trace equivalence; thus, $\overline{ld}^s(s,t)$ is the Hausdorff distance between $Tr(s)$ and $Tr(t)$.

We relate the linear distance to the logic QLTL, a quantitative version of LTL [12]. When interpreted on a quantitative transition system, QLTL formulas yield a real value in the interval [0,1]. The formula **"next p"** returns the (quantitative) value of p in the next step of a trace, while **"eventually p"** seeks the maximum value attained by p throughout the trace. The logical connectives "and" and "or" are interpreted as "min" and "max", and **"not x"** is interpreted as $1 - x$. Furthermore, QLTL has a bounded difference operator $\dot{-}$, defined as $x \dot{-} y = \max\{x - y, 0\}$.

In the boolean setting, for a relation to characterize a logic, two states must be related if and only if all formulas from the logic have the same truth value on them. In the quantitative framework, we can achieve a finer characterization: in addition to relating those states that formulas cannot distinguish, we can also *measure* to what extent the logic can tell one state from the other. We show that the linear distances provide such a measure for QLTL: for all states s,t we have $\overline{ld}^s(s,t) = \sup_{\varphi \in \text{QLTL}} |\varphi(s) - \varphi(t)|$ and $ld^s(s,t) = \sup_{\varphi \in \text{QLTL}} (\varphi(s) \dot{-} \varphi(t))$. We investigate what syntactic fragment of QLTL is necessary for such a characterization, showing in particular that the fragment must include the operator $\dot{-}$, in line with the results of [5,11]. We also consider linear distances based on the asymmetric trace distance $\|\sigma \dot{-} \rho\|_\infty$ for traces σ and ρ. Intuitively, if $\|\sigma \dot{-} \rho\|_\infty = b$, then all predicate valuations along ρ are no more than b below the corresponding valuations in σ. Such asymmetrical distances are useful in optimization and control problems, where it is desired to approximate a given quantity from above or below. We show that these distances are characterized by the *positive* fragment of QLTL, in which all propositions occur with positive polarity.

We then study the branching distances that are the analogous of simulation and bisimulation on quantitative systems. A state s simulates a state t via R if the proposition valuations at s and t coincide, and if every successor of s is related via R to some successor of t. We generalize simulation to a distance bd^{As} over states. If $bd^{\text{As}}(s,t) = b$, then $\|s - t\|_\infty < b$, and every successor of s can be matched by a successor of t within bd^{As}**-distance** b. In a similar fashion, we can define a distance bd^{Ss} that is a quantitative analogous of bisimulation; such a distance has been studied in [5,16]. We relate these distances to QMU, a quantitative fixpoint calculus that essentially coincides with the **μ-calculus** of [2], and is related to the calculi of [9,3] (see also [8,13]). In particular, we show that $bd^{\text{Ss}}(s,t) = \sup_{\varphi \in \text{QMU}} |\varphi(s) - \varphi(t)|$ and $bd^{\text{As}}(s,t) = \sup_{\varphi \in \exists\text{QMU}} (\varphi(s) \dot{-} \varphi(t))$, where $\exists$**QMU** is the fragment of QMU in which only existential predecessor operators occur. Similarly, starting from the asymmetrical state distance $\|s \dot{-} t\|_\infty$, we obtain branching distances that are characterized by the corresponding positive fragments of

QMU. As before, these characterizations require the presence of the $\dot{-}$ operator in the calculus.

We relate linear and branching distances, showing that just as simulation implies trace containment, so the branching distances are greater than or equal to the corresponding linear distances. However, while trace inclusion (resp. equivalence) coincides with simulation (resp. bisimulation) for deterministic boolean transition systems, we show that linear and branching distances do not coincide for deterministic quantitative transition systems. Finally, we present algorithms for computing linear and branching distances over quantitative transition systems. We show that the problem of computing the linear distances is PSPACE-complete, and it remains PSPACE-complete even over deterministic systems, showing once more that determinism plays a lesser role in quantitative transition systems. The branching distances can be computed in polynomial time using standard fixpoint algorithms [2].

We also present our results in a *discounted* version, in which distances occurring i steps in the future are multiplied by α^i, where α is a discount factor in $[0,1]$. This discounted setting is common in the theory of games (see e.g. [6]) and optimal control (see e.g. [4]), and it leads to robust theories of quantitative systems [2].

2 Preliminaries

For two numbers $x, y \in [0,1]$, we write $x \sqcup y = \max(x,y)$, $x \sqcap y = \min(x,y)$, $x \dot{+} y = 1 \sqcap (x+y)$ and $x \dot{-} y = 0 \sqcup (x-y)$. We lift the operators $\sqcup$ and $\sqcap$, and the relations $<, \leq$ to functions via their pointwise extensions. Given a function $d : X^2 \to \mathbb{R}^{\geq 0}$, we denote by $\mathrm{Zero}(d) = \{(x,y) \in X^2 \mid d(x,y) = 0\}$ its zero set.

Quantitative transition systems. A *quantitative transition system* (QTS) $\mathcal{S} = (S, \tau, \Sigma, [\cdot])$ consists of a set S of states, a transition relation $\tau \subseteq S \times S$, a finite set Σ of propositions, and a function $[\cdot]\colon S \to (\Sigma \to [0,1])$ which assigns to each state $s \in S$ and proposition $r \in \Sigma$ a value $[s](r)$. For a state $s \in S$, we write $\tau(s)$ for $\{t \in S \mid (s,t) \in \tau\}$. We require that $\mathcal{S}$ is finite-branching and non-blocking: for all $s \in S$, the set $\tau(s)$ is finite and non-empty. We call a function $u : \Sigma \to [0,1]$ a *Σ-valuation*, and we denote by $\mathcal{U}$ the set of all Σ-valuations. A QTS $\mathcal{S}$ is *boolean* if for all $s \in S$ and all $r \in \Sigma$, we have $[s](r) \in \{0,1\}$. A QTS $\mathcal{S}$ is *deterministic* if for all states $s \in S$ and $t, t' \in \tau(s)$ with $t \neq t'$, there is $r \in \Sigma$ such that $[t](r) \neq [t'](r)$. When discussing algorithmic complexity, we assume that values $x \in [0,1]$ are encoded as fixed-point binary numbers, and we denote by $|x|_b$ the number of bits their encoding. We define the size of a (finite) QTS $\mathcal{S} = (S, \tau, \Sigma, [\cdot])$ by $|\mathcal{S}| = \sum_{s \in S} \sum_{r \in \Sigma} |[s](r)|_b + \sum_{s \in S} |\tau(s)|$.

Paths and traces. Given a set A and a sequence $\pi = a_0 a_1 a_2 \cdots \in A^\omega$, we write π_i for the i-th element a_i of π, and we write $\pi^i = a_i a_{i+1} a_{i+2} \cdots$ for the (infinite) suffix of π starting from π_i. A *path of* $\mathcal{S}$ is an infinite sequence $\pi = s_0 s_1 s_2 \cdots$ of states such that $(s_i, s_{i+1}) \in \tau$ for all $i \in \mathbb{N}$. Given a state $s \in S$, we write $\mathit{Pts}(s)$ for the set of all paths starting in s. A *Σ-trace* is an infinite sequence $\sigma = u_0 u_1 u_2 \cdots \in \mathcal{U}^\omega$; we call a Σ-trace simply a trace when Σ is clear from the context. Every path π of $\mathcal{S}$ induces a

Σ-trace $[\pi] = [\pi_0][\pi_1][\pi_2]\cdots$; we write $Tr(s) = \{[\pi] \mid \pi \in Pts(s)\}$ for the set of traces from $s \in S$.

We define simulation, bisimulation, and trace containment for QTS as usual. Specifically, for a QTS $\mathcal{S} = (S, \tau, \Sigma, [\cdot])$, the simulation relation $\preceq_{sim}$ (resp. the bisimulation relation $\approx_{bis}$) is the largest relation $R \subseteq S \times S$ such that, for all sRt, the following conditions (i) and (ii) (resp. (i), (ii), and (iii)) hold: (i) $[s] = [t]$; (ii) for all $s' \in \tau(s)$, there is $t' \in \tau(t)$ with $s'Rt'$; (iii) for all $t' \in \tau(t)$, there is $s' \in \tau(s)$ with $s'Rt'$. For $s, t \in S$, we write $s \sqsubseteq_{tr} t$ if $Tr(s) \subseteq Tr(t)$, and $s \equiv_{tr} t$ if $Tr(s) = Tr(t)$.

Directed metrics and pseudometrics. A *directed metric* on X is a function $d : X \times X \to \mathbb{R}^{\geq 0}$ that satisfies $d(x,x) = 0$ for all $x \in X$ and the triangle inequality: $d(x,z) \leq d(x,y) + d(y,z)$ for all $x, y, z \in X$. A *pseudometric* d is a directed metric that is symmetric, i.e. $d(x,y) = d(y,x)$ for all $x, y \in X$. Given a directed metric, we denote by $\bar{d}$ its *symmetrization,* defined by $\bar{d}(s,t) = d(s,t) \sqcup d(t,s)$.

We develop our definitions in terms of directed metrics. Given a directed metric d on X and a mapping $q : X \to [0,1]$, the "directed" bound $d(x,y) \geq q(x) \dot{-} q(y)$ for all $x, y \in X$ immediately yields the "symmetrical" bound $\bar{d}(x,y) \geq |q(x) - q(y)|$ for all $x, y \in X$. Hence, we focus on directed metrics and directed bounds, deriving the symmetrical results through the above observation.

3 Linear Distances and Logics

Throughout this paper, unless specifically noted, we consider a fixed a QTS $\mathcal{S} = (S, \tau, \Sigma, [\cdot])$. The propositional distance between two states measures the maximum difference in their proposition evaluations.

Definition 1 (propositional distance) We define the *propositional distance* $pd : \mathcal{U}^2 \to [0,1]$, for all $u, v \in \mathcal{U}$, as $pd(u,v) = \max_{r \in \Sigma} (u(r) \dot{-} v(r))$. ∎

For ease of notation, we write $pd(s,t)$ for $pd([s],[t])$. For $u, v \in \mathcal{U}$ we have $(u,v) \in \mathrm{Zero}(\overline{pd})$ iff $u = v$, and $(u,v) \in \mathrm{Zero}(pd)$ iff $u(r) \leq v(r)$ for all $r \in \Sigma$. The definition of trace distance discounts the propositional distance at positions i of the trace by multiplying it by α^i, for $\alpha \in [0,1]$.

Definition 2 (trace distance) We define the *trace distance* $td_\alpha : \mathcal{U}^\omega \to [0,1]$ by letting, for $\sigma, \rho \in \mathcal{U}^\omega$ and $\alpha \in [0,1]$, $td_\alpha(\sigma,\rho) = \sup_{i \in \mathbb{N}} \alpha^i pd(\sigma_i, \rho_i)$. ∎

For $\alpha = 1$, the definitions reduce to the classical notions of trace distance: $td_1(\sigma,\rho) = \|\sigma \dot{-} \rho\|_\infty$, and $\overline{td}_1(\sigma,\rho) = \|\sigma - \rho\|_\infty$. We note that $\overline{td}_\alpha$ is a generalization of the Cantor metric, which equals $\overline{td}_{1/2}$. Lifting $\leq$ and $=$ to traces in a pointwise fashion, for all $\sigma, \rho \in \mathcal{U}^\omega$ and $\alpha \in (0,1]$ we have that $(\sigma,\rho) \in \mathrm{Zero}(\overline{td}_\alpha)$ iff $\sigma = \rho$, and $(\sigma,\rho) \in \mathrm{Zero}(td_\alpha)$ iff $\sigma \leq \rho$. The linear distances between two states are obtained by lifting trace distances to the set of all traces from the two states, as in the definition of Hausdorff distance.

Definition 3 (linear distance) We define the two *linear distances* ld^a and ld^s over S as follows, for $s, t \in S$ and $\alpha \in [0,1]$:

$$ld^a_\alpha(s,t) = \sup_{\sigma \in Tr(s)} \inf_{\rho \in Tr(t)} td_\alpha(\sigma,\rho) \qquad ld^s_\alpha(s,t) = \sup_{\sigma \in Tr(s)} \inf_{\rho \in Tr(t)} \overline{td}_\alpha(\sigma,\rho)$$ ∎

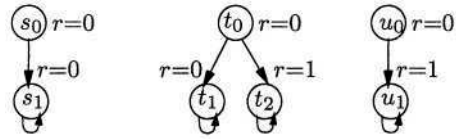


Fig. 1. A QTS showing the difference between ld^{a}_α, ld^{s}_α, $\overline{ld}^{\mathrm{a}}_\alpha$, and $\overline{ld}^{\mathrm{s}}_\alpha$.

One can easily check that, for all $\alpha \in [0,1]$, the functions ld^{a}_α, ld^{s}_α are directed metrics and $\overline{ld}^{\mathrm{a}}_\alpha$, $\overline{ld}^{\mathrm{s}}_\alpha$ are pseudometrics. Intuitively, the distance ld^{s}_α is a quantitative extension of trace containment: for $s,t \in S$, the distance $ld^{\mathrm{s}}_\alpha(s,t)$ measures how closely (in a quantitative sense) can a trace from t simulate a trace from s. The symmetrization of ld^{s}_α is $\overline{ld}^{\mathrm{s}}_\alpha$, which is related to trace equivalence. Indeed, we will see that the valuation of QLTL formulas at s and t can differ by at most $\overline{ld}^{\mathrm{s}}_\alpha(s,t)$, and similarly, the valuation of any QLTL formula at t is at most $ld^{\mathrm{s}}_\alpha(s,t)$ below the valuation at s.

Theorem 1 *For all $\alpha \in (0,1]$, we have $\sqsubseteq_{tr} = \mathrm{Zero}(ld^{\mathrm{s}}_\alpha)$ and $\equiv_{tr} = \mathrm{Zero}(\overline{ld}^{\mathrm{s}}_\alpha)$.*

For $\alpha = 1$, the distances ld^{a}_α and $\overline{ld}^{\mathrm{a}}_\alpha$ have the following intuitive characterization. For a trace $\sigma \in \mathcal{U}^\omega$ and $c \in \mathbb{R}$, denote by $\sigma \dot{-} c$ the trace defined by $(\sigma \dot{-} c)_k(r) = \sigma_k(r) \dot{-} c$ for all $k \in \mathbb{N}$ and $r \in \Sigma$: in other words, $\sigma \dot{-} c$ is obtained from σ by decreasing all proposition valuations by c. For all $s,t \in S$, if $ld^{\mathrm{a}}_1(s,t) = c$ then for every trace σ from s there is a trace ρ from t such that $\rho \geq \sigma \dot{-} c$. This means that $ld^{\mathrm{a}}_1(s,t)$ is a "positive" version of trace containment: for each trace σ of s, the goal of a trace ρ from t is not that of being close to σ, but rather, that of not being below $\sigma \dot{-} c$. This version of trace containment preserves within c the valuation of QLTL formulas that have only positive occurrences of propositions (called positive QLTL formulas). The relations among linear distances are summarized by the following theorem.

Theorem 2 *The relations in Figure 4(a) hold for all $\alpha \in [0,1]$. Moreover, for $\alpha \in (0,1]$ the inequalities cannot be replaced by equalities.*

Proof. The inequalities are immediate. For $\alpha \in (0,1]$ and the QTS in Figure 1, we have

$$\begin{array}{lll}
ld^{\mathrm{a}}_\alpha(s_0,t_0) = 0 & ld^{\mathrm{a}}_\alpha(t_0,u_0) = 0 & ld^{\mathrm{a}}_\alpha(u_0,t_0) = 0 \\
ld^{\mathrm{s}}_\alpha(s_0,t_0) = 0 & ld^{\mathrm{s}}_\alpha(t_0,u_0) = \alpha & ld^{\mathrm{s}}_\alpha(u_0,t_0) = 0 \\
\overline{ld}^{\mathrm{a}}_\alpha(s_0,t_0) = \alpha & \overline{ld}^{\mathrm{a}}_\alpha(t_0,u_0) = 0 & \overline{ld}^{\mathrm{a}}_\alpha(u_0,t_0) = 0 \\
\overline{ld}^{\mathrm{s}}_\alpha(s_0,t_0) = \alpha & \overline{ld}^{\mathrm{s}}_\alpha(t_0,u_0) = \alpha & \overline{ld}^{\mathrm{s}}_\alpha(u_0,t_0) = \alpha
\end{array}$$

Thus, we have an example where $ld^{\mathrm{a}}_\alpha \neq ld^{\mathrm{s}}_\alpha$, $ld^{\mathrm{a}}_\alpha \neq \overline{ld}^{\mathrm{a}}_\alpha$, $ld^{\mathrm{s}}_\alpha \neq \overline{ld}^{\mathrm{s}}_\alpha$, $\overline{ld}^{\mathrm{a}}_\alpha \neq \overline{ld}^{\mathrm{s}}_\alpha$, and neither $ld^{\mathrm{s}}_\alpha \leq \overline{ld}^{\mathrm{a}}_\alpha$ nor $ld^{\mathrm{s}}_\alpha \geq \overline{ld}^{\mathrm{a}}_\alpha$. ∎

3.1 Quantitative Linear-Time Temporal Logic

The linear distances introduced above can be characterlized in terms *quantitative linear-time temporal logic* (QLTL), a quantitative extension of linear-time temporal logic [12]

which includes quantitative versions of the temporal operators and logic connectives. Following [5], QLTL has a "threshold" operator, enabling the comparison of a formula against a constant in the interval [0,1]. The QLTL formulas over a set Σ of propositions are generated by the following grammar:

$$\varphi ::= r \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid c \dot{+} \varphi \mid c \dot{-} \varphi \mid \bigcirc_\alpha \varphi \mid \ominus_\alpha \varphi \mid \Diamond_\alpha \varphi \mid \Box_\alpha \varphi$$

Here $r \in \Sigma$ is a proposition, $c \in [0,1]$ a constant and $\alpha \in [0,1]$ a discount factor. A formula φ assigns a value $[\![\varphi]\!](\rho) \in [0,1]$ to each trace $\sigma \subseteq \mathcal{U}^\omega$:

$$\begin{aligned}
[\![r]\!](\sigma) &= \sigma_0(r) \\
[\![\neg\varphi]\!](\sigma) &= 1 - [\![\varphi]\!](\sigma) \qquad & [\![\varphi_1 \wedge \varphi_2]\!](\sigma) &= [\![\varphi_1]\!](\sigma) \sqcap [\![\varphi_2]\!](\sigma) \\
[\![c \dot{+} \varphi]\!](\sigma) &= c \dot{+} [\![\varphi]\!](\sigma) & [\![\varphi_1 \vee \varphi_2]\!](\sigma) &= [\![\varphi_1]\!](\sigma) \sqcup [\![\varphi_2]\!](\sigma) \\
[\![c \dot{-} \varphi]\!](\sigma) &= c \dot{-} [\![\varphi]\!](\sigma) & [\![\Diamond_\alpha \varphi]\!](\sigma) &= \sup\{\alpha^i \cdot [\![\varphi]\!](\sigma^i) \mid i \geq 0\} \\
[\![\bigcirc_\alpha \varphi]\!](\sigma) &= \alpha \cdot [\![\varphi]\!](\sigma^1) & [\![\Box_\alpha \varphi]\!](\sigma) &= \inf\{1 - \alpha^i \cdot (1 - [\![\varphi]\!](\sigma^i)) \mid i \geq 0\} \\
[\![\ominus_\alpha \varphi]\!](\sigma) &= 1 - \alpha + \alpha \cdot [\![\varphi]\!](\sigma^1)
\end{aligned}$$

A QLTL formula φ assings a real value $[\![\varphi]\!](s) \in [0,1]$ to each state s of a given QTS, by defining[2] $[\![\varphi]\!](s) = \sup\{[\![\varphi]\!](\rho) \mid \rho \in Tr(s)\}$. Thanks to the equivalences $\neg\bigcirc_\alpha\varphi \equiv \ominus_\alpha\neg\varphi$, $\neg(c \dot{+} \varphi) \equiv ((1-c) \dot{-} \varphi)$, $\neg(c \dot{-} \varphi) \equiv ((1-c) \dot{+} \varphi)$, $\neg(\Diamond_\alpha\varphi) \equiv \Box_\alpha\neg\varphi$, and the classical dualities between $\wedge$, $\vee$, μ, and ν, the syntax of QLTL allows negations to be pushed to the atomic propositions without affecting the value of a formula. For $\alpha \in [0,1]$, we denote by $\mathbf{QLTL}_\alpha$ the set of formulas containing only discount factors smaller than or equal to α. All QLTL operators are *positive,* with the exception of $\neg$ and $c \dot{-}$ for $c \in [0,1]$, which are *negative.* We say that a QLTL formula is *positive* if all propositions occur with positive polarity, that is, within an even number of negative operators; we denote by $\mathbf{QLTL}^+_\alpha$ the positive fragment of $\mathbf{QLTL}_\alpha$. Furthermore, for $ops \subseteq \{\bigcirc, \ominus, \Diamond, \Box, \dot{+}, \dot{-}\}$, we denote by $\mathbf{QLTL}_\alpha(ops)$ the set of formulas which only contain boolean connectives and operators in *ops.* We denote by $\mathbf{QLTL}^+_\alpha(ops)$ the restrictions of these sets to positive formulas. Notice that for $\alpha = 1$, $\bigcirc_\alpha$ and $\ominus_\alpha$ coincide with the usual $\bigcirc$ operator of LTL. Thus, if we forbid the use of $\dot{+}$ and $\dot{-}$ and we take all discount factors to be 1, the semantics of QLTL on boolean systems coincides with the one of LTL.

3.2 Logical Characterization of Linear Distances

Linear distances provide a bound for the difference in valuation of QLTL formulas. We begin by relating distances and logics over traces.

Lemma 1 *For all $\alpha \in [0,1]$ and all traces $\sigma, \rho \in \mathcal{U}^\omega$, the following holds.*

$$\begin{aligned}
&\textit{For all } \varphi \in \mathbf{QLTL}^+_\alpha: \quad td_\alpha(\sigma,\rho) \geq [\![\varphi]\!](\sigma) \dot{-} [\![\varphi]\!](\rho); \\
&\textit{For all } \varphi \in \mathbf{QLTL}_\alpha: \quad \overline{td}_\alpha(\sigma,\rho) \geq |[\![\varphi]\!](\sigma) - [\![\varphi]\!](\rho)|.
\end{aligned}$$

[2] We chose to give the existential interpretation of QLTL. Obviously, the minimum value of φ from s is obtained by 1 minus the maximum value of $\neg\varphi$ in s.

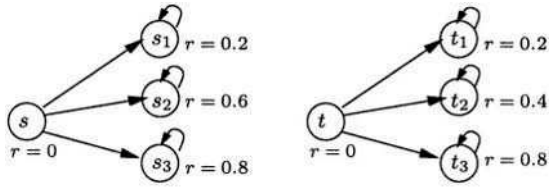


Fig. 2. QLTL cannot distinguish between s and t.

The following theorem uses the linear distances to provide the desired bounds for QLTL.

Theorem 3 *For all* $\alpha \in [0,1]$ *and* $s,t \in S$, *we have:*

$$\textit{For all } \varphi \in \mathrm{QLTL}_\alpha^+ : ld_\alpha^{\mathrm{a}}(s,t) \geq [\![\varphi]\!](s) \dot{-} [\![\varphi]\!](t) \textit{ and } \overline{ld}_\alpha^{\mathrm{a}}(s,t) \geq |[\![\varphi]\!](s) - [\![\varphi]\!](t)|,$$
$$\textit{For all } \varphi \in \mathrm{QLTL}_\alpha : ld_\alpha^{\mathrm{s}}(s,t) \geq [\![\varphi]\!](s) \dot{-} [\![\varphi]\!](t) \textit{ and } \overline{ld}_\alpha^{\mathrm{s}}(s,t) \geq |[\![\varphi]\!](s) - [\![\varphi]\!](t)|.$$

The results for ld_α^{s} and $\overline{ld}_\alpha^{\mathrm{s}}$ are the quantitative analogous of the standard connection between trace containment and trace equivalence, and LTL. For instance, the result about ld_α^{s} states that, if $ld_\alpha^{\mathrm{s}}(s,t) = c$, then for every formula $\varphi \in \mathrm{QLTL}_\alpha$ and every trace σ from s, there is a trace ρ from t such that $[\![\varphi]\!](\rho) \geq [\![\varphi]\!](\sigma) - c$.

The following theorem states that the linear distances can be characterized by a syntactic subset of the logics that includes only the $\bigcirc$ and $\dot{+}$ operators, in addition to boolean connectives. Together with Theorem 3, this result constitutes a full characterization of linear distances in terms of QLTL.

Theorem 4 *For all* $\alpha \in [0,1]$ *and* $s,t \in S$,

$$ld_\alpha^{\mathrm{a}}(s,t) = \sup_{\varphi \in \mathrm{QLTL}_\alpha^+(\bigcirc,\dot{+})} [\![\varphi]\!](s) \dot{-} [\![\varphi]\!](t) \qquad \overline{ld}_\alpha^{\mathrm{a}}(s,t) = \sup_{\varphi \in \mathrm{QLTL}_\alpha^+(\bigcirc,\dot{+})} |[\![\varphi]\!](s) - [\![\varphi]\!](t)|$$
$$ld_\alpha^{\mathrm{s}}(s,t) = \sup_{\varphi \in \mathrm{QLTL}_\alpha(\bigcirc,\dot{+})} [\![\varphi]\!](s) \dot{-} [\![\varphi]\!](t) \qquad \overline{ld}_\alpha^{\mathrm{s}}(s,t) = \sup_{\varphi \in \mathrm{QLTL}_\alpha(\bigcirc,\dot{+})} |[\![\varphi]\!](s) - [\![\varphi]\!](t)|$$

The next result shows that the operator $\dot{+}$ is indeed necessary to obtain such a characterization ($\bigcirc$ is also trivially necessary). This result is reminiscent of a result by [5] for Markov systems.

Theorem 5 *There is a finite QTS and two states* s *and* t *such that, for all* $\alpha \in (0,1]$, $\overline{ld}_\alpha^{\mathrm{s}}(s,t) = ld_\alpha^{\mathrm{s}}(s,t) > 0$, *and* $\sup_{\varphi \in \mathrm{QLTL}_\alpha(\bigcirc,\ominus,\Diamond,\Box)} |[\![\varphi]\!](s) - [\![\varphi]\!](t)| = 0$.

As an example, consider the QTS in Figure 2. It holds that $\overline{ld}_1^{\mathrm{s}}(s,t) = ld_1^{\mathrm{s}}(s,t) = 0.2$. A suitable formula for distinguishing s and t is $\varphi : \bigcirc[(0.6 \dot{+} \neg r) \wedge (0.4 \dot{+} r)]$; we have $\varphi(s) = 1$ and $\varphi(t) = 0.8$. On the other hand, it can be proved by induction on the structure of the formula that, if $\dot{+}$ and $\dot{-}$ are not used, there is no QLTL formula that distinguishes between s and t.

3.3 Computing the Linear Distance

Given a finite QTS $\mathcal{S} = (S, \tau, \Sigma, [\cdot])$ we wish to compute $ld^x_\alpha(s_0, t_0)$, for all $s_0, t_0 \in S$, all $x \in \{\mathrm{a}, \mathrm{s}\}$, and all $\alpha \in (0, 1]$ (the case $\alpha = 0$ is trivial). We describe the computation of ld^{a}_α, as the computation of ld^{s}_α is analogous. We can read the definition of ld^{a}_α as a two-player game. Player 1 chooses a path $\pi = s_0 s_1 s_2 \cdots$ from s_0; Player 2 chooses a path $\pi' = t_0 t_1 t_2 \cdots$ from t_0; the goal of Player 1 (resp. Player 2) is to maximize (resp. minimize) $\sup_k \alpha^k pd(\pi_k, \pi'_k)$. The game is played with partial information: after $s_0 \cdots s_n$, Player 1 must choose s_{n+1} without knowledge[3] of $t_0 \cdots t_n$. Such a game can be solved via a variation of the subset construction [14]. The key idea is to associate with each final state s_n of a finite path $s_0 s_1 \cdots s_n$ chosen by Player 1, all final states t_n of finite paths $t_0 t_1 \cdots t_n$ chosen by Player 2, each labeled by the distance $v(s_0 \cdots s_n, t_0 \cdots t_n) = \max_{0 \le k \le n} \alpha^{k-n} pd(s_k, t_k)$.

From $\mathcal{S}$, we construct another QTS $\mathcal{S}' = (S', \tau', \{r\}, [\cdot]')$, having set of states $S' = S \times 2^{S \times \mathbb{D}}$. If $\alpha = 1$ we can take $\mathbb{D} = \{pd(s,t) \mid s,t \in S\}$, so that $|\mathbb{D}| \le |S|^2$. For $\alpha \in (0,1)$, we take $\mathbb{D} = \{pd(s,t)/\alpha^k \mid s,t \in S \wedge k \in \mathbb{N} \wedge pd(s,t) \le \alpha^k\} \cup \{1\}$, so that $|\mathbb{D}| \le |S|^2 \cdot \lceil \log_\alpha \min\{pd(s,t) \mid s,t \in S \wedge pd(s,t) > 0\} \rceil + 1$. The transition relation τ' consists of all pairs $(\langle s, C\rangle, \langle s', C'\rangle)$ such that $s' \in \tau(s)$ and $C' = \{\langle t', v'\rangle \mid \exists \langle t, v\rangle \in C \,.\, t' \in \tau(t) \wedge v' = (v/\alpha \sqcup pd(s', t')) \sqcap 1\}$. Note that only Player 1 has a choice of moves in this game, since the moves of Player 2 are accounted for by the subset construction. Finally, the interpretation $[\cdot]'$ is given by $[\langle s, C\rangle]'(r) = \min\{v \mid \langle t, v\rangle \in C\}$, so that r indicates the minimum distance achievable by Player 2 while trying to match a path to $\langle s, C\rangle$ chosen by Player 1. The goal of the game, for Player 1, consists in reaching a state of $\mathcal{S}'$ with the highest possible (discounted) value or r. Thus, for all $s, t \in S$, we have $ld^{\mathrm{a}}_\alpha(s,t) = [\![\exists \Diamond_\alpha r]\!]_{\mathcal{S}'}(\langle s, \{\langle t, pd(s,t)\rangle\}\rangle)$, where the right-hand side is to be computed on $\mathcal{S}'$. This expression can be evaluated by a depth-first traversal of the state space of $\mathcal{S}'$, noting that no state of $\mathcal{S}'$ needs to be visited twice, as subsequent visits do not increase the value of $\Diamond_\alpha r$.

Theorem 6 *For all $x \in \{\mathrm{a}, \mathrm{s}\}$, the following assertions hold:*

1. *Computing ld^x_α for $\alpha \in [0,1]$ and QTS $\mathcal{S}$ is PSPACE-complete in $|\mathcal{S}| + |\alpha|_b$.*
2. *Computing ld^x_α for $\alpha \in [0,1]$ and deterministic QTS $\mathcal{S}$ is PSPACE-complete in $|\mathcal{S}| + |\alpha|_b$.*
3. *Computing ld^x_α for $\alpha \in [0,1]$ and boolean, deterministic QTS $\mathcal{S}$ is in time $O(|\mathcal{S}|^4)$.*

The upper complexity bound for part 1 comes from the above algorithm; the lower bound comes from a reduction from the corresponding result for trace inclusion [15]. Part 2 states that, unlike in the boolean case, the problem remains PSPACE-complete even for deterministic QTSs. This result is proved via a logspace reduction: by introducing perturbations in the valuations, we can tranform a nondeterministic QTS into a deterministic one; for appropriately small perturbations, the distances computed on the derived deterministic QTS enable the determination of the distances over the nondeterministic QTS. Finally, part 3 is a consequence of Theorems 13 and 12.

[3] Indeed, if the game were played with total information, we would obtain the branching distances of the next section.

4 Branching Distances and Logics

Definition 4 (branching distances) Consider the following four equations involving the function $d : S^2 \to [0,1]$ and the parameter $\alpha \in [0,1]$.

$$\text{(Aa)} \quad d(s,t) = pd(s,t) \sqcup \alpha \cdot \max_{s' \in \tau(s)} \min_{t' \in \tau(t)} d(s',t')$$

$$\text{(As)} \quad d(s,t) = \overline{pd}(s,t) \sqcup \alpha \cdot \max_{s' \in \tau(s)} \min_{t' \in \tau(t)} d(s',t')$$

$$\text{(Sa)} \quad d(s,t) = pd(s,t) \sqcup \alpha \cdot \max_{s' \in \tau(s)} \min_{t' \in \tau(t)} d(s',t') \sqcup \alpha \cdot \max_{t' \in \tau(t)} \min_{s' \in \tau(s)} d(s',t')$$

$$\text{(Ss)} \quad d(s,t) = \overline{pd}(s,t) \sqcup \alpha \cdot \max_{s' \in \tau(s)} \min_{t' \in \tau(t)} d(s',t') \sqcup \alpha \cdot \max_{t' \in \tau(t)} \min_{s' \in \tau(s)} d(s',t')$$

For $x \in \{\text{Aa}, \text{As}, \text{Sa}, \text{Ss}\}$, we define the branching distance bd_α^x as the smallest function $d : S^2 \to [0,1]$ satisfying the equation (x). ∎

For all $\alpha \in [0,1]$, the functions bd_α^{Aa}, bd_α^{As}, and bd_α^{Sa} are directed metrics, and the functions bd_α^{Ss}, $\overline{bd}_\alpha^{\text{Aa}}$, $\overline{bd}_\alpha^{\text{As}}$, and $\overline{bd}_\alpha^{\text{Sa}}$ are pseudometrics.

The distance bd_α^{Ss} is a quantitative generalization of bisimulation, and it coincides essentially with the metrics of [5,16,2]; as it is already symmetrical, we have $\overline{bd}_\alpha^{\text{Ss}} = bd_\alpha^{\text{Ss}}$. Similarly, the distance bd_α^{As} generalizes simulation, and $\overline{bd}_\alpha^{\text{As}}$ generalizes mutual simulation.

Theorem 7 *For all* $\alpha \in (0,1]$, *we have* $\preceq_{sim} = \text{Zero}(bd_\alpha^{\text{As}})$ *and* $\approx_{bis} = \text{Zero}(bd_\alpha^{\text{Ss}})$.

The distances bd_α^{Aa} and bd_α^{Sa} correspond to quantitative notions of simulation and bisimulation with respect to the asymmetrical propositional distance *pd;* in particular, if $bd_\alpha^{\text{Aa}}(s,t) = 0$ (that is, if s is related to t), then $[s] \leq [t]$. These distances are not symmetrical, and we indicate their symmetrical versions by $\overline{bd}_\alpha^{\text{Aa}}$ and $\overline{bd}_\alpha^{\text{Sa}}$. The distance bd_α^{Aa} generalizes a boolean notion of simulation proposed in [10] for the preservation of *positive* ACTL formulas, that is, ACTL formulas where all propositions occur with positive polarity; a similar characterization holds for bd_α^{Aa}.

Just as in the boolean case mutual similarity is not equivalent to bisimulation, so in our quantitative setting $\overline{bd}_\alpha^{\text{As}}$ can be strictly smaller than bd_α^{Ss}, and $\overline{bd}_\alpha^{\text{Aa}}$ can be strictly smaller than bd_α^{Sa}.

Theorem 8 *The relations in Figure 4(b) hold for all QTS and for all* $\alpha \in [0,1]$. *For* $\alpha \in (0,1]$, *no other inequalities hold on all QTSs.*

4.1 Quantitative μ-Calculus

We define quantitative **μ-calculus** after [3,2]. Given a set of variables X and a set of propositions Σ, the formulas of the *quantitative* ***μ-calculus*** are generated by the grammar:

$$\varphi ::= r \mid x \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid c \dot{+} \varphi \mid c \dot{-} \varphi \mid$$
$$\exists \bigcirc_\alpha \varphi \mid \exists \Theta_\alpha \varphi \mid \forall \bigcirc_\alpha \varphi \mid \forall \Theta_\alpha \varphi \mid \mu x \,.\, \varphi \mid \nu x \,.\, \varphi$$

for propositions $r \in \Sigma$, variables $x \in X$, constants $c \in [0,1]$, and discount factors $\alpha \in [0,1]$. Denoting by $\mathcal{F} = (S \to [0,1])$, a (variable) interpretation is a function $\mathcal{E} : X \to \mathcal{F}$. Given an interpretation $\mathcal{E}$, a variable $x \in X$ and a function $f \in \mathcal{F}$, we denote by $\mathcal{E}[x := f]$ the interpretation $\mathcal{E}'$ such that $\mathcal{E}'(x) = f$ and, for all $y \neq x$, $\mathcal{E}'(y) = \mathcal{E}(y)$. Given a QTS and an interpretation $\mathcal{E}$, every formula φ of the quantitative μ-**calculus** defines a valuation $[\![\varphi]\!]_\mathcal{E} : S \to [0,1]$:

$$
\begin{array}{ll}
[\![r]\!]_\mathcal{E}(s) & = [s](r) \\
[\![x]\!]_\mathcal{E} & = \mathcal{E}(x) \\
[\![\varphi_1 \wedge \varphi_2]\!]_\mathcal{E} & = [\![\varphi_1]\!]_\mathcal{E} \sqcap [\![\varphi_2]\!]_\mathcal{E} \\
[\![\varphi_1 \vee \varphi_2]\!]_\mathcal{E} & = [\![\varphi_1]\!]_\mathcal{E} \sqcup [\![\varphi_2]\!]_\mathcal{E} \\
[\![\neg\varphi]\!]_\mathcal{E}(s) & = 1 - [\![\varphi]\!]_\mathcal{E}(s) \\
[\![c \dot{+} \varphi]\!]_\mathcal{E}(s) & = c \dot{+} [\![\varphi]\!]_\mathcal{E}(s) \\
[\![c \dot{-} \varphi]\!]_\mathcal{E}(s) & = c \dot{-} [\![\varphi]\!]_\mathcal{E}(s)
\end{array}
\qquad
\begin{array}{ll}
[\![\exists \bigcirc_\alpha \varphi]\!]_\mathcal{E}(s) & = \alpha \cdot \max_{s' \in \tau(s)} [\![\varphi]\!]_\mathcal{E}(s') \\
[\![\exists \ominus_\alpha \varphi]\!]_\mathcal{E}(s) & = 1 - \alpha + \alpha \cdot \max_{s' \in \tau(s)} [\![\varphi]\!]_\mathcal{E}(s') \\
[\![\forall \bigcirc_\alpha \varphi]\!]_\mathcal{E}(s) & = \alpha \cdot \min_{s' \in \tau(s)} [\![\varphi]\!]_\mathcal{E}(s') \\
[\![\forall \ominus_\alpha \varphi]\!]_\mathcal{E}(s) & = 1 - \alpha + \alpha \cdot \min_{s' \in \tau(s)} [\![\varphi]\!]_\mathcal{E}(s') \\
[\![\mu x \,.\, \varphi]\!]_\mathcal{E} & = \inf\{f \in \mathcal{F} \mid f = [\![\varphi]\!]_{\mathcal{E}[x:=f]}\} \\
[\![\nu x \,.\, \varphi]\!]_\mathcal{E} & = \sup\{f \in \mathcal{F} \mid f = [\![\varphi]\!]_{\mathcal{E}[x:=f]}\}.
\end{array}
$$

The existence of the required fixpoints is guaranteed by the monotonicity and continuity of all operators. If φ is closed, we write $[\![\varphi]\!]$ for $[\![\varphi]\!]_\mathcal{E}$. A formula is *positive* if all atomic propositions occur in the scope of an even number of negations. For all $\alpha \in [0,1]$, we call $\textsc{CLMUCALC}_\alpha$ the set of closed μ–**calculus** formulas where all discount factors are smaller than or equal to α and $\textsc{CLMUCALC}^+_\alpha$ the subset of $\textsc{CLMUCALC}_\alpha$ that only contains positive formulas. We denote by $\exists\textsc{CLMUCALC}_\alpha$, $\exists\textsc{CLMUCALC}^+_\alpha$ the respective subsets with no occurrences of $\forall$. For $ops \subseteq \{\bigcirc, \ominus, \Diamond, \Box, \dot{+}, \dot{-}, \mu, \nu, \exists, \forall\}$, we denote by $\textsc{CLMUCALC}_\alpha(ops)$ the set of formulas that only contain boolean connectives and operators in *ops*. Notice that, if we omit the operators $\dot{+}$ and $\dot{-}$ and we take all discount factors to be 1, then the semantics of the quantitative μ-**calculus** on boolean systems coincides with the one of the classical μ-**calculus**.

4.2 Logical Characterizations of Branching Distances

The following result shows that the branching distances provide bounds for the corresponding fragments of the μ-**calculus**. The statement for bd^{Ss}_α is essentially from [5].

Theorem 9 *For all QTSs, states* s *and* t*, and* $\alpha \in [0,1]$*, we have*

$$
\begin{array}{ll}
\textit{for all } \varphi \in \exists\textsc{CLMUCALC}^+_\alpha & bd^{\mathrm{Aa}}_\alpha(s,t) \geq [\![\varphi]\!](s) \dot{-} [\![\varphi]\!](t) \\
\textit{for all } \varphi \in \exists\textsc{CLMUCALC}_\alpha & bd^{\mathrm{As}}_\alpha(s,t) \geq [\![\varphi]\!](s) \dot{-} [\![\varphi]\!](t) \\
\textit{for all } \varphi \in \textsc{CLMUCALC}^+_\alpha & bd^{\mathrm{Sa}}_\alpha(s,t) \geq [\![\varphi]\!](s) \dot{-} [\![\varphi]\!](t) \\
\textit{for all } \varphi \in \textsc{CLMUCALC}_\alpha & bd^{\mathrm{Ss}}_\alpha(s,t) \geq |[\![\varphi]\!](s) - [\![\varphi]\!](t)|
\end{array}
$$

As noted before, each bound of the form $d(s,t) \geq [\![\varphi]\!](s) \dot{-} [\![\varphi]\!](t)$, trivially leads to a bound of the form $\overline{d}(s,t) \geq |[\![\varphi]\!](s) - [\![\varphi]\!](t)|$. The bounds are tight, and the following theorem identifies which fragments of quantitative μ-**calculus** suffice for characterizing each branching distance.

Theorem 10 *For all QTSs, states s and t, and $\alpha \in [0,1]$, we have*

$$bd_\alpha^{\mathrm{Aa}}(s,t) = \sup_{\varphi \in \mathrm{CLMUCALC}_\alpha^+(\exists,\ominus,\dot{+})} [\![\varphi]\!](s) \mathbin{\dot{-}} [\![\varphi]\!](t),$$
$$bd_\alpha^{\mathrm{As}}(s,t) = \sup_{\varphi \in \mathrm{CLMUCALC}_\alpha(\exists,\ominus,\dot{+})} [\![\varphi]\!](s) \mathbin{\dot{-}} [\![\varphi]\!](t),$$
$$bd_\alpha^{\mathrm{Sa}}(s,t) = \sup_{\varphi \in \mathrm{CLMUCALC}_\alpha^+(\exists,\forall,\ominus,\dot{+})} [\![\varphi]\!](s) \mathbin{\dot{-}} [\![\varphi]\!](t),$$
$$bd_\alpha^{\mathrm{Ss}}(s,t) = \sup_{\varphi \in \mathrm{CLMUCALC}_\alpha(\exists,\forall,\ominus,\dot{+})} [\![\varphi]\!](s) \mathbin{\dot{-}} [\![\varphi]\!](t).$$

The next result shows that the operator $\dot{+}$ (or $\dot{-}$), which is not present in the ordinary μ-calculus, is necessary to characterize the branching distances. This parallels a result of [5] for a metric related to bd_α^{Ss} on labeled Markov chains, and a result of [11] for Markov decision processes and games.

Theorem 11 *There is a finite QTS and two states s and t such that, for all $\alpha \in (0,1]$, $bd_\alpha^{\mathrm{Ss}}(s,t) = bd_\alpha^{\mathrm{As}}(s,t) > 0$ and for all $\varphi \in \mathrm{CLMUCALC}$ that do not contain $\dot{+}$ and $\dot{-}$, we have $[\![\varphi]\!](s) = [\![\varphi]\!](t)$.*

Proof (sketch). Consider again the QTS in Figure 2 and take $\alpha = 1$. Then $bd_\alpha^{\mathrm{Ss}}(s,t) = bd_\alpha^{\mathrm{As}}(s,t) = 0.2$. Theorem 5 states that formulas from $\mathrm{QLTL}(\bigcirc, \Diamond)$ are not sufficient for distinguishing s from t. Compared to QLTL, the μ-calculus allows to specify branching formulas and take fixpoints of expressions. However, in the example here, these capabilities do not help, since starting from s or t the only branching point occurs in the first state. ∎

4.3 Computing the Branching Distances

Given a finite QTS $\mathcal{S} = (S, \tau, \Sigma, [\cdot])$ a rational number $\alpha \in [0,1]$, and $x \in \{\mathrm{Ss}, \mathrm{Sa}, \mathrm{As}, \mathrm{Aa}\}$, we can compute $bd_\alpha^x(s,t)$ for all states $s,t \in S$ by computing in an iterative fashion the fixpoints of Definition 4. For instance, bd_α^{Aa} can be computed by letting $d^0(s,t) = 0$ for all $s,t \in S$ and, for $k \in \mathbb{N}$, by letting $d^{k+1}(s,t) = pd(s,t) \sqcup \alpha \cdot \max_{s' \in \tau(s)} \min_{t' \in \tau(t)} d^k(s',t')$, for all $s,t \in S$. Then $bd_\alpha^x = \lim_{k \to \infty} d^k$, and it can be shown that this and the other computations terminate in at most $|S|^2$ iterations. This gives the following complexity result.

Theorem 12 *Computing bd_α^x for $x \in \{\mathrm{Ss}, \mathrm{Sa}, \mathrm{As}, \mathrm{Aa}\}$, $\alpha \in [0,1]$ and a QTS $\mathcal{S}$ can be done in time $O(|\mathcal{S}|^4)$.*

5 Comparing the Linear and Branching Distances

Just as similarity implies trace inclusion, we have both $ld_\alpha^{\mathrm{a}} \leq bd_\alpha^{\mathrm{Aa}}$ and $ld_\alpha^{\mathrm{s}} \leq \overline{bd}_\alpha^{\mathrm{As}}$; just as bisimilarity implies trace equivalence, we have $\overline{ld}_\alpha^{\mathrm{s}} \leq bd_\alpha^{\mathrm{Ss}}$ and $\overline{ld}_\alpha^{\mathrm{a}} \leq \overline{bd}_\alpha^{\mathrm{Sa}}$. Moreover, in the non-quantitative setting, trace inclusion (resp. trace equivalence) coincides with (bi-)similarity on deterministic systems. This result generalizes to distances over QTSs that are both deterministic and boolean, but not to distances over QTSs that are just deterministic.

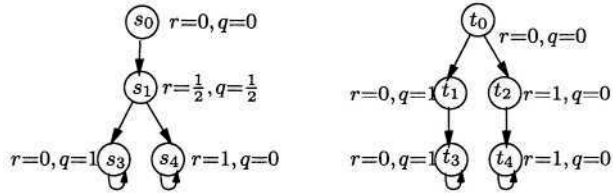
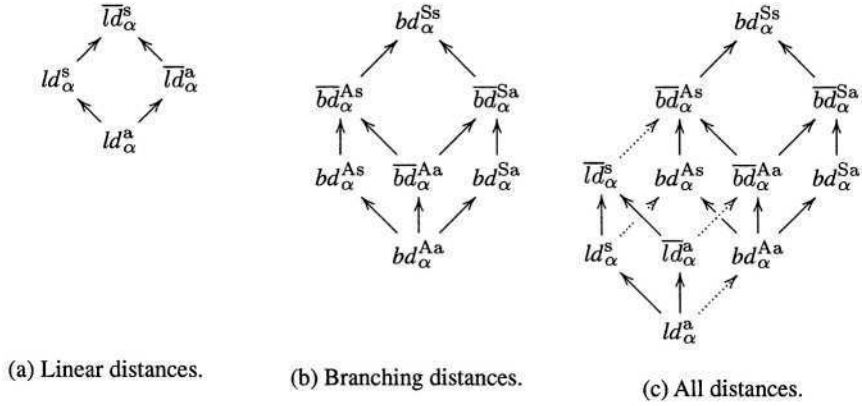


Fig. 3. Linear versus branching distances on a deterministic QTS.



(a) Linear distances. (b) Branching distances. (c) All distances.

Fig. 4. Relations between distances, where $f \to g$ means $f \leq g$. In (c), the dotted arrows collapse to equality for boolean, deterministic QTSs.

Theorem 13 *The following properties hold.*

1. *The relations in Figure 4(c) hold for all* $\alpha \in [0,1]$. *Moreover, for* $\alpha \in (0,1]$, *the inequalities cannot be replaced by equalities.*
2. *For all boolean, deterministic QTSs, all* $\alpha \in [0,1]$, *we have*

$$ld^{a}_{\alpha} = bd^{Aa}_{\alpha} \quad ld^{s}_{\alpha} = bd^{As}_{\alpha} \quad \overline{ld}^{a}_{\alpha} = \overline{bd}^{Aa}_{\alpha} \quad \overline{ld}^{s}_{\alpha} = \overline{bd}^{As}_{\alpha}.$$

These equalities need not to hold for non-boolean, deterministic QTSs.

To see that on deterministic, non-boolean QTSs, the linear distances between states can be strictly smaller than the corresponding branching ones, consider the QTS in Figure 3. We assume that $\alpha > \frac{1}{2}$; a similar example works if $\alpha \leq \frac{1}{2}$. Then $ld^{a}_{\alpha}(s,t) = ld^{s}_{\alpha}(s,t) = \overline{ld}^{a}_{\alpha}(s,t) = \overline{ld}^{s}_{\alpha}(s,t) = \frac{1}{2}\alpha$, while $bd^{Aa}_{\alpha}(s,t) = bd^{As}_{\alpha}(s,t) = \overline{bd}^{Aa}_{\alpha}(s,t) = \overline{bd}^{As}_{\alpha}(s,t) = \alpha^2$.

Acknowledgements. We thank the anonymous reviewers for their helpful comments.

References

1. P. Caspi and A. Benveniste. Toward an approximation theory for computerized control. In *Proceedings of EMSOFT,* volume 2491 of *LNCS,* pages 294–304, 2002.
2. L. de Alfaro, T. A. Henzinger, and R. Majumdar. Discounting the future in systems theory. In *Proceedings of ICALP,* volume 2719 of *LNCS,* pages 1022–1037. Springer, 2003.
3. L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. In *Proceedings STOC,* pages 675–683. ACM Press, 2001.
4. C. Derman. *Finite State Markovian Decision Processes.* Academic Press, 1970.
5. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled markov systems. In *Proceedings of CONCUR,* volume 1664 of *LNCS,* pages 258–273, 1999.
6. J. Filar and K. Vrieze. *Competitive Markov Decision Processes.* Springer-Verlag, 1997.
7. Peter Fletcher and William F. Lindgren. *Quasi-uniform spaces,* volume 77 of *Lecture Notes in Pure and Applied Mathematics.* Marcel Dekker Inc., New York, 1982.
8. M. Huth and M. Kwiatkowska. Quantitative analysis and model checking. In *Proceedings of UCS,* pages 111–122, 1997.
9. D. Kozen. A probabilistic PDL. In *Proc. 15th ACM Symp. Theory of Comp.,* pages 291–297, 1983.
10. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design: An International Journal,* 6(1):11–44, January 1995.
11. R. Majumdar. *Symbolic algorithms for verification and control.* PhD thesis, University of California, Berkeley, 2003.
12. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.
13. A. McIver and Carroll Morgan. Games, probability, and the quantitative μ-calculus qMμ. In *Proceedings of LPAR,* pages 292–310, 2002.
14. J.H. Reif. Universal games of incomplete information. In *11th Annual ACM Symposium on Theory of Computing,* pages 288–308, April, Atlanta, Georgia 1979.
15. L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time. In *Proc. 5th ACM Symp. Theory of Comp.,* pages 1–9. ACM Press, 1973.
16. F. van Breugel and J. Worrel. An algorithm for quantitative verification of probabilistic transition systems. In *Proceedings of CONCUR,* volume 2154 of *LNCS,* pages 336–350, 2001.

Learning a Hidden Subgraph

Noga Alon[1*] and Vera Asodi[2]

[1] Department of Mathematics,
Raymond and Beverly Sackler Faculty of Exact Sciences,
Tel Aviv University, Tel Aviv, Israel.
nogaa@post.tau.ac.il.
[2] Department of Computer Science,
Raymond and Beverly Sackler Faculty of Exact Sciences,
Tel Aviv University, Tel Aviv, Israel.
veraa@post.tau.ac.il.

Abstract. We consider the problem of learning a labeled graph from a given family of graphs on n vertices in a model where the only allowed operation is to query whether a set of vertices induces an edge. Questions of this type are motivated by problems in molecular biology. In the deterministic nonadaptive setting, we prove nearly matching upper and lower bounds for the minimum possible number of queries required when the family is the family of all stars of a given size or all cliques of a given size. We further describe some bounds that apply to general graphs.

1 Introduction

Let $\mathcal{H}$ be a family of labeled graphs on the set $V = \{1, 2, \ldots, n\}$, and suppose $\mathcal{H}$ is closed under isomorphism. Given a hidden copy of some $H \in \mathcal{H}$, we have to identify it by asking queries of the following form. For $F \subseteq V$, the query Q_F is: does F contain at least one edge of H? Our objective is to identify H by asking as few queries as possible. We say that a family $\mathcal{F}$ solves the $\mathcal{H}$-problem if for any two distinct members H_1 and H_2 of $\mathcal{H}$, there is at least one $F \in \mathcal{F}$ that contains an edge of one of the graphs H_i and does not contain any edge of the other. Obviously, any such family enables us to learn an unknown member of $\mathcal{H}$ deterministically and non-adaptively, by asking the questions Q_F for each $F \in \mathcal{F}$. Note that for any family $\mathcal{H}$, the set of all pairs of vertices solves the $\mathcal{H}$-problem. Note also that the information theoretic lower bound implies that we need at least $\log |\mathcal{H}|$ queries, where here and throughout the paper, all logarithms are in base 2, unless otherwise specified, and we omit all floor and ceiling signs, when these are not crucial.

There are some families of graphs for which the above problem has been studied, motivated by applications in molecular biology. These include matchings ([1]) and Hamiltonian cycles ([5,6]). The biological problem is to find, given a set

* Research supported in part by a USA-Israeli BSF grant, by the Israel Science Foundation and by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 110–121, 2004.

of molecules, pairs that react with each other. Here the vertices correspond to the molecules, the edges to the reactions, and the queries correspond to experiments of putting a set of molecules together in a test tube and determining whether a reaction occurs. The problem of finding a hidden matching is the one encountered by molecular biologists when they apply multiplex PCR in order to close the gaps left in a DNA strand after shotgun sequencing. See [1] and its references for more details.

The previous works in this field study the minimum number of queries needed to identify a hidden graph, from various families of graphs. Some of these works consider different query models than the one described above. The authors of [1] study the hidden subgraph problem for the family of matchings. In that paper it is shown that under the deterministic and non-adaptive model, the minimum number of queries that one has to ask in order to identify a hidden matching is $\Theta(n^2)$, that is, one can do better than the trivial algorithm of asking all pairs only by a constant factor. It is also proved that $\Omega(n^2)$ queries are needed in order to find a hidden copy of any bounded-degree graph with a linear size matching. The authors further present randomized non-adaptive algorithms that use $\Theta(n \log n)$ random queries, and deterministic k-round algorithms, that ask $O(n^{1+1/(2(k-1))}\text{polylog}n)$ queries. Grebinski and Kucherov [5,6] study the family of Hamiltonian cycles. A few query models are discussed in those papers. Besides the model presented above, they consider the additive model, in which the answer to a query is not just "yes" or "no" but the number of edges in the subset. Both models are considered also when the size of the queries is bounded. They present matching lower and upper bounds under each of these models, where some of the upper bounds are achieved by 2-round algorithms, and the other algorithms are fully adaptive. In [7], Grebinski and Kucherov study the problem for low degree graphs, and prove matching lower and upper bounds under the additive non-adaptive model.

In the present paper we consider only the deterministic non-adaptive model, where the answers are only "yes" or "no". The main families considered are families of stars and families of cliques. We study both families of stars or cliques of a given size, and the families of all cliques or all stars. It is shown that the trivial upper bound of $\binom{n}{2}$ is tight up to a $1 + o(1)$-multiplicative term for the families of stars of k edges, for all $n^{\frac{2}{3}} \log^{\frac{2}{3}} n << k \leq n-2$. For smaller stars, we show that less queries suffice, and give upper and lower bounds on the minimum number of queries needed. These bounds are tight up to some polylog$n$ factor for all sizes of stars, and they are of order k^3, up to the polylog$n$ factors. We show that the problem is easier when the hidden subgraph is a clique. In fact, even for the family of all cliques, the problem can be solved using $O(n \log^2 n)$ queries. We study, as in the case of stars, the problem of a hidden clique of size k, for all values of k. In all cases, we prove upper and lower bounds that are tight up to some polylog$n$ factor, and are of order k^2, up to the polylog$n$ factors. We also consider the case where the family of graphs consists of all the graphs isomorphic to a given general graph G, and give a lower bound that depends

on the maximum size of an independent set in G. From this general bound, we obtain a lower bound of $\Omega(\frac{n^2}{\log^2 n})$ for the random graph $G(n, \frac{1}{2})$.

In Section 2 we study the hidden subgraph problem where the hidden graph is a star, in Section 3 we consider the case where the hidden graph is a clique, and in Section 4 we prove a result for general graphs. Section 5 contains some concluding remarks and open problems. Due to space limitations, we omit some of the proofs. All the proofs will appear in the full version of this paper.

2 Stars

In this section we consider the case where the graphs in $\mathcal{H}$ are stars. Denote by $\mathcal{S}_k$ the family of all graphs on $V = \{1, 2, \ldots, n\}$ that consist of a copy of $K_{1,k}$ and $n - k - 1$ isolated vertices. Let $\mathcal{S} = \cup_{k=1}^{n-1} \mathcal{S}_k$. We begin with the following simple claim, whose proof will appear in the full version.

Proposition 1. *The minimum size of a family $\mathcal{F}$ that solves the $\mathcal{S}$-problem is exactly $\binom{n}{2}$.*

The proof of this proposition actually shows that even the solution of the $\mathcal{S}_{n-2} \cup \mathcal{S}_{n-1}$-problem requires $\binom{n}{2}$ queries. We now consider the case where the size of the star is known, and prove the following theorem, which gives lower and upper bounds on the minimum size of a family that solves the $\mathcal{S}_k$-problem. These bounds are tight in all cases up to some polylog$n$ factor.

Theorem 1. *For all $k \leq n - 2$ and $n > 2$, there exists a family of size*

$$min(\lceil \frac{n(n-2)}{2} \rceil, O(k^3 \log n))$$

that solves the $\mathcal{S}_k$-problem, and every family that solves the $\mathcal{S}_k$-problem either contains $(1 - o(1))\binom{n}{2}$ pairs, or it is of size at least $\Omega(\frac{k^3}{\log^2 n})$. Moreover, if $k \leq \sqrt{n}$, then the size of any family that solves the $\mathcal{S}_k$-problem is at least $\Omega(\frac{k^3 \log n}{\log k})$. For $k = n - 1$, the minimum size of such a family is exactly $\lceil \log n \rceil$.

The best bounds we get, for various values of k, are summarized in Table 1. In the rest of this section we prove these results.

The bounds for $k = n - 1$ and $k = n - 2$ are stated in the next two simple propositions. The proofs of these propositions will appear in the full version.

Proposition 2. *For all $n > 2$, the minimum size of a family $\mathcal{F}$ that solves the $\mathcal{S}_{n-1}$-problem is exactly $\lceil \log n \rceil$.*

Proposition 3. *The minimum size of a family $\mathcal{F}$ that solves the $\mathcal{S}_{n-2}$-problem is 2 for $n = 3$, it is 5 for $n = 4$, and $\lceil \frac{n(n-2)}{2} \rceil$ for all $n \geq 5$.*

Table 1. Bounds on the size of a family that solves the $\mathcal{S}_k$-problem.

k	Lower bound	Upper bound
$k \leq \sqrt{n}$	$\Omega(\frac{k^3 \log n}{\log k})$	$O(k^3 \log n)$
$\sqrt{n} < k < \frac{n^{2/3}}{(2\log n)^{1/3}}$	$\Omega(\frac{k^3}{\log^2 n})$	$O(k^3 \log n)$
$\frac{n^{2/3}}{(2\log n)^{1/3}} \leq k \leq O(n^{2/3}\log^{2/3} n)$	$\Omega(\frac{k^3}{\log^2 n})$	$\lceil \frac{n(n-2)}{2} \rceil$
$k = \omega(n^{2/3}\log^{2/3} n),\ k \leq n-3$	$(1-o(1))\binom{n}{2}$	$\lceil \frac{n(n-2)}{2} \rceil$
$k = n-2$	$\lceil \frac{n(n-2)}{2} \rceil$	$\lceil \frac{n(n-2)}{2} \rceil$
$k = n-1$	$\lceil \log n \rceil$	$\lceil \log n \rceil$

Note that the above upper bound holds for all $\mathcal{S}_k$, where $3 \leq k \leq n-2$.

We now give some general upper and lower bounds on the minimum size of a family that solves the $\mathcal{S}_k$-problem. These bounds are tight up to some polylog$n$ factor. From now on we assume, throughout the section, that n is large.

Proposition 4. *For every k, there exists a family $\mathcal{F}$ of size $O(k^3 \log n)$ that solves the $\mathcal{S}_k$-problem.*

Proof. Let $m = ck^3 \log n$ for some absolute constant c, and let $F_1, F_2, ..., F_m$ be m random subsets of V, chosen independently as follows. For every F_i, every $v \in V$ is chosen to be in F_i independently with probability $\frac{1}{k}$. Let S_1 and S_2 be two stars of size k, such that $|E(S_1) \setminus E(S_2)| = |E(S_2) \setminus E(S_1)| = 1$. Let u be the center of S_1 and S_2, let $u_1, ..., u_{k-1}$ be the other common vertices, and let v be the additional vertex of S_1, and w the additional vertex of S_2. F_i distinguishes between S_1 and S_2 if and only if $u \in F_i$, $u_j \notin F_i$ for all $1 \leq j \leq k-1$, and exactly one vertex among v and w is in F_i. Thus the probability that F_i distinguishes between S_1 and S_2 is

$$\frac{2}{k^2}\left(1-\frac{1}{k}\right)^k = \Omega\left(\frac{1}{k^2}\right).$$

Therefore, the probability that no F_i distinguishes between S_1 and S_2 is

$$\left[1-\Omega\left(\frac{1}{k^2}\right)\right]^m < n^{-(2k+2)}$$

provided c is sufficiently large. For two stars that differ in more edges, this probability is smaller. The number of pairs of stars is smaller than n^{2k+2}, and hence, there is a family $\mathcal{F} = \{F_1, F_2, ..., F_m\}$ that solves the $\mathcal{S}_k$-problem. □

We show that the upper bound given in Proposition 4 is tight up to a factor of polylog$n$. More precisely, we show that for every $k \leq n-2$, a family $\mathcal{F}$ that solves the $\mathcal{S}_k$-problem either contains $(1-o(1))\binom{n}{2}$ pairs, or it is of cardinality $\Omega(\frac{k^3}{\text{polylog}n})$.

Proposition 5. *For every $k \leq n-2$, if $\mathcal{F}$ is a family that solves the $\mathcal{S}_k$-problem, then $\mathcal{F}$ either contains $(1-o(1))\binom{n}{2}$ pairs, or it is of cardinality at least $\Omega(\frac{k^3}{\log^2 n})$.*

Proof. Let $\mathcal{F}$ be a family that solves the $\mathcal{S}_k$-problem. Then, for every $u \in V$ and $A, B \subseteq V \setminus \{u\}$ such that $|A| = 2$, $|B| = k-1$ and $A \cap B = \emptyset$, there exists a set $F \in \mathcal{F}$ such that $u \in F$, $|F \cap A| = 1$ and $F \cap B = \emptyset$. Indeed, otherwise $\mathcal{F}$ would not distinguish between the two stars whose center is u, which share the vertices of B, and where the additional vertex of one star is one vertex of A, and the additional vertex of the other one is the other vertex of A.

Denote by $\mathcal{F}_0$ the family of all sets $F \in \mathcal{F}$ of size 2. Let $m = c \cdot \frac{n \log n}{k}$ for some absolute constant c, and define $\mathcal{F}_1 = \{F \in \mathcal{F} \mid 2 < |F| \leq m\}$, and $\mathcal{F}_2 = \mathcal{F} \setminus (\mathcal{F}_0 \cup \mathcal{F}_1)$. We show that for any constant $\epsilon > 0$, if $|\mathcal{F}_0| \leq (1-\epsilon)\binom{n}{2}$ then $|\mathcal{F}_1 \cup \mathcal{F}_2| > c_1 \epsilon^3 \cdot \frac{k^3}{\log^2 n}$ for some constant c_1 that depends only on c. Suppose $|\mathcal{F}_0| \leq (1-\epsilon)\binom{n}{2}$ and $|\mathcal{F}_1 \cup \mathcal{F}_2| \leq c_1 \epsilon^3 \cdot \frac{k^3}{\log^2 n}$. For every $u \in V$, denote by V_u the set of vertices $v \in V \setminus \{u\}$ such that $\{u, v\} \notin \mathcal{F}_0$. Let $V' = \{u \in V \mid |V_u| \geq \frac{\epsilon}{2}(n-1)\}$. Since $|\mathcal{F}_0| \leq (1-\epsilon)\binom{n}{2}$, $|V'| \geq \frac{\epsilon}{2} n$. Otherwise, since the pairs of vertices that are not in $\mathcal{F}_0$ are pairs $\{u, v\}$ such that $v \in V_u$, and since $v \in V_u$ if and only if $u \in V_v$, we have

$$
\begin{aligned}
|\mathcal{F}_0| &= \binom{n}{2} - \frac{1}{2} \sum_{u \in V} |V_u| \\
&= \binom{n}{2} - \frac{1}{2} \left(\sum_{u \in V'} |V_u| + \sum_{u \in V \setminus V'} |V_u| \right) \\
&> \binom{n}{2} - \frac{1}{2} \left[|V'|(n-1) + |V \setminus V'| \frac{\epsilon}{2}(n-1) \right] \\
&> \binom{n}{2} - \frac{1}{2} \left[\frac{\epsilon}{2} n(n-1) + n \frac{\epsilon}{2}(n-1) \right] \\
&= (1-\epsilon)\binom{n}{2}.
\end{aligned}
$$

Choose uniformly a vertex $u \in V'$, and then choose uniformly a subset $A = \{v, w\} \subseteq V_u$. Define $\mathcal{F}_1' = \{F \in \mathcal{F}_1 \mid u \in F, |F \cap A| = 1\}$. For each $F \in \mathcal{F}_1$

$$
Pr(F \in \mathcal{F}_1') = \frac{|F|(|F|-1)(n-|F|)}{\frac{\epsilon}{2} n \binom{\frac{\epsilon}{2}(n-1)}{2}} \leq \frac{32}{\epsilon^3} \cdot \frac{|F|^2}{n^2}.
$$

Therefore,

$$
E\left[|\mathcal{F}_1'|\right] \leq \frac{32}{\epsilon^3} \sum_{F \in \mathcal{F}_1} \frac{|F|^2}{n^2} \leq \frac{32}{\epsilon^3} |\mathcal{F}_1| \frac{m^2}{n^2},
$$

and hence, there is a choice of u and A such that

$$
|\mathcal{F}_1'| \leq \frac{32}{\epsilon^3} |\mathcal{F}_1| \frac{m^2}{n^2} \leq 32 c_1 c^2 \cdot \frac{k^3}{\log^2 n} \cdot \frac{n^2 \log^2 n}{k^2 n^2} \leq \frac{k}{2} - 1
$$

provided c_1c^2 is sufficiently small. Thus, there exists a subset $B_1 \subseteq V \setminus (\{u\} \cup A)$ of size $\frac{k}{2} - 1$ that intersects every $F \in \mathcal{F}_1'$. Choose a random subset $B_2 \subseteq V$ of size $\frac{k}{2}$. For every $F \in \mathcal{F}_2$

$$Pr(F \cap B_2 = \emptyset) = \frac{\binom{n-|F|}{\frac{k}{2}}}{\binom{n}{\frac{k}{2}}} \leq \left(1 - \frac{|F|}{n}\right)^{\frac{k}{2}} \leq e^{-\frac{km}{2n}} = n^{-c_2}$$

for some constant $c_2 = \Theta(c)$. Therefore, if c is sufficiently large, with high probability, $u \notin B_2$, $A \cap B_2 = \emptyset$ and $\forall F \in \mathcal{F}_2 \quad F \cap B_2 \neq \emptyset$.

Denote $B' = B_1 \cup B_2$. $B' \subseteq V \setminus (\{u\} \cup A)$ and $|B'| \leq k - 1$. Let B be an arbitrary extension of B' to a subset of $V \setminus (\{u\} \cup A)$ of size $k - 1$. Consider the following two stars S_1 and S_2; u is the center of S_1 and S_2, they share the vertices of B, the additional vertex of S_1 is v, and the additional vertex of S_2 is w. Since A was chosen from V_u, the pairs $\{u, v\}$ and $\{u, w\}$ are not in $\mathcal{F}_0$, and thus no set in $\mathcal{F}_0$ can distinguish between S_1 and S_2. Neither can the sets in $\mathcal{F}_1$ that do not contain u, nor those whose intersection with A is not of size 1. All other sets in $\mathcal{F}_1$, i.e. sets $F \in \mathcal{F}_1$ such that $u \in F$, and $|F \cap A| = 1$, and all the sets in $\mathcal{F}_2$, contain a vertex of B, so they cannot distinguish between these two stars either. Thus $\mathcal{F}$ cannot distinguish between S_1 and S_2, contradicting the assumption that it solves the $\mathcal{S}_k$-problem. □

We now prove a better lower bound for $k \leq \sqrt{n}$. This bound is tight up to a factor of $\log k$. For the proof of the this bound, we need a variant of a lemma proved in [8,4].

Definition 1. *Let $\mathcal{A}$ be a family of subsets of a set S. We say that $\mathcal{A}$ is* r-cover-free *if no set in $\mathcal{A}$ is contained in the union of any r other sets in $\mathcal{A}$.*

Lemma 1. *Let S be a set of size m, and let $\mathcal{A}$ be a family of n subsets of S. Suppose $\mathcal{A}$ is r-cover-free, where $r \leq 2\sqrt{n}$. Then,*

$$m > \frac{r^2 \log(n - \frac{r}{2})}{10 \log r}.$$

In [8], it is proved that for fixed r and large $n \geq n(r)$, $\frac{\log n}{m} \leq 8 \cdot \frac{\log r}{r^2}$. By a simple modification of that proof, which will be described in the full version, we show that the lemma as stated above holds for every $r \leq 2\sqrt{n}$. We use this lemma to improve the lower bound, for $k \leq \sqrt{n}$.

Proposition 6. *For every $k \leq \sqrt{n}$, if $\mathcal{F}$ is a family the solves the $\mathcal{S}_k$-problem, then $|\mathcal{F}| = \Omega(\frac{k^3 \log n}{\log k})$.*

Proof. Let $\mathcal{F}$ be a family that solves the $\mathcal{S}_k$-problem. Choose, randomly, $A, B \subseteq V$, such that $|A| = 2$, $|B| = \frac{k}{2} - 1$, and $A \cap B = \emptyset$. Define $\mathcal{G} = \{F \in \mathcal{F} \mid |F \cap A| = 1, F \cap B = \emptyset\}$. Clearly,

$$Pr(F \in \mathcal{G}) = \frac{|F|(n-|F|)}{\binom{n}{2}} \cdot \frac{\binom{n-|F|-1}{\frac{k}{2}-1}}{\binom{n-2}{\frac{k}{2}-1}}$$
$$= \frac{2|F|}{n} \cdot \frac{\binom{n-|F|}{\frac{k}{2}}}{\binom{n-1}{\frac{k}{2}}}$$
$$\leq \frac{2|F|}{n}(1 - \frac{|F|-1}{n-1})^{\frac{k}{2}}$$
$$\leq \frac{2|F|}{n} e^{-\frac{k|F|}{4n}}.$$

If $|F| \leq \frac{4n}{k}$ then $Pr(F \in \mathcal{G}) \leq \frac{8}{k}$. If $|F| > \frac{4n}{k}$, denote $x = \frac{k|F|}{4n}$. Since $x > 1$ we have $Pr(F \in \mathcal{G}) \leq \frac{8}{k} x e^{-x} < \frac{8}{ek}$. Hence, for all F, $Pr(F \in \mathcal{G}) \leq \frac{c}{k}$ for some constant c, and thus the expected size of $\mathcal{G}$ is $\frac{c|\mathcal{F}|}{k}$. Therefore, there exists a choice of A and B for which $|\mathcal{G}| \leq \frac{c|\mathcal{F}|}{k}$.

Denote $V' = V \setminus (A \cup B)$, and consider the family $\mathcal{G}' = \{F \cap V' \mid F \in \mathcal{G}\}$. Since $\mathcal{F}$ solves the $\mathcal{S}_k$-problem, for all $u \in V'$, and every $C \subseteq V' \setminus \{u\}$ of size $\frac{k}{2}$, there is a set $F \in \mathcal{G}'$ such that $u \in F$ and $F \cap C = \emptyset$. Otherwise, $\mathcal{F}$ would not distinguish between the two stars whose center is u, that share the $k-1$ vertices of $B \cup C$, and for which the additional vertex of one of them is one element of A, and the additional vertex of the other one is the other element of A. Let $m = |\mathcal{G}'|$, $n' = |V'| = n - \frac{k}{2} - 1$, and let M be the m by n' matrix whose rows are the incidence vectors of the sets in $\mathcal{G}'$. Now let us look at the columns of M as the incidence vectors of subsets of another set, of size m. For every column i, and every set J of $\frac{k}{2}$ columns such that $i \notin J$, there exists a row in which the i^{th} coordinate is 1, and for all $j \in J$, the j^{th} coordinate is 0. Thus, no subset corresponding to a column is contained in the union of $\frac{k}{2}$ subsets correspond to any other $\frac{k}{2}$ columns, and by Lemma 1,

$$|\mathcal{G}'| = m > \frac{(\frac{k}{2})^2 \log(n' - \frac{k}{4})}{10 \log \frac{k}{2}} = \Omega\left(\frac{k^2 \log n}{\log k}\right),$$

and hence

$$|\mathcal{F}| \geq \Omega(k|\mathcal{G}|) \geq \Omega(k|\mathcal{G}'|) \geq \Omega\left(\frac{k^3 \log n}{\log k}\right).$$

□

3 Complete Graphs

In this section we consider the case where the hidden graphs are complete graphs. Denote by $\mathcal{C}_k$ the family of all graphs on $V = \{1, 2, \ldots, n\}$, that consist of a copy of K_k, and $n-k$ isolated vertices. Let $\mathcal{C} = \cup_{k=2}^{n} \mathcal{C}_k$.

In the following theorem, we prove lower and upper bounds on the minimum size of a family that solves the $\mathcal{C}$-problem. The proof will appear in the full version.

Table 2. Bounds on the size of a family that solves the C_k-problem.

k	Lower bound	Upper bound
$k \leq n^{\frac{1}{3}}$	$\Omega(\frac{k^2 \log n}{\log k})$	$O(k^2 \log n)$
$n^{\frac{1}{3}} < k \leq \sqrt{n}$	$\Omega(k^2)$	$O(k^2 \log n)$
$\sqrt{n} < k < \sqrt{n \log n}$	$\Omega(\frac{k^2}{\log n})$	$O(k^2 \log n)$
$\sqrt{n \log n} \leq k \leq n - \log^2 n$	$\Omega(n)$	$O(n \log^2 n)$
$k = n - s,\ s < \log^2 n$	$\Omega(n)$	$(s+1)\lceil \frac{n}{2} \rceil$

Theorem 2. *Any family that solves the C-problem is of size at least $\Omega(n \log n)$, and there exists a family of size $O(n \log^2 n)$ that solves the C-problem.*

We now give upper and lower bounds for cliques of a given size. These results are tight up to a factor of polylog$n$ for all admissible sizes.

Theorem 3. *For every k, there exists a family $\mathcal{F}$ of size $O(k^2 \log n)$ that solves the C_k-problem, and every family that solves the C_k-problem either contains $\Omega(n)$ pairs, or it is of size at least $\Omega(\frac{k^2}{\log n})$. Moreover, for all $k \leq n^{\frac{1}{3}}$, the size of any family that solves the C_k-problem is at least $\Omega(\frac{k^2 \log n}{\log k})$, and for all $k \leq \sqrt{n}$ it is at least $\Omega(k^2)$. In addition, for all s, there exists a family of size $(s+1)\lceil \frac{n}{2} \rceil$ that solves the C_{n-s}-problem.*

The best bounds we have, for various values of k, are summarized in Table 2. In the rest of this section we prove these results.

Proposition 7. *For every k, there exists a family $\mathcal{F}$ of size $O(k^2 \log n)$ that solves the C_k-problem.*

The proof of Proposition 7 is similar to the proof of Proposition 4. The details will appear in the full version.

Proposition 8. *For every k, if $\mathcal{F}$ is a family the solves the S_k-problem, then $\mathcal{F}$ either contains $\Omega(n)$ pairs, or it is of cardinality at least $\Omega(\frac{k^2}{\log n})$.*

The proof of Proposition 8 is similar to the proof of Proposition 5. The details will appear in the full version.

We now prove a better lower bound for $k \leq n^{\frac{1}{3}}$. This bound is tight up to a factor of $\log k$. In order to prove this bound, we need the following lemma, whose proof will appear in the full version.

Lemma 2. *Let S be a set of size m, and let $\mathcal{A}$ be a family of n subsets of S. Suppose that there are no distinct $A, B_1, \ldots, B_r, C_1, \ldots, C_r \in \mathcal{A}$ for which*

$$A \subseteq \bigcup_{i=1}^{r} B_i$$

and

$$A \subseteq \bigcup_{i=1}^{r} C_i,$$

where $r \leq n^{\frac{1}{3}}$. *Then* $m = \Omega(\frac{r^2 \log n}{\log r})$.

Proposition 9. *For every* $k \leq n^{\frac{1}{3}}$, *if* $\mathcal{F}$ *is a family that solves the* $\mathcal{C}_k$*-problem, then* $|\mathcal{F}| = \Omega(\frac{k^2 \log n}{\log k})$.

Proof. Let $\mathcal{F}$ be a family that solves the $\mathcal{C}_k$-problem. Define $m = |\mathcal{F}|$, and let M be the m by n matrix whose rows are the incidence vectors of the sets in $\mathcal{F}$. Consider the columns of M as the incidence vectors of subsets of another set, of size m. For $1 \leq i \leq n$, let G_i be the subset corresponding to the i^{th} column of M. Define the family $\mathcal{G}$ as follows. $\mathcal{G} = \{G_{2i-1} \cup G_{2i} \mid 1 \leq i \leq \frac{n}{2}\}$. We claim that there are no distinct sets $A, B_1, ..., B_{\frac{k-1}{4}}, C_1, ..., C_{\frac{k-1}{4}} \in \mathcal{G}$, such that

$$A \subseteq \bigcup_{i=1}^{\frac{k-1}{4}} B_i \tag{1}$$

and

$$A \subseteq \bigcup_{i=1}^{\frac{k-1}{4}} C_i. \tag{2}$$

Suppose there were such sets. A is the union of two subsets corresponding to two distinct columns of M. Let u and v be the vertices corresponding to these columns. Similarly, let $w_1, ..., w_{k-1}$ be the vertices corresponding to $B_1, ..., B_{\frac{k-1}{4}}$, $C_1, ..., C_{\frac{k-1}{4}}$. The members of A are the queries that contain u or v. Since (1) and (2) hold, each such query contains at least two vertices from $w_1, ..., w_{k-1}$. Thus, no query distinguishes between the complete graph on $u, w_1, ..., w_{k-1}$ and the complete graph on $v, w_1, ..., w_{k-1}$. Hence, there are no such sets in $\mathcal{G}$, and therefore, by Lemma 2, with $r = \frac{k-1}{4}$ and $\mathcal{A} = \mathcal{G}$,

$$|\mathcal{F}| = m = \Omega\left(\frac{k^2 \log n}{\log k}\right).$$

□

We now prove that for all $n^{\Omega(1)} \leq k \leq \sqrt{n}$, any family that solves the $\mathcal{C}_k$-problem is of size at least $\Omega(k^2)$.

Definition 2. *Let* A *be a subset of a set* S, *and let* $\mathcal{A}$ *be a family of subsets of* S. *We say that* A *is* covered twice *by* $\mathcal{A}$ *if for all* $a \in A$, *there are at least two sets in* $\mathcal{A}$ *that contain* a.

The proof of the following lemma will appear in the full version.

Lemma 3. *Let S be a set of size m, and let $\mathcal{A}$ be a family of n subsets of S. Suppose that no set in $\mathcal{A}$ is covered twice by any other r sets in $\mathcal{A}$, where $n^{\Omega(1)} \leq r \leq \sqrt{n}$. Then $m = \Omega(r^2)$.*

Proposition 10. *For every $n^{\Omega(1)} \leq k \leq \sqrt{n}$, if $\mathcal{F}$ is a family that solves the C_k-problem, then $|\mathcal{F}| = \Omega(k^2)$.*

This proposition is proved similarly to Proposition 9, using Lemma 3 instead of Lemma 2. The proof will appear in the full version.

We conclude the section with a simple upper bound, which improves our estimate for cliques that contain almost all the vertices.

Proposition 11. *For every s, there exists a family of size at most*

$$(s+1)\left\lceil \frac{n}{2} \right\rceil$$

that solves the C_{n-s}-problem.

Proof. For each $u \in V$, ask $s+1$ pairs that contain u. u is in the clique if and only if the answer to at least one of these queries is "yes". □

4 General Graphs

In this section we consider families that contain all the graphs on V isomorphic to a graph G. Denote by $\mathcal{H}_G$ the family of all graphs isomorphic to G.

Theorem 4. *Let $G = (V, E)$ be a graph on n vertices, and suppose that there are three vertices $u, v, w \in V$, such that for every two of them, the sets of their neighbours except these vertices themselves are distinct, i.e. $N(u) \setminus \{v\} \neq N(v) \setminus \{u\}$, $N(u) \setminus \{w\} \neq N(w) \setminus \{u\}$, and $N(v) \setminus \{w\} \neq N(w) \setminus \{v\}$. Then, the size of any family that solves the $\mathcal{H}_G$-problem is at least $\Omega(\frac{n^2}{\alpha^2(G)})$, where $\alpha(G)$ is the maximum size of an independent set in G.*

Proof. For any two vertices $x, y \in V$, denote by $A(x, y)$ the set of vertices $z \in V \setminus \{x, y\}$ such that z is a neighbour of both x and y, or of none of them. We show that there are two vertices among u, v, and w, for which the size of this set is at least $\frac{1}{3}n - 1$. Suppose that $A(u, v) < \frac{1}{3}n - 1$. Then, $V \setminus (A(u, v) \cup \{u, v, w\}) > \frac{2}{3}n - 2$, and each one of these vertices is a neighbour of exactly one vertex among u and v. Thus, each one of these vertices is in $A(u, w)$ or in $A(v, w)$, and hence at least one of these sets is of size at least $\frac{1}{3}n - 1$. Assume, without loss of generality, that $|A(u, v)| \geq \frac{1}{3}n - 1$.

Let $\mathcal{F}$ be a family that solves the $\mathcal{H}_G$-problem, and let $\alpha = \alpha(G)$. Assume that $|\mathcal{F}| < \frac{n^2}{12\alpha^2}$. Every set $F \in \mathcal{F}$ is of size at most α, or otherwise the answer to Q_F is "yes" (and is known in advance). For every $x \in V$, denote by $f(x)$ the number of sets $F \in \mathcal{F}$ such that $x \in F$.

$$\sum_{x \in V} f(x) = \sum_{F \in \mathcal{F}} |F| \leq \alpha |\mathcal{F}| < \frac{n^2}{12\alpha}. \tag{3}$$

Let $V' = \{x \in V \mid f(x) < \frac{n}{6\alpha}\}$. Then $|V'| \geq \frac{n}{2}$, since otherwise

$$\sum_{x \in V} f(x) \geq \sum_{x \in (V \setminus V')} f(x) \geq \frac{n}{2} \cdot \frac{n}{6\alpha} = \frac{n^2}{12\alpha},$$

contradicting (3). For $x \in V'$, the number of vertices $z \in V$ such that there exists a set $F \in \mathcal{F}$ that contains both x and z is at most

$$\sum_{F: x \in F} |F| \leq f(x)\alpha < \frac{n}{6}.$$

Let $x, y \in V'$, and let A be the set of all vertices $z \in V$ such that there exists a set $F \in \mathcal{F}$ that contains x or y, and z.

$$|A| \leq \sum_{F: x \in F} |F| + \sum_{F: y \in F} |F| < \frac{n}{3}.$$

Let G_1 be a graph isomorphic to G, where u is mapped to x, v is mapped to y, and only vertices from $A(u,v)$ are mapped into A. Let G_2 be the graph in which u is mapped to y, v is mapped to x, and the rest of it is identical to G_1. The only queries that could distinguish between G_1 and G_2 are queries Q_F where F contains x or y, but then all the other vertices in F are in $A(u,v)$, and thus, the answer to Q_F is the same for G_1 and G_2. Therefore, $\mathcal{F}$ cannot distinguish between G_1 and G_2, contradicting the assumption that it solves the $\mathcal{H}_G$-problem. □

Corollary 1. *Let $G = G(n, \frac{1}{2})$ be the random graph on n vertices. Then, almost surely, any family that solves the $\mathcal{H}_G$-problem is of size at least $\Omega(\frac{n^2}{\log^2 n})$.*

Proof. The corollary follows from Theorem 4, since, almost surely, $\alpha(G) = O(\log n)$ (see, for example, [3] or [2]), and since obviously, there are, almost surely, three vertices u, v and w with distinct sets of neighbours, as defined in the theorem. □

5 Concluding Remarks

In this paper we have studied the hidden subgraph problem for families of stars and cliques. We have shown upper and lower bounds on the minimum number of required queries under the deterministic non-adaptive model. Those bounds are tight up to polylogarithmic factors. It would be interesting to close these gaps between the upper and the lower bounds.

We have also presented a lower bound for general graphs based on the size of the maximum independent set. This bound is almost tight for the random graph $G(n, \frac{1}{2})$. However, for graphs with large independent sets this bound might be far from the actual number of required queries. It would be interesting to find

better estimations for general graphs. In particular, the problem of characterizing all graphs for which the trivial upper bound of $O(n^2)$ is best possible seems interesting. Our results enable us to prove an $\Omega(n^2)$ lower bound for the number of queries required to identify a hidden copy of any graph with at least one isolated vertex, containing a vertex of degree 1 which is adjacent to a vertex of high degree. We omit the details.

In this work we have focused on non-adaptive algorithms. However, the number of queries can be reduced if we allow more than one round. For example, our upper bound of $O(k^3 \log n)$ for the family of stars of size k can be reduced to $O(k^2 \log n)$ when two rounds are permitted. This can be done by identifying the center of the star in the first round, and then finding the leaves in the second round. The first round can be carried out using $O(k^2 \log n)$ queries. The proof is similar to the one presented for the non-adaptive upper bound (Proposition 4), but here we only have to distinguish between pairs of stars with distinct centers. Once the center is known, finding the leaves is a simple group testing problem that can be solved in one round using $O(k^2 \log n)$ queries (see, e.g., [4, 8] and their references). It would be interesting to study the number of queries when more rounds are allowed or when the algorithms are fully adaptive, for the family of stars as well as for other families of graphs.

Another variation of this problem that originates from the biological problem we are motivated by is introducing erroneous results for some queries. It would be interesting to study how different error models affect the number of required queries.

References

1. N. Alon, R. Beigel, S. Kasif, S. Rudich, B. Sudakov, Learning a Hidden Matching, Proceedings of the 43rd IEEE FOCS, 2002, 197-206.
2. N. Alon and J. H. Spencer, **The Probabilistic Method,** Second Edition, Wiley, New York, 2000.
3. B. Bollobás, **Random Graphs,** Academic Press, 1985.
4. A. G. Dyachkov and V. V. Rykov, Bounds on the Length of Disjunctive Codes, *Problemy Peredachi Informatsii* Vol. 18, no. 3 (1982), 158-166.
5. V. Grebinski and G. Kucherov, Optimal Query Bounds for Reconstructing a Hamiltonian Cycle in Complete Graphs, Proc. 5th Israeli Symposium on Theoretical Computer Science (1997), 166-173.
6. V. Grebinski and G. Kucherov, Reconstructing a Hamiltonian Cycle by Querying the Graph: Application to DNA Physical Mapping, *Discrete Applied Math.* 88 (1998), 147-165.
7. V. Grebinski and G. Kucherov, Optimal Reconstruction of Graphs under the Additive Model, *Algorithmica* 28(1) (2000), 104-124.
8. M. Ruszinkó, On the Upper Bound of the size of the r-cover-free families, *Journal of Combinatorial Theory* Series A vol. 66, no. 2, May 1994, 302-310.

Optimal Reachability for Weighted Timed Games*

Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan

University of Pennsylvania

Abstract. Weighted timed automata are timed automata annotated with costs on locations and transitions. The optimal game-reachability problem for these automata is to find the best-cost strategy of supplying the inputs so as to ensure reachability of a target set within a specified number of iterations. The only known complexity bound for this problem is a doubly-exponential upper bound. We establish a singly-exponential upper bound and show that there exist automata with exponentially many states in a single region with pair-wise distinct optimal strategies.

1 Introduction

Timed automata [2] extend finite-state automata with real-valued clock variables, and have proved to be useful in modeling real-time systems. The canonical problem for timed automata is reachability, and can be solved in polynomial-space using a finite-state quotient—the so-called *region graph*—of the underlying infinite state-space. A natural generalization of reachability corresponds to *optimed reachability* that asks how soon can a target set be reached from an initial state. This problem, and its variations, are theoretically interesting as decidability and finiteness of representation are not immediate from the region graph construction, and have been studied by many researchers (cf. [7,1,3,11]). In particular, in a *weighted timed automaton* (also known as a *priced timed automaton*), each discrete transition has an associated nonnegative integer denoting the cost to be paid when the transition is taken, and each location has an associated nonnegative integer denoting the cost rate with respect to the time spent at that location. The minimum-cost reachability problem for weighted timed automata can be solved in exponential-time [3]. An alternative branch-and-bound solution is implemented in the verification tool UPPAAL with applications to scheduling problems [11,5].

In this paper, we consider games on weighted timed automata. Games are useful for synthesizing controllers, and for generating schedules in the context of real-time systems. In one round of our game, the controller chooses an input symbol a, and a time $t \geq 0$ at which it wants to supply the input. The adversary updates the state of the automaton either by executing an uncontrolled discrete transition at time $t' \leq t$, or by executing an a-labeled discrete transition at time

* This research was partially supported by ARO URI award DAAD19-01-1-0473, and NSF awards ITR/SY 0121431 and CCR 0306382.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 122–133, 2004.

t. Given a set of target locations, an initial state s, a cost bound C, and a bound k on the number of rounds, the *optimal game-reachability problem* is to determine if the controller has a strategy to enforce the game started in state s into a target location within k rounds while ensuring that the cost of the run is bounded by C. In absence of costs and optimality, there is a long history of research on games for timed automata, and such games are known to be decidable (cf. [15,12,8,6]). Time-optimal games, that is, games in which the cost of a run equals the total time spent, are considered in [4], and are shown to be decidable (however, no complexity bounds, upper or lower, are reported and the solution technique does not generalize to weighted timed games). The general case for (bounded games) on weighted timed automata is considered in [14], and the authors show that the problem can be encoded using first-order theory of reals with addition [9], leading to a doubly-exponential solution (note that the first-order theory over reals with addition is not decidable in nondeterministic exponential time [10]).

In this paper, we provide an exponential-time solution to the optimal game-reachability problem. We show how to compute a sequence of functions opt_i such that for each i, for each state s, $opt_i(s)$ is the optimal cost of reaching a target location starting from s in i steps in the timed game, and the representation of opt_i is exponential in i and in the size of the automaton.

It is easy to show that each region can be split into finitely many subregions (or cells) such that the optimal cost function opt_i is linear within each cell. The main technical challenge in this paper is getting a tight bound on the required splitting into cells. While computing the function opt_i from opt_{i-1}, one source of complexity is the discrete *min-max* nature of the game. If f_1 and f_2 are functions with n pieces, then the *min* or *max* operation could result in a function which has $O(n^d)$ splits (where d is the number of clocks). However, this analysis only gives a doubly exponential bound on the growth of the number of cells. We show that the partitioning of a region into cells can be organized as a tree, where each node has an associated cell, a set of hyperplanes, and a child for every subcell formed by these hyperplanes. In this representation, *min-max* operation adds just one level to the tree, and the number of hyperplanes at a node of the tree for opt_i grows linearly. The second source of complexity is the continuous *inf-sup* nature of the game: the controller picks a time t and the adversary picks a time $t' \leq t$. In a timed automaton, all clocks increase uniformly, and hence, if we restrict attention to a diagonal tube where the same set of cells are relevant, the interesting choices of t and t' are at the cell boundaries. Our final analysis combines the tube-based and tree-based representations, and shows that each f_i can be represented using at most exponentially many cells.

We also show that the bound on splitting a region into cells is tight: for every n, there exists a weighted timed automaton A_n, a region R of A_n, and exponentially many states within R, such that the discrete components of the optimal cost strategies are all different for games starting at these states, and thus, the region R must be split into exponentially many cells.

2 Model

2.1 Weighted Timed Automata

Let X be a finite set of clocks. Let $\mathbb{R}_+$ denote the set of all non-negative reals and let $\mathbb{N}$ denote the set of all non-negative integers. A clock valuation is a map $\nu : X \to \mathbb{R}_+$. The set of *constraints* over X, denoted $G(X)$, is the set of boolean combinations of constraints of the form $x \sim \beta$ or $x - y \sim \beta$ where $x, y \in X$, $\beta \in \mathbb{N}$, and $\sim \in \{<, >, =, \leq, \geq\}$. The notion of when a clock valuation ν over X satisfies a constraint over X is the natural one. Let $\bar{0}$ denote the valuation that maps each clock in X to 0.

Definition 1. *A weighted timed automaton (WTA) is a tuple* $A = (Q, Q_F, X, \Sigma, u, \delta, Inv, W_Q, W_\delta)$ *where Q is a finite set of locations, $Q_F \subseteq Q$ is a set of target locations, X is a finite set of clocks, Σ is a finite set of actions that contains the special symbol u, $\delta \subseteq Q \times \Sigma \times G(X) \times 2^X \times Q$ is the transition relation, $Inv : Q \to G(X)$ is an* invariant function, *$W_Q : Q \to \mathbb{N}$ gives the cost for each location and $W_\delta : \delta \to \mathbb{N}$ gives the cost for each transition.*

For a transition $e = (q, a, g, Z, q') \in \delta$, the label of e is a, and is denoted by $Action(e)$. Transitions labeled u model uncontrolled transitions.

A *state* of A is a pair $s = (q, \nu)$ where $q \in Q$ and ν is a clock valuation over the set of clocks X. Let *States* denote the set of all states. For a clock valuation ν and $t \in \mathbb{R}_+$, let $\nu + t$ denote the clock valuation ν' where $\nu'(x) = \nu(x) + t$, for each $x \in X$. Also, for any clock valuation ν and a set of clocks $Z \subseteq X$, let $\nu/reset(Z)$ denote the valuation ν' where $\nu'(z) = 0$ for each $z \in Z$ and for each $x \notin Z$, $\nu'(x) = \nu(x)$.

We now define timed transitions and discrete transitions between states. A timed transition is of the form $(q, \nu) \xrightarrow{t} (q, \nu + t)$, where $(q, \nu), (q, \nu + t) \in States$ and $t \in \mathbb{R}_+$, such that for every $0 \leq t' \leq t$, $\nu + t'$ satisfies $Inv(q)$, the invariant at q.

A discrete transition is of the form $(q, \nu) \xrightarrow{e} (q', \nu')$, where e is a transition of the form $(q, a, g, Z, q') \in \delta$ such that ν satisfies g, $\nu' = \nu/reset(Z)$ and ν' satisfies $Inv(q')$. We say e is enabled at a state (q, ν) if there is a transition of the form $(q, \nu) \xrightarrow{e} (q', \nu')$. We say an action $a \in \Sigma$ is enabled at (q, ν) if some a-labeled transition is enabled at (q, ν).

A run of length k of a WTA A from a state s_1 is a sequence of $2k$ alternating timed and discrete transitions $\rho = s_1 \xrightarrow{t_1} s_1' \xrightarrow{e_1} s_2 \xrightarrow{t_2} s_2' \ldots s_k' \xrightarrow{e_k} s_{k+1}$. For such a run, we define the cost of ρ, denoted $W(\rho)$ to be the cost incurred along this run, i.e. if $s_i = (q_i, \nu_i)$, for each i, then $W(\rho) = (\sum_{i=1}^{k} W_Q(q_i) \cdot t_i) + (\sum_{i=1}^{k} W_\delta(e_i))$.

Let $\Sigma' = \Sigma \setminus \{u\}$. The game is played by two players—the *controller* and the *adversary*. At any state, the controller first picks a time t and an action $a \in \Sigma'$ to signal that it would like to make an a-labeled transition after time t and not any transition before that time. This choice must be valid in the sense that it must be possible to wait for time t without violating the invariant and after time t, some a-labeled transition must be available. The adversary now has two

choices: it can wait for some time $0 \leq t' \leq t$ and execute a transition labeled u or it can decide to wait for time t and choose to take some a-labeled transition. The game then evolves to a new state and the players proceed to play as before.

Formally, a *(controller) strategy* is a function $str: States \to \mathbb{R}_+ \times \Sigma'$. A run $\rho = s_1 \xrightarrow{t'_1} s'_1 \xrightarrow{e_1} s_2 \ldots \xrightarrow{t'_k} s'_k \xrightarrow{e_k} s_{k+1}$ of A is said to be a play according to a controller strategy *str* if for every i, if $str(s_i) = (t_i, a_i)$, then, either $t'_i = t_i$ and $Action(e_i) = a$, or, $t'_i \leq t_i$ and $Action(e_i) = u$.

Let $\rho = s_1 \xrightarrow{t'_1} s'_1 \xrightarrow{e_1} s_2 \ldots \xrightarrow{t'_k} s'_k \xrightarrow{e_k} s_{k+1}$ be a run of length k. We say that ρ *wins within i steps* if there is some $i' \leq i$ such that $s_{i'} = (q, \nu)$ where $q \in Q_F$.

A controller strategy *str* is said to be winning from a state s_1 in k-steps and within cost *Cost* if for every play $\rho = s_1 \xrightarrow{t'_1} s'_1 \xrightarrow{e_1} s_2 \ldots \xrightarrow{t'_k} s'_k \xrightarrow{e_k} s_{k+1}$ of length k according to *str,* there is an $i \leq k$ such that:

- ρ wins within i steps and the cost of the prefix run $\rho_i = s_1 \xrightarrow{t'_1} s'_1 \xrightarrow{e_1} s_2 \ldots \xrightarrow{t'_i} s'_i \xrightarrow{e_i} s_{i+1}$ is less than or equal to *Cost,* i.e. $W(\rho_i) \leq Cost$.
- For every $j \leq i$, if $s_i = (q_i, \nu_i)$ and $str(s_i) = (t, a)$, then $(q_i, \nu_i) \xrightarrow{t} (q_i, \nu_i + t)$ is a timed transition and a is enabled at $(q_i, \nu_i + t)$.

The first condition above formalizes the requirement that the controller must force the play to Q_F within k steps and while doing so incur cost less than *Cost,* and the second formalizes the condition that while playing the game, the controller must pick valid times and actions.
We can now state the main problem we consider:

Optimal Bounded Weighted Timed Game Problem:

Given a weighted timed automaton A, an initial state q_{in} and a number k, find the optimal cost *Cost* such that there is a controller strategy that wins from the state $(q_{in}, \bar{0})$ in k steps and within cost *Cost.*

The solution we give in fact solves the more general *uniform* timed game problem, where we find a function $f_k : States \to \mathbb{R}_+ \cup \{\infty\}$ such that $f_k(s)$ is the least cost such that there is a controller strategy that wins from s in k steps and within cost $f_k(s)$ ($f_k(s)$ is ∞ if there is no strategy that wins in k steps).

3 Optimal Cost Functions

Regions. Let us fix a WTA $A = (Q, Q_F, X, \Sigma, u, \delta, Inv, W_Q, W_\delta)$ for the rest of this subsection and let β_{max} be the largest constant mentioned in the constraints in A. Fix an order on the clocks, say $\langle x_1, \ldots, x_d \rangle$. Then clock valuations naturally correspond to points in $\mathbb{R}^d_+$; we use this correspondence freely.

The notion of a *clock region* is standard in the theory of timed automata and is used to partition the clock space into partitions with respect to a timed bisimilar relation. Due to lack of space, we assume this notion and the notion of a *timed successor* of a region (see [2]).

Let us denote the set of regions as $\mathcal{R}$; note that the size of $\mathcal{R}$ is exponential in the number of clocks and the length of constants. Also, there are at most

$O(d \cdot \beta_{max})$ successive timed-successors to any region, where d is the number of clocks, i.e. if $R_0, R_1, \ldots R_l$ are such that each R_{i+1} is a successor of R_i, then $l = O(d \cdot \beta_{max})$.

A pair (q, R), where q is a location and R is a clock region, is called a *region* of A. We say a state (q, ν) belongs to a region (q', R) if $q = q'$ and ν belongs to R. The regions hence partition the set of all states of an automaton A.

Let $Enabled_\delta(q, R)$ denote the set of all transitions enabled at some (and hence all) states (q, ν) in (q, R). Let $Enabled_\Sigma(q, R)$ denote the set of actions enabled at some (and hence all) states (q, ν) in (q, R). If $e \in Enabled_\delta(q, R)$, let $succ((q, R), e)$ denote the region reached when the discrete transition e is taken from any state in (q, R) (this notion is well-defined).

We say a region R is *thin* if letting any time elapse from any point in the region leads to a clock valuation outside R, i.e. if all points in R satisfy a constraint of the form $x = \beta$ for some $x \in X$ and $\beta \in \mathbb{N}$. Note that thin regions always have timed-successors. If R is not thin but has a timed-successor region R', then for every clock valuation ν in R, the minimum time required such that $\nu + t$ is in R' is $\beta - \nu(x)$ where x is the clock that has the maximum fractional value in R (i.e. x is such that for every y, $(x - \lfloor x \rfloor) \leq (y - \lfloor y \rfloor)$ holds in R) and β is the smallest constant such that for every point ν' in R, $\nu'(x) < \beta$. We then call $\beta - x$ as the *critical clock expression* for the region R and denote it as *cce(R)*. If R is not thin and does not have a timed successor region (i.e. if it is a maximal region), we define the critical clock expression of R to be ∞.

Expressions for the Optimal Cost: We now wish to define a set of functions $opt_i^{(q,R)} : R \to \mathbb{R}_+ \cup \{\infty\}$ that is supposed to capture the optimal cost for the controller to win a game from any state in (q, R) in i steps. That is, we want that for any $\nu \in R$, $opt_i^{(q,R)}(\nu)$ is the minimum cost *Cost* such that the controller has a winning strategy that wins the game in i steps and within cost *Cost* from the state (q, ν). However, this will not be precisely true as such a *Cost* may not exist, in which case we take the infimum of all possible costs within which the controller can win (see Lemma 1 below).

The following is an inductive definition of $opt_i^{(q,R)}$, by induction on i. Further, for any fixed i, we define $opt_i^{(q,R)}$ inductively with respect to the partial order imposed by the timed-successor relation. That is, when defining $opt_i^{(q,R)}$, we assume the functions $opt_i^{(q,R')}$ have been defined, where R' is a transitive timed-successor of R.

- For every location $q \in Q_F$, $R \in \mathcal{R}$, $opt_0^{(q,R)} = 0$ and for every location $q \notin Q_F$, $R \in \mathcal{R}$, $opt_0^{(q,R)} = \infty$.
- Let $i \geq 1$ and let (q, R) be a region.
  If (q, R) is a thin region, then let R' be the timed-successor of R; otherwise, let $R' = R$. Note that R' is not thin. Let $T = cce(R')$ be the critical clock expression of R'.
  If R' has a timed-successor, let it be R''.

Let $A = \mathit{Enabled}_{\Sigma}(q,R) \cap \Sigma'$ be the controller actions enabled at (q,R) and for any $a \in \Sigma$, let $Ev(a) = \{e \mid e \in \mathit{Enabled}_{\delta}(q,R), \mathit{Action}(e) = a\}$ be the a-labeled transitions enabled at (q,R). Similarly, let $A' = \mathit{Enabled}_{\Sigma}(q,R') \cap \Sigma'$, and $Ev'(a) = \{e \mid e \in \mathit{Enabled}_{\delta}(q,R'), \mathit{Action}(e) = a\}$. Then

$$\mathit{opt}_i^{(q,R)}(\nu) = \begin{cases} \min\{\mathit{opt}_{i-1}^{(q,R)}(\nu), h_1(\nu), h_2(\nu), h_3(\nu)\} & \text{(if } R' \text{ has a timed succ)} \\ \min\{\mathit{opt}_{i-1}^{(q,R)}(\nu), h_1(\nu), h_2(\nu)\} & \text{(otherwise)} \end{cases}$$

where

$$h_1(\nu) = \min_{a \in A} \max_{e \in Ev(a) \cup Ev(u)} \{\mathit{opt}_{i-1}^{succ((q,R),e)}(\nu/\mathit{reset}(e)) + W_{\delta}(e)\}$$

$$h_2(\nu) = \inf_{0<t<T} \min_{a \in A'} \max\{g_1(\nu,t),\ g_2(\nu,a,t)\}$$

$$h_3(\nu) = \max\{g_1(\nu,T),\ W_Q(q) \cdot T + \mathit{opt}_i^{(q,R'')}(\nu + T)\}$$

$$g_1(\nu,t'') = \sup_{0<t' \le t''} \max_{e \in Ev'(u)} \{\mathit{opt}_{i-1}^{succ((q,R'),e)}((\nu+t')/\mathit{reset}(e)) + W_Q(q) \cdot t' + W_{\delta}(e)\}$$

$$g_2(\nu,a,t'') = \max_{e \in Ev'(a)} \{\mathit{opt}_{i-1}^{succ((q,R'),e)}((\nu + t'')/\mathit{reset}(e)) + W_Q(q) \cdot t'' + W_{\delta}(e)\}$$

The following is not hard to prove:

Lemma 1. *Let A be a weighted timed automaton, $k \in \mathbb{N}$, and s be a state in the region (q,R). Then,*

$$\mathit{opt}_k^{(q,R)}(s) = \inf\{\mathit{Cost} \mid \textit{controller wins from } s \textit{ in } k \textit{ steps and within cost } \mathit{Cost}\}$$

In order to compute the functions $\mathit{opt}_i^{(q,R)}$, it turns out that we need to handle primarily only three functions—min and max of a set of functions and the function:

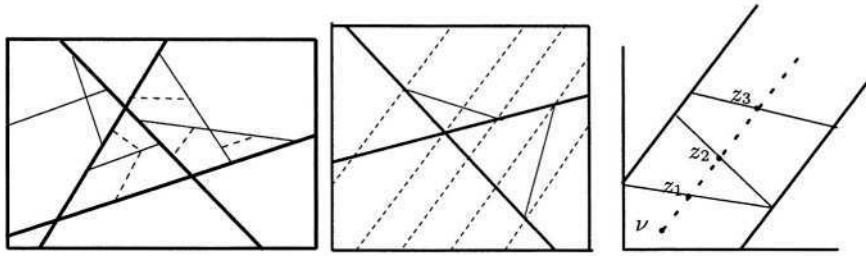
$$f(\bar{x}) = \inf_{t \in [0,T]} \max \begin{cases} f_1(\bar{x} + t) + wt \\ \sup_{0 \le t' \le t}(f_2(\bar{x} + t') + wt') \end{cases} \tag{1}$$

where f_1 and f_2 have already been computed, T is a critical clock expression and w is a constant.

4 The Algorithm

4.1 Motivation of Definitions

In this section, we describe briefly the main difficulties that arise in showing that each function $\mathit{opt}_i^{(q,R)}$ is a piece-wise linear function with at most an exponential number of pieces and informally describe the ideas to circumvent these. The next section gives a formal but terse summary of the required technical results.



(a) A nested partition (b) A tube partition (c) An atomic tube

Fig. 1.

We illustrate the ideas for the setting where we have two clocks $\{x_1, x_2\}$. Cost functions are hence functions from regions to $\mathbb{R}_+$, where a region is a subset of $\mathbb{R}_+^2$; these functions will be piece-wise linear and we refer to the 'pieces' as 'cells'.

As mentioned in the introduction, bookkeeping in terms of the number of cells does not suffice as superpositioning the cells of two functions f and g, each having n cells, could cause $O(n^2)$ cells and hence a double-exponential growth in cells. We can circumvent this difficulty by instead keeping track of the number of lines that partition the region into cells. If the cells of f and g are defined using at most n lines each, the superposition of cells of f and g are formed using at most $2n$ lines. Moreover, n lines can form at most $O(n^2)$ cells (for d-dimensions, n hyperplanes can form at most $O(n^d)$ cells), and the bookkeeping works as far as superpositioning is concerned.

However, when we take $h = \min\{f, g\}$, each new cell formed by the intersection of a cell of f and a cell of g gets further split into two (along the line where $f = g$) and causes an extra line to be added. Hence there could be $O(n^2)$ new lines defining cells in h and, again, leads only to a double-exponential bound on cell growth.

The crucial observation is that the new lines that are added are contained *within* a cell and *do not intersect with lines added in other cells.* For example, in Figure 1(a), the cells formed by lines are cut by the dashed lines into at most two parts but the dashed lines do not extend beyond the cell. We exploit this structure by introducing the notion of a *nested partition* of cells. A nested partition is a tree structure where every level of the tree refines the partition of the region by dividing cells into subcells. More formally, each node is associated with a cell and also associated with a set of lines that partition this cell into subcells. Figure 1(a) illustrates a nested partition: the three bold lines partition the region into 7 cells, the thin lines partition each such cell into at most 4 cells, and the dotted lines partition each of these cells into at most 2 cells.

The complexity of a nested partition is written as a tuple of numbers $\langle n_1, \ldots, n_i \rangle$ which means the following: the region is split by n_1 lines; *each* cell formed is further split by at most n_2 lines; in general, a cell formed at the j'th level is split by n_{j+1} lines. For instance, the nested partition in Figure 1(a) has complexity $\langle 3, 2, 1 \rangle$.

Now, if we take $\min\{f, g\}$ where f and g have complexity $\langle n_1, \ldots, n_i \rangle$, then we get a function that has complexity at most $\langle 2n_1, \ldots, 2n_i, 1 \rangle$ as the lines at each level add up and each atomic cell formed in the superposition of f and g can be split by one line, which causes a new level with a single line. The number of cells formed by a nested partition of complexity $\langle n_1, \ldots, n_k \rangle$, is at most $3^k n_1^d n_2^d \ldots n_k^d$, the growth of cells is hence under control and min and max operations can be handled.

Now let us consider the expression in (1). For any clock valuation ν, when time elapses, the points $\nu + t$ lie along a *diagonal* line drawn upwards from ν. The relevant positions that need to be examined for evaluating the expression for ν hence depend on this diagonal line and the cells that this diagonal line passes through.

In order to group together points which are such that the diagonal lines from the points meet the same set of cells, we draw diagonal lines from *every intersection of lines that form cells,* as illustrated by the dotted lines in Figure 1(b). This results in a set of diagonally placed cells that we call *tubes.*

Now consider an atomic tube (i.e. a tube within which the lines forming cells do not intersect) as shown in Figure 1(c). For any point ν, we can show that (i) the distance to any of the lines along the diagonal from ν is linear and (ii) in order to optimize the expression in (1), the values of t and t' must be at the times that correspond to when the diagonal from ν meets one of these lines (depicted as z_1, z_2 and z_3 in the figure). This reduces the quantification of t and t' over possibly infinite sets to a finite set; this leads us to reduce the expression in (1) to an expression that involves just min and max and we can use the procedures developed before to handle this.

However, when evaluating this expression, the cells could further get split and this could create new intersection points from which we may need to draw diagonals again in order to evaluate (1) in cells below the current cell. But we show that splitting of cells can happen only along diagonal lines which avoids this complication.

Finally, the number of diagonal lines introduced could be large compared to the number of lines defining the cells; however diagonal lines once formed cannot contribute further to forming diagonal lines. So we enhance the nested representation so that diagonal lines are accounted for separately and use the above property to show a bound on the growth of cells.

In dimensions higher than two, diagonal hyperplanes could intersect and to control their growth, we need a nesting of tubes as well. A nested tube partition is the final structure we use and it has a complexity of the form $\langle l_1, \ldots, l_m, n_1, \ldots, n_k \rangle$ where the l_i's denote the complexity of diagonal lines that contribute to defining nested tubes and the n_i's denote how each tube thus formed is further partitioned into nested cells.

4.2 Clock Space Geometry

Let $d \in \mathbb{N}$ denote the number of dimensions (d will be the number of clocks in the timed automaton) and consider the space $\mathbb{R}_+^d$. A *hyperplane* in $\mathbb{R}_+^d$ is a set

of points that satisfy an equation of the form $a_1x_1 + a_2x_2 + \ldots + a_dx_d + b = 0$. We say that such a hyperplane is *diagonal* if $\sum_{i=1}^{d} a_i = 0$.

A *cell* is a (convex) set of points of $\mathbb{R}_+^d$ that is bounded by hyperplanes. Formally, a cell is a ***d-dimensional*** set of points defined by a finite set of inequalities of the form $h(x_1, \ldots, x_d) \leq 0$, where $h(x_1, \ldots, x_d)$ is a linear expression over the variables $\langle x_1, \ldots, x_d \rangle$.

Let c be a cell defined by the inequalities I and let H be a set of hyperplanes. Then the hyperplanes in H partition c into a number of subcells. Formally, the set of *subcells* of c induced by H, $\mathit{Subcells}(c, H)$, is the set of all minimal cells c', where each c' is defined by I in conjunction with a set of inequalities of the form $h'(x_1, \ldots, x_d) \leq 0$, where $h'(x_1, \ldots, x_d) = 0$ belongs to H. It is well known that $\mathit{Subcells}(c, H)$ contains $O(|H|^d)$ cells (in fact $3|H|^d$ cells; see [13]).

Definition 2. *A* nested partition *of dimension* d *and depth* i *is a structure* (Tr, η, H) *where* $Tr = (V, E)$ *is a finite rooted tree, and for each* $v \in V$, η *is a function that maps* v *to a cell of dimension* d *and* H *is a function that maps* v *to a finite set of hyperplanes in* $\mathbb{R}_+^d$ *such that the following hold:*

- *A nested partition of depth* 0 *is* (Tr, η, H) *where* Tr *is a single node tree* $(\{v\}, \emptyset)$, η *maps* v *to a cell and* $H(v) = \emptyset$.
- *A nested partition of depth* $i + 1$ $(i \geq 0)$ *is a tree (V,E) that satisfies the following. If the root is* r, *then let* $c = \eta(r)$. *Then for every* $c' \in \mathit{Subcells}(c, H(r))$, *there is precisely one child* v *of the root such that* $\eta(v) = c'$ *and these are the only children of the root. Also the tree rooted at the children of the root must be nested partitions of depth* j, *where* $j \leq i$, *and the tree rooted at at least one child is a nested partition of depth* i.

The domain of a nested partition P is $D_P = \eta(\mathit{root})$ where *root* is the root of P, i.e. D_P is the cell that labels the root of the tree. For any tree, we say a vertex is at level i if its distance from the root is $i-1$ (the root is hence at level 1 and if a tree is of depth k, then there is a leaf at level $k+1$).

For a nested partition P, we say that P is of *complexity at most* $\langle n_1, \ldots n_k \rangle$, denoted $P \prec \langle n_1, \ldots n_k \rangle$, if P has depth k and for every vertex v, if v is at level i, then $H(v)$ contains at most n_i hyperplanes.

We are interested in the set of cells that are at the leaves of a nested partition P; let us call these *base cells* and denote the set of base cells as $BC(P) = \{c \mid \exists v \in V, v \text{ is a leaf, and } \eta(v) = c\}$. It is not hard to see that if $P \prec \langle n_1, \ldots n_k \rangle$, then $|BC(P)| \leq 3^k n_1^d \ldots n_k^d$.

We define an operation on nested partitions that takes two nested partitions P_1 and P_2 and creates the coarsest nested partition that refines both P_1 and P_2. This operation $P_1 \oplus P_2$ creates a nested partition P where for every two base cells c_1 in P_1 and c_2 in P_2, if $c' = c_1 \cap c_2$ is nonempty, then c' is a base cell of P. It turns out that if $P_i \prec \langle n_1^i, \ldots, n_k^i \rangle$, $i = 1, 2$, then $P_1 \oplus P_2 \prec \langle n_1^1 + n_1^2, \ldots, n_k^1 + n_k^2 \rangle$.

A *partition cost function* is a pair (P, F) where P is a nested partition and F is a mapping that maps each leaf node v of P to a linear expression f_v over the variables $\{x_1, \ldots, x_d\}$ such that the following condition holds: let u belong to two different base cells at leaves v and v'; then, $F(v)(u) = F(v')(u)$.

In other words, a partition cost function defines linear cost functions at the base cells and if a point is present in many base cells, then the cost for this point is the same at all these base cells. A partition cost function (P, F) hence assigns a cost to each point in the domain given by $\hat{F} : D_P \to \mathbb{R}_+$ with $\hat{F}(u) = F(v)(u)$ where v is any leaf such that the base cell at v contains u.

A *tube* is a cell that is formed by diagonal hyperplanes. Let $m_1, m_2 \in \mathbb{N}$. Then an (m_1, m_2) *nested tube partition* of dimension d is a nested partition P of dimension d that has depth $(m_1 + m_2)$ such that for all $i : 1 \leq i \leq m_1$, if v is a node at level i, then $H(v)$ contains only diagonal planes.

Let us now consider operations on partition cost functions and the change in complexity that the operations result in. For any partition cost function (P, F), let us denote the function it represents as $\hat{F}$.

Theorem 1. *Let (P_i, F_i), $i = 1, \ldots m$, be m d-dimensional partition cost functions, defined over the same domain D, $P_i \prec \langle n_1, \ldots, n_k \rangle$ for all i. Then there exists a partition cost function (P, F) over D, $P \prec \langle m \cdot n_1, \ldots, m \cdot n_k, m^2 \rangle$ such that $\hat{F} = \min_{i=1,\ldots,m}\{\hat{F}_i\}$.*

The above theorem also holds for the max function.[1]

Theorem 2. *Let (P_F, F) and (P_G, G) be two partition cost functions over the same domain, where P_F and P_G are (k_1, k_2) nested tube partitions of complexities at most $\langle l_1, \ldots l_{k_1}, n_1, \ldots n_{k_2} \rangle$. Consider the function:*

$$S(\bar{x}) = \inf_{t:\bar{x}+t\in D} \max \begin{cases} \hat{F}(\bar{x}+t) + wt \\ \sup_{0\leq t'\leq t} \hat{G}(\bar{x}+t') + wt' \end{cases} \tag{2}$$

Then there exists a partition cost function (P_K, K) such that P_K is a d-dimensional $(k_1 + 2, k_2 + 1)$ nested tube partition of complexity at most $\langle 2l_1, \ldots, 2l_{k_1}, (k_2+1)3^{k_2+1}(2n_1)^{d+1} \cdot \ldots \cdot (2n_{k_2})^{d+1}, (2n_1 + \ldots + 2n_{k_2} + 3)^2, 2n_1, \ldots, 2n_{k_2}, 7 \rangle$ and $\hat{K} = S$.

We prove the theorem using several lemmas. Below we will use the convention that if $\bar{x} = (x_1, \ldots x_m)$ is a vector then $\bar{x} + \alpha$ denotes $(x_1 + \alpha, \ldots x_m + \alpha)$.

Let P be the partition $P_F \oplus P_G$ with a new level created by taking the hyperplane along which F and G divide each cell. First, we show that the values of t and t' that we need to consider to evaluate $S(\nu)$ can be constrained to belong to the points at which the diagonal from ν meets the various hyperplanes of P, or 0. We now want to "drop diagonal hyperplanes" (as in Figure 1(b)) from each point of intersection of hyperplanes that define the nested partition.

For any node v of a nested partition P, let L_v denote the union of the sets of all hyperplanes that label v and its ancestors. If h_1 and h_2 are two non-diagonal hyperplanes, we say that (h_1, h_2) is a *ridge* if there exists a node v such that h_1 and h_2 belong to L_v. A simple counting argument shows the following:

[1] For precise complexity bounds, we must also establish bounds on the growth of the coefficients used in the definition of hyperplanes. Typically, the representation of coefficients grows when we consider intersections of hyperplanes, but these grow slowly (linearly).

Lemma 2. *Let $P \prec \langle n_1, \ldots n_k \rangle$ be a nested partition of dimension d. Then the number of ridges in P is bounded by $k \cdot 3^k \cdot n_1^{d+1} \cdot \ldots \cdot n_k^{d+1}$.*

We say that a tube partitioning P of type (k_1, k_2) is *atomic* if hyperplanes that partition cells at the last k_2 levels do not intersect with each other in the interior of the tube they belong to. We now want to "drop" diagonal hyperplanes from each ridge so that the resulting tubes are atomic. Using Lemma 2 we can show the following:

Lemma 3. *Let (P, F) be a partition cost function, where P is a d-dimensional (k_1, k_2) nested tube partition and $P \prec \langle l_1, \ldots l_{k_1}, n_1, \ldots n_{k_2} \rangle$. Then there exists a partition cost function (P', F'), where P' is atomic, which defines the same function (i.e. $\hat{F} = \widehat{F'}$) such that P' is a $(k_1 + 1, k_2)$ nested tube partition and $P' \prec \langle l_1, \ldots l_{k_1}, k_2 3^{k_2} n_1^{d+1} \ldots n_{k_2}^{d+1}, n_1, \ldots, n_{k_2} \rangle$.*

Note that if time elapses from two points within the same cell of an atomic tube, then they meet the same set of "border" hyperplanes (that partition the region). We can show that, for any cell, the time required by points in the cell to reach a particular border is a linear expression. Using the fact that the values of t and t' have to be evaluated only for the values that correspond to hitting these borders, we show we can rewrite the expression for S using min's and max's. When evaluating this expression, a cell within an atomic tube could get split and hence cause additional ridges from which we may have to drop diagonal hyperplanes—however, we show that this cannot happen as these splits will be diagonal hyperplanes themselves. A careful analysis of the cost of evaluating the expression then yields the theorem.

4.3 The Main Results

Theorem 3. *Given a WTA A and $k \in \mathbb{N}$, the (uniform) optimal bounded weighted timed game problem can be solved in time exponential in k and the size of A.*

We can also show that an optimal strategy may have to "split" a region into exponentially many parts:

Theorem 4. *For every $k \geq 0$, there is an* acyclic *WTA A_k with* 3 *clocks, where constants in the constraints of A_k are only* 0 *or* 1, *where all edges have weight zero and states have weights* 0 *or* 1 *and such that the number of states in A_k is bound by a fixed polynomial in k, such that the following holds:*
Let str be any optimal strategy for $(A_k, 3k)$. Then there is a region (q, R) and an exponential number of states $s_1, \ldots, s_{2^{k-1}}$, in (q, R) such that each of these states is visited by some play according to str and for every pair of distinct states s_i and s_j, the discrete components (i.e. the Σ labels) of the set of plays from s_i and from s_j according to str are different.

5 Discussion

We have established an exponential upper bound on computing the optimal cost for reachability games in weighted timed automata. The complexity of our procedure depends on the number of iterations. A bound on the number of iterations can be specified by the user, or can be obtained from the automaton if we assume that in every cycle a positive cost must be paid (such an assumption is typical to avoid problems with Zeno behaviors). However, the complexity (and even decidability) of the problem in the absence of such an assumption is open. Also, though we have shown that exponential splitting of a region is necessary for representing the optimal cost as a function of the initial state, the precise lower bound on the complexity of the decision problem remains open.

References

1. R. Alur, C. Courcoubetis, and T.A. Henzinger. Computing accumulated delays in real-time systems. *Formal Methods in System Design,* 11(2):137–155, 1997.
2. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science,* 126:183–235, 1994.
3. R. Alur, S. La Torre, and G. Pappas. Optimal paths in weighted timed automata. In *Hybrid Systems: Computation and Control,* LNCS 2034, pages 49–62, 2001.
4. E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *Hybrid Systems: Comp. and Control,* LNCS 1569, pages 19–30, 1999.
5. G. Behrman, T. Hune, A. Fehnker, K. Larsen, P. Petersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Hybrid Systems: Computation and Control,* LNCS 2034, pages 147–161, 2001.
6. F. Cassez, T.A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In *HSCC,* LNCS 2289, pages 134–148, 2002.
7. C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Proc. of Third Workshop on Computer-Aided Verification,* LNCS 575, pages 399–409. Springer-Verlag, 1991.
8. D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *Proc. STACS,* LNCS 2285, pages 571–582. Springer, 2002.
9. J. Ferrante and C. Rackoff. A decision procedure for the first order theory on real addition with order. *SIAM Journal of Computing,* 4(1):69–76, 1975.
10. M.J. Fischer and M.O. Rabin. Super-exponential complexity of Presburger arithmetic. In *Proc. of SIAM-AMS Symp. in Appl. Math.,* vol. 7, pages 27–41, 1974.
11. K. Larsen, G. Behrman, E. Brinksma, A. Fehnker, T. Hune, P. Petersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proc. of CAV,* LNCS 2102, pages 493–505. Springer, 2001.
12. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science,* LNCS 900, pages 229 – 242, 1995.
13. J. Matoušek. *Lectures on Discrete Geometry.* Springer, 2002.
14. S. La Torre, S. Mukhopadhyay, and A. Murano. Optimal-reachability and control for acyclic weighted timed automata. In *Proceedings of the 17th IFIP World Computer Congress: TCS,* pages 485–497. Kluwer, 2002.
15. H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. In *IEEE Conference on Decision and Control,* pages 1527–1528, 1991.

Wavelength Assignment in Optical Networks with Fixed Fiber Capacity

Matthew Andrews and Lisa Zhang

Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974
{andrews,ylz}@research.bell-labs.com

Abstract. We consider the problem of assigning wavelengths to demands in an optical network of m links. We assume that the route of each demand is fixed and the number of wavelengths available on a fiber is some parameter μ. Our aim is to minimize the maximum ratio between the number of fibers deployed on a link e and the number of fibers required on the same link e when wavelength assignment is allowed to be fractional.

Our main results are negative ones. We show that there is no constant-factor approximation algorithm unless NP$\subseteq$ZPP. No such negative result is known if the routes are not fixed. In addition, unless all languages in NP have randomized algorithms with expected running time $O(n^{\text{polylog}(n)})$, we show that there is no $\log^{\gamma}\mu$ approximation for any $\gamma \in (0,1)$ and no $\log^{\gamma} m$ approximation for any $\gamma \in (0,0.5)$. Our analysis is based on hardness results for the problem of approximating the chromatic number in a graph.

On the positive side, we present algorithms with approximation ratios $O(\log m + \log\mu)$, $O(\log D_{\max} + \log\mu)$ and $O(D_{\max})$ respectively. Here $D_{\max}$ is the length of the longest path.

We conclude by presenting two variants of the problem and discussing which of our results still apply.

Keywords: Optical networking, wavelength assignment, fixed capacity fiber, inapproximability.

1 Introduction

We consider the problem of achieving *transparency* in optical networks. A path is said to be routed transparently if it is assigned the same wavelength from its source to its destination. Transparency is desirable since wavelength conversion is expensive and defeats the advantage of all optical transmission.

More formally, we consider an optical network consisting of vertices and optical links and a set of demands each of which needs to be routed from a source vertex to a destination vertex on a single wavelength. Each optical link has one or multiple parallel fibers deployed. The fundamental constraint is that for each wavelength λ, each fiber can carry at most one demand that is assigned wavelength λ. A common problem is to minimize the number of wavelengths

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 134–145, 2004.

required so that all demands can be routed assuming one fiber per link. However, in reality a more pertinent problem is that the number of wavelengths that each fiber can carry is fixed to some value μ, i.e. the total number of wavelengths is fixed. (For example, [11] lists the fiber capacities from different vendors.) The problem now is to minimize the number of fibers required.

For most service providers, the cost of a fiber on a link can be divided into two components. First, there is the cost of renting the fiber from a "dark-fiber" provider. Second, there is the cost of purchasing optical equipment to "light" the fiber. When networks are being designed, the exact form of these costs are often not well known. For example, the dark-fiber providers may regularly update their rental rates and the cost of optical equipment may be subject to negotiation. Moreover, the service providers may have to rent from different dark-fiber providers in different parts of the country and each may have different pricing strategies. Therefore, over time the fiber cost may vary nonuniformly from link to link.

Despite this, we do know that the number of fibers we use on a link must be at least the total number of demands routed through the link divided by the number of wavelengths per fiber. One robust way to ensure our network cost is low regardless of the exact cost structure is to minimize the ratio between the number of fibers actually used on the link and this lower bound.

In this paper we assume that the path followed by each demand is already fixed. Wavelength assignment is therefore the only problem. In an alternative formulation, routing and wavelength assignment could be performed simultaneously. However, in many practical situations arising in optical network design, routing is determined by some higher-level specifications (e.g. carriers may require min-hop routing, see [10]). Hence, it is important to consider the wavelength assignment problem in isolation. We also remark that once a demand is assigned a wavelength, which fiber on each link actually carries the demand is not an issue. This is because modern optical devices such as mesh optical add-drop multiplexers allow distinct wavelengths from different fibers to be multiplexed into a new fiber.

Fiber minimization with fixed routing is NP-hard on networks with general topology (by a simple reduction from graph coloring). In this paper we focus on upper and lower bounds for approximating the problem.

1.1 Problem Definition and Preliminaries

We now describe the basic version of our problem. We consider a network and a set of demands D where each demand i is routed on a given path P_i. We require that each demand is assigned a wavelength λ from the set $\{0, 1, \ldots, \mu - 1\}$. For each link e, if at most r_e demands passing through link e are assigned wavelength λ for each λ, then the number of fibers required on link e is r_e. If ℓ_e is the number of paths that pass through e, then $f_e = \ell_e/\mu$ is clearly a lower bound on r_e.

There are a number of distinct ways to define the objective function. For the reasons mentioned earlier we focus on a variant in which our goal is to minimize the maximum ratio between the number of fibers deployed on a link

e and the corresponding lower bound f_e. (We mention some other variants in Section 5.) The problem may be formulated as an integer program. Let variable $C_{i,\lambda}$ indicate whether or not demand i uses wavelength λ. Our problem, which we call MIN-FIBER, can be written as follows for binary $C_{i,\lambda}$.

$$\begin{aligned} &\min z \\ \text{subject to} \\ &\sum_{i:e\in P_i} C_{i,\lambda} \le z \cdot f_e \qquad \forall e,\lambda \qquad (1) \\ &\sum_{\lambda} C_{i,\lambda} = 1 \qquad \forall i \qquad (2) \end{aligned}$$

We note that the linear relaxation of the above IP always has an optimal solution $z = 1$ and $C_{i,\lambda} = 1/\mu$ for all demands i and wavelengths λ.

1.2 Our Results

- We begin in Section 2 by presenting a negative result. We show that unless NP⊆ZPP there is no polynomial-time constant-factor approximation algorithm for the MIN-FIBER problem. ZPP is the class of languages that can be recognized using a randomized algorithm that always gives the correct answer and whose expected running time is polynomial in the size of the input. Our result is based on the hardness result for graph coloring of Feige and Kilian [9].
- In Section 3 we further improve the lower bound. Unless all languages in NP have randomized algorithms with running time $O(n^{\text{polylog}(n)})$, we show that there is no $\log^{\gamma}\mu$-approximation for any $\gamma \in (0,1)$ and no $\log^{\gamma} m$-approximation for any $\gamma \in (0, 0.5)$ where m is the number of links in the network.
- In Section 4 we turn our attention to positive results. In Section 4.1 we show that using randomized rounding we can obtain a solution in which the number of fibers required on each link e is at most $2f_e + 6(\log m + \log\mu)$. (All logarithms are to the base e.) This gives us an $O(\log m + \log\mu)$ approximation algorithm. We note that this algorithm can be derandomized using the standard method of conditional expectations.
  In Section 4.2 we apply the path-length rounding scheme of [12] to create a solution in which the number of fibers required on each link e is at most $f_e + D_{\max}$, where $D_{\max}$ is the length of the longest path in the network. This gives us an $O(D_{\max})$ approximation algorithm which is an improvement over the randomized rounding method when the paths are short.
  In the full version of the paper [2] we apply a constructive version of the Lovász Local Lemma to obtain a randomized algorithm with approximation ratio $O(\log D_{\max} + \log\mu)$ and polynomial expected running time.
- In Section 5 we conclude by presenting two variants of the MIN-FIBER problem and indicating which of our results still apply.

1.3 Previous Work

For the case in which the number of available wavelengths is not fixed, the problem of minimizing the number of wavelengths used has been much studied, e.g. [1,3,4,21]. Some papers focus on common special topologies such as rings [14, 24] and trees [16,15,7]. The work listed here is by no means complete. A good survey on the subject can be found in [13].

Our problem of fiber minimization with a fixed fiber capacity has been introduced more recently. In [25,18] the authors prove that coloring demands on a *line* only requires the minimum number of fibers per link, i.e. $\lceil f_e \rceil$ fibers on link e. This generalizes the well-known algorithm for coloring interval graphs. In addition, [18] shows that the problem becomes NP-hard once the network topology is more complicated. The authors provide 2-approximation algorithms for rings and stars. Recent work on trees include [6,8] and the results in [6] imply a 4-approximation. For a general network topology, [23] uses randomized rounding to obtain an approximation algorithm for the variant of the problem in which the aim is to minimize the total amount of fiber deployed.

2 Basic Lower Bound

In this section we show that there is no constant factor approximation to the MIN-FIBER problem unless NP⊆ZPP. Our construction is based on hardness of approximation results for graph coloring. For any graph G we use $\chi(G)$ to denote the chromatic number of G and $\alpha(G)$ to denote the size of the maximum independent set of G. Throughout this section we shall use the terms "color" and "wavelength" interchangeably.

Feige and Kilian [9] construct a randomized reduction from 3SAT to graph coloring with the following properties. Given a 3CNF formula φ and a constant ε, they randomly construct an n-node graph G (where n is polynomial in the size of φ) such that,

- If φ is satisfiable then with probability 1, G can be colored with n^ε colors, i.e. $\chi(G) \leq n^\varepsilon$.
- If φ is not satisfiable then with high probability the maximum independent set in G has at most n^ε nodes, i.e. $\alpha(G) \leq n^\varepsilon$ with high probability. Note that since $\alpha(G) \cdot \chi(G) \geq n$ this immediately implies that $\chi(G) \geq n^{1-\varepsilon}$.

Feige and Kilian use this reduction to show that there is no $n^{1-\varepsilon}$ approximation for graph coloring unless NP⊆ZPP. We shall use it to show that for any constant c there is no c-approximation for MIN-FIBER unless NP⊆ZPP.

2.1 Constructing an Instance of MIN-FIBER

We now demonstrate how to take a graph G and create an instance of MIN-FIBER on a network N. For each node v in G we have a demand d_v. The links in N consist of two sets E_1 and E_2. All links in E_1 are non-adjacent, i.e. no 2

links in E_1 have a vertex in common. The links in E_2 are used to connect up the links in E_1.

More precisely, for each clique Q in G with $c+1$ nodes we create a link e_Q in N and these links form the link set E_1. The demand d_v passes through e_Q for all $v \in Q$. If demand d_v has to pass through links $e_{Q_0}, \ldots, e_{Q_{z-1}}$ then there also exists a link $f_{v,j}$ in E_2 that connects the head of e_{Q_j} with the tail of $e_{Q_{j+1}}$. The full path of d_v is $e_{Q_0}, f_{v,0}, e_{Q_1}, \ldots, e_{Q_{z-2}}, f_{v,z-2}, e_{Q_{z-1}}$. We illustrate the construction of the network N from a graph G in Figure 1. The number of colors in our instance of MIN-FIBER is $\mu = n^{\varepsilon}$.

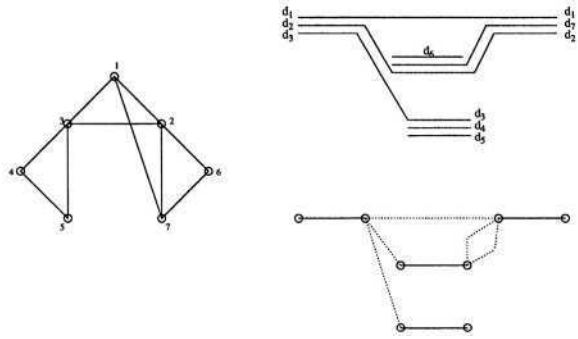


Fig. 1. An example of the construction for $c = 2$. (Left) Graph G with 4 cliques of size 3. (Upper right) Demands and routes created from G. (Lower right) Network N, solid lines represent links in E_1 and dotted lines represent those in E_2.

2.2 Reduction from 3SAT to MIN-FIBER

Given a 3CNF formula φ we first choose a constant ε such that $\varepsilon < \frac{1}{c+1}$. We then construct a random n-node graph G according to the method of Feige and Kilian [9] for this parameter ε. Finally, we convert the graph G into an instance of MIN-FIBER on a network N according to the method of the previous section. Note that since c is a constant, the number of demands and links in N are both polynomial in n which is in turn polynomial in the size of φ.

Lemma 1. *If φ is satisfiable then with probability 1 the demands in N can be colored such that at most one fiber is required on each link. If φ is not satisfiable then with high probability, for any coloring of the demands in N, some link requires $c+1$ fibers.*

Proof. Suppose that φ is satisfiable. Then with probability 1 the graph G is colorable with $\mu = n^{\varepsilon}$ colors. For any such coloring, we color the demands in N such that demand d_v receives the same color as node v. Clearly, for any clique Q in G and any color λ, there is at most one node in Q that receives color λ. Hence for any link e_Q in E_1, there is at most one demand passing through link

e_Q that receives color λ. Therefore each link in E_1 requires only one fiber in order to carry all its demands. The links in E_2 have only one demand and so they trivially require one fiber only. Hence at most one fiber is required on any link in N.

To prove the other direction, suppose that φ is unsatisfiable. Then with high probability $\alpha(G) \leq n^\epsilon$. Suppose for the purpose of contradiction that we can color the demands in N with $\mu = n^\epsilon$ colors such that each link requires at most c fibers. This implies that for any link e_Q in E_1, not all the demands passing through e_Q receive the same color. Consider now the corresponding coloring of the nodes in G.[1] By the construction of our network N, for any clique Q with $c+1$ nodes, not every node in Q receives the same color.

Let X be the induced subgraph of G on the set of nodes that constitutes the largest color class. We have just shown that X does not contain a clique of size $c+1$. Moreover, since X is contained in G, $\alpha(X) \leq \alpha(G) \leq n^\epsilon$. Ramsey's theorem (see e.g. [19]) immediately implies that,

$$|X| \leq \binom{\alpha(G) + (c+1) - 2}{(c+1) - 1} \leq \alpha(G)^c. \tag{3}$$

Since X constitutes the largest color class and there are n^ϵ colors, $|X|n^\epsilon \geq n$. Hence,

$$\begin{aligned} &|X| \geq n^{1-\epsilon} \\ \Rightarrow\ &\alpha(G)^c \geq n^{1-\epsilon} \\ \Rightarrow\ &\alpha(G) \geq n^{\frac{1-\epsilon}{c}} > n^\epsilon, \end{aligned}$$

since $\epsilon < \frac{1}{c+1}$. This contradicts the fact that $\alpha(G) \leq n^\epsilon$.

Theorem 1. *There is no c-approximation to* MIN-FIBER *for any constant* c *unless* $NP \subseteq ZPP$.

Proof. Suppose for the purpose of contradiction that C is a polynomial time c-approximation algorithm. We use this to construct a randomized algorithm B for 3SAT. For each instance φ, algorithm B creates a random graph G and then converts it to an instance of MIN-FIBER on a network N as described above. It then runs algorithm C on the instance of MIN-FIBER. If the solution returned by algorithm C is at most c then algorithm B returns "satisfiable", otherwise algorithm B returns "unsatisfiable". Lemma 1 implies that,

- If φ is satisfiable then the optimal solution to the instance of MIN-FIBER is 1. Since algorithm C is a c-approximation algorithm, it returns a value of at most c. Therefore algorithm B outputs "satisfiable".
- If φ is unsatisfiable then with high probability the optimal solution to the instance of MIN-FIBER is $c+1$. Therefore algorithm C returns a solution of at least $c+1$. Therefore algorithm B outputs "unsatisfiable".

[1] Note that this is not necessarily a proper coloring. Some edges in G may have both endpoints assigned the same color.

Note that algorithm B has one-sided error. Hence $3SAT \in coRP$ and so $NP \subseteq coRP$. This implies $RP \subseteq NP \subseteq coRP \subseteq coNP$ which in turn implies $NP = coNP = RP = coRP = RP \cap coRP = ZPP$.

3 Improved Lower Bound

In this section we derive more general hardness results by examining the construction of Feige and Kilian in more detail. In particular, given a 3CNF formula φ and a constant ε, they construct a random graph G on n nodes with parameters a, ρ, A and k. (As an aside, the parameters a, ρ and A are associated with a randomized Probabilistically Checkable Proof for NP and k is associated with a random graph product on a graph generated from the PCP. However, these interpretations are not important for our purposes.) The parameters are chosen so that the following relationships hold. More specifically, the parameters a and ρ are fixed to some constants such that (5) holds. The parameter A is polynomial in the size of φ and k is polylogarithmic in the size of φ. In particular, k is chosen sufficiently large such that Lemma 2 holds.

$$n = a^k \tag{4}$$

$$1 \geq \frac{\log a\rho}{\log a} \geq 1 - \varepsilon \tag{5}$$

$$A = \text{poly}(|\varphi|) \tag{6}$$

$$k = \Theta(\log^{1/\delta} |\varphi|) \qquad \text{for any } \delta \in (0,1) \text{ of our choice} \tag{7}$$

Feige and Kilian show that Graph G has the following properties.

1. If φ is satisfiable then with probability 1, G can be colored with $(1+\log n)/\rho^k$ colors, i.e. $\chi(G) \leq (1 + \log n)/\rho^k$.
2. If φ is not satisfiable then $\alpha(G) \leq kA$ with high probability, which implies $\chi(G) \geq n/(kA)$.

From the graph G we construct an instance of MIN-FIBER in the same manner as in the previous section. The number of links m in the newly constructed network N is $O(n^c)$. We set,

$$\mu = (1 + \log n)/\rho^k \tag{8}$$

$$c = \log^{1-\delta} n \qquad \text{for any } \delta \in (0,1) \text{ of our choice} \tag{9}$$

Lemma 2. *We can choose* $k = \Theta(\log^{1/\delta} |\varphi|)$ *such that* $\frac{(1+\log n)(kA)^c}{\rho^k} < n$.

Proof. Immediate from the parameter definitions.

The following is analogous to Lemma 1.

Lemma 3. *If φ is satisfiable then with probability* 1 *the demands in N can be colored with μ colors such that at most* 1 *fiber is required on each link. If φ is not satisfiable then with high probability, for any coloring of the demands in N, some link requires $c+1$ fibers.*

Proof. For the case in which φ is satisfiable, the proof is identical to Lemma 1.

For the other direction, suppose that φ is unsatisfiable but we color the demands in N with μ colors such that each link requires at most c fibers. Consider the corresponding coloring of G and let X be the induced subgraph of G on the set of nodes that constitutes the largest color class. As in inequality (3) in the proof of Lemma 1, $|X| \leq \alpha(G)^c$. By the construction of G, with high probability $\alpha(G) \leq kA$, which implies $|X| \leq (kA)^c$. Since X constitutes the largest color class and there are $\mu = (1+\log n)/\rho^k$ colors, $|X| \geq n/\mu = n\rho^k/(1+\log n)$. These inequalities imply that $(kA)^c \geq n\rho^k/(1+\log n)$ which contradicts Lemma (2).

Note that since we have a link in the network N for each subset of $c+1$ nodes in G, the size of the instance of MIN-FIBER is polynomial in n^c. The following is analogous to Theorem 1.

Theorem 2. *Unless 3SAT has a randomized algorithm with expected running time $O(|\varphi|^{polylog(|\varphi|)})$, there is no $\log^\gamma \mu$-approximation to* MIN-FIBER *for any $\gamma \in (0,1)$, and there is no $\Theta(\log^\gamma m)$-approximation for any $\gamma \in (0, 0.5)$. Here, μ is the number of colors per fiber and m is the number of links in* MIN-FIBER.

Proof. As in Theorem 1 we assume for the purpose of contradiction that C *is* a polynomial time c-approximation algorithm where c is defined in (9). From C we can construct a randomized algorithm B for 3SAT such that if φ is satisfiable then B outputs "satisfiable"; if φ is unsatisfiable then with high probability B outputs "unsatisfiable".

The correctness of B is identical to Theorem 1. The running time of B is $O(|\varphi|^{\text{polylog}(|\varphi|)})$ since both k and c are polylogarithmic in $|\varphi|$. Since $\mu \leq n$ and $m = O(n^c)$ we can show that $c > (\log \mu)^{1-\delta}$ and $c = \Omega((\log m)^{1-1/(2-\delta)})$. We note that B can give an incorrect answer with low probability. However, in the same way that NP⊆coRP implies NP⊆ZPP we can convert B into a randomized algorithm that always gives the correct answer and whose expected running time is $O(|\varphi|^{\text{polylog}(|\varphi|)})$.

4 Upper Bounds

4.1 Randomized Rounding

Recall that the linear relaxation of the our MIN-FIBER problem always has an optimal solution $z = 1$ and $C_{i,\lambda} = 1/\mu$ for all demands i and wavelengths λ. We adopt the technique of randomized rounding introduced in [20]. For each demand i we choose a number x_i uniformly at random in the range [0,1]. If $x_i \in [k/\mu, (k+1)/\mu)$ then we round $C_{i,\lambda}$ to 1 for $\lambda = k$ and round $C_{i,\lambda}$ to 0 for $\lambda \neq k$. After rounding the constraint (2) still holds. We use the Chernoff Bound [17] to see how much constraint (1) is violated. Let $\hat{C}_{i,\lambda}$ denote the rounded solution.

[Chernoff Bound.] If $X_1, \ldots, X_n$ are independent binary random variables where the expectation $x = E[\sum_i X_i]$, then it holds for all $\delta > 0$ that,

$$\Pr[\sum_i X_i \geq (1+\delta)x] \leq e^{-\min(\delta^2,\delta)\cdot x/3}.$$

Lemma 4. *For a particular link e and wavelength λ,*

$$\Pr\left[\sum_{i:e\in P_i} \hat{C}_{i,\lambda} \geq 2f_e\right] \leq m^{-2}\mu^{-2} \quad \text{if } f_e \geq 6(\log m + \log \mu),$$

$$\Pr\left[\sum_{i:e\in P_i} \hat{C}_{i,\lambda} \geq f_e + 6(\log m + \log \mu)\right] \leq m^{-2}\mu^{-2} \quad \text{if } f_e < 6(\log m + \log \mu).$$

Proof. By definition, the expected value of $E[\hat{C}_{i,\lambda}]$ is $1/\mu$. Hence, $E\left[\sum_{i:e\in P_i} \hat{C}_{i,\lambda}\right] = f_e$. Note that for a fixed link e and wavelength λ, the rounding of variables $C_{i,\lambda}$ for demands i that go through e are independent events. We can therefore apply the Chernoff Bound.
If $f_e \geq 6(\log m + \log \mu)$, then

$$\Pr\left[\sum_{i:e\in P_i} \hat{C}_{i,\lambda} \geq (1+1)f_e\right] \leq e^{-f_e/3} \leq e^{-2\log m - 2\log \mu} = m^{-2}\mu^{-2}.$$

If $f_e < 6(\log m + \log \mu)$, then

$$\Pr\left[\sum_{i:e\in P_i} \hat{C}_{i,\lambda} \geq \left(1 + \tfrac{1}{f_e}\cdot 6(\log m + \log \mu)\right) f_e\right] \leq e^{-2\log m - 2\log \mu} = m^{-2}\mu^{-2}.$$

By applying the union bound over all links and wavelengths, we obtain the following.

Theorem 3. *We can round the fractional optimal solution such that with high probability the number of fibers deployed on each link e is at most $2f_e + O(\log m + \log \mu)$. This implies an $O(\log m + \log \mu)$ approximation algorithm.*

We note that this algorithm can be derandomized by the standard method of conditional probabilities. We also note that for large values of f_e the approximation ratio approaches 2. Lastly, we remark that by using the slightly tighter Chernoff bound $Pr[\sum_i X_i \geq (1+\delta)x] \leq (e^\delta/(1+\delta)^{1+\delta})^x$, the approximation ratio can be marginally improved to $O(\frac{\log m}{\log\log m} + \frac{\log \mu}{\log\log \mu})$. However, for ease of exposition we typically ignore "log log" factors in this paper.

In the full version of the paper [2] we apply a constructive version of the Lovász Local Lemma (see for example Theorem 3.84 of [22]) to obtain a randomized algorithm with polynomial expected running time and approximation ratio $O(\log D_{\max} + \log \mu)$, where $D_{\max}$ is the maximum number of links along any demand path. In many optical networks $D_{\max}$ is significantly smaller than m. We omit the proof from this version in the interests of space.

Theorem 4. *We can round the fractional optimal solution such that the number of fibers deployed on each link e is at most $2f_e + 6(\log D_{\max} + \log \mu)$.*

4.2 Path Length Rounding

The following rounding theorem is due to Karp, Leighton, Rivest, Thompson, Vazirani and Vazirani [12].

[KLRTVV Rounding Theorem.] Let A be a real-valued $r \times s$ matrix and let x be a real-valued s-vector, let b be a real-valued r-vector such that $Ax = b$ and let Δ be a positive real number such that in every column of A,

1. *the column sum of the positive elements is at most Δ, and*
2. *the column sum of the negative elements is at least $-\Delta$.*

Then we can compute an integral s-vector $\hat{x}$ such that,

1. *$\hat{x}$ is a rounded version of x, i.e. $\hat{x}_i = \lfloor x_i \rfloor$ or $\hat{x}_i = \lceil x_i \rceil$ for $1 \leq i \leq s$, and*
2. *$A\hat{x} = \hat{b}$ where $\hat{b}_i - b_i \leq \Delta$ for all $1 \leq i \leq r$. In the case that all entries in A and b are integers, then a stronger bound applies: $\hat{b}_i - b_i \leq \Delta - 1$.*

It is easy to see that matrix A in the LP formulation of MIN-FIBER has 0/1 entries and its column sum is upper bounded by the longest path length plus 1, i.e. $\max_i |P_i| + 1$. By applying the KLRTVV Rounding Theorem, we obtain,

Theorem 5. *We can round the fractional optimal solution such that the number of fibers deployed on each link e is at most $f_e + D_{\max}$.*

5 Conclusions

In this paper we have presented positive and negative results for approximating the MIN-FIBER problem. We conclude by briefly discussing two variants of MIN-FIBER with different objective functions and seeing how our results apply. In the basic MIN-FIBER problem the objective is to minimize the ratio between the number of fibers deployed on link e and the lower bound f_e. For the sake of comparison, we restate the integer program.

Basic Version

$$\min z$$

subject to

$$\sum_{i:e \in P_i} C_{i,\lambda} \leq z \cdot f_e \quad \forall e, \lambda$$

$$\sum_{\lambda} C_{i,\lambda} = 1 \quad \forall i$$

In the first variant the new objective is to minimize the maximum, over all links e, of the number of fibers used on link e. As an integer program, this variant may be written as,

Variant 1

$$\min z$$

subject to

$$\sum_{i:e \in P_i} C_{i,\lambda} \leq z \quad \forall e, \lambda$$

$$\sum_{\lambda} C_{i,\lambda} = 1 \quad \forall i$$

We note that the hardness results of Sections 2 and 3 follow through, e.g. there is no constant-factor approximation for Variant 1 of MIN-FIBER unless NP⊆ZPP. This is because for the case in which the 3CNF formula ϕ is satisfiable, all links in the network require exactly 1 fiber. We also note that a lower bound on the optimal value of this problem is $\max_e f_e$. By concentrating on the link e with the maximum value of f_e, it is not hard to see that the approximation ratios proved in Section 4 also hold for this variant.

In the second variant we assume that we somehow know the cost per fiber on link e. We denote this cost by L_e. Our objective is to minimize the total cost of fiber needed to carry all the demands. We can formulate this variant as the following integer program.

Variant 2

$$\min \sum_e z_e L_e$$

subject to

$$\sum_{i:e\in P_i} C_{i,\lambda} \le z \quad \forall e, \lambda$$

$$\sum_\lambda C_{i,\lambda} = 1 \quad \forall i$$

Once again, the approximation ratios proved in Sections 4 apply to this variant. Furthermore, it can be shown that randomized rounding actually gives an $O(\log \mu)$ approximation for this variant [5]. However, our hardness results of Sections 2 and 3 no longer apply. Indeed, for the instances constructed in our reductions, randomized rounding gives a constant factor approximation for the problem of minimizing the total fiber length.

Acknowledgement. The authors wish to thank Chandra Chekuri, Bruce Shepherd and the anonymous referees for many helpful comments.

References

1. A. Aggarwal, A. Bar-Noy, D. Coppersmith, R. Ramaswami, B. Schieber, and M. Sudan. Efficient routing and scheduling algorithms for optical networks. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms,* pages 412 – 423, January 1994.
2. M. Andrews and L. Zhang. Wavelength assignment in optical networks with fixed fiber capacity. *Bell Labs Technical Memorandum,* 2003.
3. D. Banerjee and B. Mukherjee. A practical approach for routing and wavelength assignment in large wavelength-routed optical networks. *IEEE Journal on Selected Areas in Communications,* 14(5) :903 – 908, 1996.
4. I. Caragiannis, A. Ferreira, C. Kaklamanis, S. Perennes, and H. Rivano. Fractional path coloring with applications to WDM networks. In *Proceedings of the 28th International Colloqium on Automata, Languages, and Programming (ICALP '01),* pages 732 – 743, 2001.
5. C. Chekuri. Personal communication. 2003.
6. C. Chekuri, M. Mydlarz, and F. B. Shepherd. Multicommodity demand flow in a tree. In *ICALP,* 2003.

7. T. Erlebach, K. Jansen, C. Kaklamanis, M. Mihail, and P. Persiano. Optimal wavelength routing in directed fiber trees. *Theoretical Computer Science,* 221(1–2):119 – 137, 1999.
8. T. Erlebach, A. Pagourtzis, K. Potika, and S. Stefanakos. Resource allocation problems in multifiber WDM tree networks. In *Proceedings of the 29th International Workshop on Graph Theoretic Concepts in Computer Science,* pages 218 – 229, 2003.
9. U. Feige and J. Kilian. Zero knowledge and the chromatic number. In *IEEE Conference on Computational Complexity,* pages 278–287, 1996.
10. S. Fortune, W. Sweldens, and L. Zhang. Line system design for DWDM networks. In *Proceedings of the 11th International Telecommunications Network Strategy and Planning Symposium (Networks),* Vienna, Austria, 2004.
11. Alan Gnauck. Digital transmission. Post-OFC2004 Reviews, 2004.
12. R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani. Global wire routing in two-dimensional arrays. *Algorithmica,* 2:113 – 129, 1987.
13. R. Klasing. Methods and problems of wavelength-routing in all-optical networks. Technical Report CS-RR-348, Department of Computer Science, University of Warwick, 1998.
14. V. Kumar. Approximating circular arc coloring and bandwidth allocation in all-optical ring networks. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX '98),* pages 147–158, 1998.
15. V. Kumar and E. Schwabe. Improved access to optical bandwidth in trees. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms,* pages 437 – 444, January 1997.
16. M. Mihail, C. Kaklamanis, and S. Rao. Efficient access to optical bandwidth. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science,* pages 548 – 557, 1995.
17. R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge University Press, 1995.
18. C. Nomikos, A. Pagourtzis, and S. Zachos. Routing and path multi-coloring. *Information Processing Letters,* 2001.
19. G. Polya, R. Tarjan, and D. Woods. *Notes on Introductory Combinatorics, Progress in Computer Science, No. 4.* Birkhauser, Boston, Basel, Stuttgart, 1983.
20. P. Raghavan and C.D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica,* 7:365 – 374, 1991.
21. P. Raghavan and E. Upfal. Efficient routing in all-optical networks. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing,* 1994.
22. C. Scheideler. *Probabilistic Methods for Coordination Problems.* Habilitation thesis, Paderborn University, 2000.
23. B. Shepherd and A. Vetta. Lighting fibers in a dark network. *Bell Labs Technical Memorandum,* January 2003.
24. G. Wilfong and P. Winkler. Ring routing and wavelength translation. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms,* pages 333 – 341, January 1998.
25. P. Winkler and L. Zhang. Wavelength assignment and generalized interval graph coloring. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms,* January 2003.

External Memory Algorithms for Diameter and All-Pairs Shortest-Paths on Sparse Graphs

Lars Arge[1]*, Ulrich Meyer[2]**, and Laura Toma[3]***

[1] Department of Computer Science, Duke University, Durham, NC 27708, USA.
[2] Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany.
[3] Department of Computer Science, Bowdoin College, Brunswick, ME 04011, USA.

Abstract. We develop I/O-efficient algorithms for diameter and all-pairs shortest-paths (APSP). For general undirected graphs $G(V, E)$ with non-negative edge weights and $E/V = o(B/\log V)$ our approaches are the first to achieve $o(V^2)$ I/Os. We also show that for unweighted undirected graphs, APSP can be solved with just $O(V \cdot \text{sort}(E))$ I/Os. Both our weighted and unweighted approaches require $O(V^2)$ space. For diameter computations we provide I/O-space tradeoffs. Finally, we provide improved results for both diameter and APSP computation on directed planar graphs.

1 Introduction

Computing shortest paths and diameter of a graph are fundamental problems in algorithmic graph theory. For example, research in web modeling uses shortest path and diameter computations as primitive routines for investigating the structure of the web. Further applications often appear in Geographic Information Systems (GIS).

In the recent years an increasing number of graph applications involve massive graphs. When working with massive graphs, only a fraction of the data can be held in the main memory of a state-of-the-art computer. Thus, the transfer of data between main memory and secondary, disk-based memory, and not the internal memory computation, is often the bottleneck. Therefore, efficient external memory (or I/O-efficient) algorithms with optimized data access patterns can lead to considerable runtime improvements. Unfortunately, current shortest paths algorithms are only I/O-efficient on dense graphs whereas most real-world graphs are sparse. Therefore we aim to develop I/O-efficient shortest-path algorithms for sparse graphs of arbitrary structure. As a side effect of our research we also obtain significantly improved algorithms for the special case of planar directed graphs (digraphs).

* Supported in part by the National Science Foundation through ESS grant EIA–9870734, RI grant EIA-9972879 and CAREER grant EIA-9984099.
** Support by DFG grant SA 933/1-1. Part of this work was done while visiting Duke.
*** Part of this work was done while a PhD-student at Duke University.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 146–157, 2004.

1.1 Problem and Model Definitions

Let $G = (V, E)$[1] be an undirected weighted graph; we will call a graph sparse iff $E = O(V)$. Let s and t be two vertices in G. The shortest path $\delta(s,t)$ from s to t is the path of minimum length among all paths from s to t in G, where the length of a path is the sum of the weights of its edges. The length of the shortest path $\delta(s,t)$ is called the *distance* from s to t in G. The *single-source shortest-path* (SSSP) problem finds the shortest paths from a source vertex to all other vertices in G. The *all-pairs shortest-paths* (APSP) problem finds the shortest path between every pair of vertices in G. Often, one only needs the shortest-path distances and not the paths themselves. The *diameter* of G is the maximum distance between any two vertices of G. For unweighted graphs, SSSP and APSP are also referred to as *breadth-first search* (BFS) and *all-pairs breadth-first search* (AP-BFS).

Our results assume the standard two-level I/O-model with one (logical) disk [1]. The model defines two parameters: M is the number of vertices/edges that fit into internal memory, and B is the number of vertices/edges that fit into a disk block, where $M < V$ and $1 \leq B \leq M/2$. It is common practice to treat B and M as parameters even though for a fixed machine they may be constant (for example $B \simeq 10^6$ and $M \simeq 10^9$). An *Input/Output operation* (or simply *I/O*) is the operation of transferring a block of data between main memory and disk. The *I/O-complexity* of an algorithm is the number of I/Os it performs. The *scanning bound,* $\mathrm{scan}(N) = \frac{N}{B}$ is the number of I/Os necessary to read N contiguous items from disk. The *sorting bound,* $\mathrm{sort}(N) = \Theta(\frac{N}{B} \log_{M/B} \frac{N}{B})$ represents the number of I/Os required to sort N contiguous items on disk [1]. For all realistic values of V, B, and M, $\mathrm{scan}(V) < \mathrm{sort}(V) \ll V$ and $\log V \ll B$. An external-memory algorithm for a graph problem with internal-memory complexity $C(V, E)$ is usually called I/O-efficient if it performs $O(\mathrm{sort}(C(V,E)))$ I/Os.

1.2 Previous Work

There exists a vast number of results for APSP; for a survey see [16]. The classical method to solve APSP requires $\tilde{O}(V \cdot E)$ time[2] by subsequently running an SSSP algorithm for each vertex in the graph. Even though there are improved APSP algorithms for *dense* graphs (the currently best solution based on matrix multiplication requires $O(V^{2.575})$ time [17]), the classical method still constitutes by far the fastest way to solve APSP on general *sparse* graphs.

I/O-efficient graph algorithms have been considered by a number of authors; for a recent review see [12]. A direct conversion of the classical APSP approach to external memory requires an SSSP algorithm that is I/O-efficient on sparse graphs. However, the currently best algorithm for unweighted SSSP (i.e., BFS)

[1] For convenience we use the name of a set to denote both the set and its cardinality. Furthermore, we assume $E = \Omega(V)$ in order to simplify notation.

[2] We use $\tilde{O}(f(V, E))$ as a shorthand for $O(f(V,E) \cdot (\log V)^{O(1)})$.

needs $\Omega(V/\sqrt{B})$ I/Os on sparse graphs [11]; in the case of general non-negative edge weights, even $\Omega(V)$ I/Os are needed [8] (resulting in $\Omega(V^2)$ I/Os for the respective APSP conversion).[3] For directed graphs, the currently best BFS algorithms take $\Omega(V)$ I/Os [4,5]. This is far from the currently best lower bound for BFS (and SSSP) of $\Omega(\min\{V, \text{sort}(V)\} + E/B)$ I/Os.

The long and so far unsuccessful search for a BFS/SSSP algorithm using $\tilde{O}(\text{sort}(E))$ I/Os on general graphs has led to a number of improved results for special graph classes (e.g., BFS and SSSP can be solved using $O(\text{sort}(V))$ I/Os on planar graphs [2,10]). Seemingly, it has also discouraged researches from exploring I/O-efficient APSP/diameter algorithms for general sparse graphs.

1.3 Our Results

In this paper we show that the APSP and diameter problems can be tackled even if the respective I/O-efficient BFS/SSSP algorithms may not exist. In Section 2 we provide the first I/O-efficient algorithm on sparse undirected graphs with general non-negative edge weights. Under the realistic condition $E/V \leq B/\log V$, our algorithm needs $O(V \cdot (\sqrt{(V \cdot E \cdot \log V)/B} + \text{sort}(E)))$ I/Os, which is $O(V^2 \cdot \sqrt{(\log V)/B})$ I/Os on sparse graphs. Compared to the best previous approach (V times SSSP [8]) this is an improvement by a factor of up to $\Theta(\sqrt{B/\log V})$. Furthermore, in Section 3.1 we show that AP-BFS can be solved with just $O(V \cdot \text{sort(E)})$ I/Os. For sparse graphs this is an improvement by a factor of nearly $\sqrt{B}$.

The solutions above require $O(V^2)$ external space, thus matching the size of the output for APSP. For diameter computations, where the output size is $O(1)$, it is desirable to use only $\tilde{O}(E)$ space. In Section 3.2 we therefore provide I/O-space tradeoffs. In particular, we show how to solve the unweighted diameter problem on sparse graphs using $O(\text{sort}(k \cdot V^2 \cdot B^{1/k}))$ I/Os with $O(k \cdot V)$ space for any integer k, $3 \leq k \leq \log B$. Note that for the extreme case $k = \lfloor \log B \rfloor$ we require $O(\text{sort}(V^2 \cdot \log B))$ I/Os and $O(V \cdot \log B)$ space for sparse unweighted graphs.

Finally, in Section 4 we consider planar digraphs. We show that on this class of graphs APSP can be computed optimally in $O(\text{scan}(V^2))$I/Os and the diameter in only $O(\text{sort}(V^2)/B)$ I/Os and $O(V)$ space. This is a factor of B less I/Os than the previously best approach.

2 I/O-Efficient APSP on General Sparse Graphs

In this section we give an APSP algorithm for undirected sparse graphs with non-negative edge weights and show that it needs $O(V \cdot (\sqrt{(V \cdot E \cdot \log V)/B} + \text{sort}(E)))$ I/Os assuming $E/V \leq B/\log V$. Before describing our algorithm in Section 2.2, we first review some basic concepts for external-memory SSSP.

[3] It has been shown, however, that there is an algorithm for SSSP with bounded edge weights that requires $O(\sqrt{(V \cdot E/B) \cdot \log(W/w)} + \text{sort}(E))$ I/Os, where W and w are the largest and smallest weights in G, respectively [13].

2.1 Preliminaries

Dijkstra's algorithm [6] is the classical internal-memory SSSP approach. It uses a priority queue Q to store all vertices of G that have not yet been settled; the priority of a vertex v in Q is the length of the currently known shortest path from s to v. The next vertex v to be settled is retrieved from Q using a *deletemin* operation; then the algorithm relaxes the edges between v and all its non-settled neighbors, that is, performs a $\mathit{decrease\_key}(w, \mathrm{dist}(s,v) + \omega(v,w))$ operation for each such neighbor w whose priority is greater than $\mathrm{dist}(s,v) + \omega(v,w)$.

An I/O-efficient version of Dijkstra's algorithm has to (a) avoid accessing adjacency lists at random, (b) deal with the lack of efficient *decrease_key* operations in current external-memory priority queues, and (c) efficiently remember settled vertices. The SSSP algorithm of [8], **KS-SSSP** for short, ignores (a) and spends $\Omega(1)$ I/Os on retrieving the adjacency list of each settled vertex. As for (b), **KS-SSSP** uses an I/O-efficient "tournament tree" priority-queue, I/O-TT for short, which emulates *insert* and *decrease_key* operations using a weaker *update* operation, described below. As for (c), **KS-SSSP** performs an *update* operation for *every* neighbor of a settled vertex, which eliminates the need to remember previously settled vertices, but may re-insert settled vertices into the priority queue Q ("spurious updates"). Using a second priority queue Q^*, these re-inserted vertices are removed from Q before they can be settled for a second time: **KS-SSSP** proceeds in $O(E)$ rounds each of which examines the top-priority elements of both Q and Q^* and removes at least one of them (and eliminates a spurious element of Q if required)[4].

The I/O-TT of [8] supports three operations: (i) *deletemin* (extract the element $\langle x, k\rangle$ with smallest key k and replace it by the new entry $\langle x, \infty\rangle$); (ii) *delete(x)* (replace $\langle x, \mathit{oldkey}\rangle$ by $\langle x, \infty\rangle$); (iii) *update(x,newkey)* (replace $\langle x, \mathit{oldkey}\rangle$ by $\langle x, \mathit{newkey}\rangle$ if $\mathit{newkey} < \mathit{oldkey}$). Note that (ii) and (iii) do not require the old key to be known.

The I/O-TT is based on a complete binary tree, where some rightmost leaves may be missing. Let $M' = c \cdot M$ for some positive constant $c < 1$; the I/O-TT for V elements has $\lceil V/M' \rceil$ leaves and hence $O(\log(V/M))$ levels. Elements with indices in the range $\{i \cdot M', \ldots, (i+1) \cdot M' - 1\}$ are mapped to the i-th leaf. The index range of internal nodes of the I/O-TT is given by the union of the index ranges of their children. Internal nodes of the I/O-TT keep a list of at least $M'/2$ and at most M' elements each (sorted according to their priorities). If the list of a tree node v contains z elements, then they are the smallest z out of all those elements in the tree being mapped to the leaves that are descendants of v. Furthermore, each internal node is equipped with a signal buffer of size M'. Initially, the I/O-TT stores the elements $\langle 0, +\infty\rangle, \langle 1, +\infty\rangle, \ldots, \langle V-1, +\infty\rangle$, out of which the lists of internal nodes keep at least $M'/2$ elements each.

[4] The original version of **KS-SSSP** [8] does not properly handle adjacent vertices with the same SSSP-distance. This can be fixed by processing all such vertices in one round [15]. Essentially the same idea also works with our new APSP approach. Details will be given in the full version of this paper.

The operations (i)–(iii) generate *signals* that serve to propagate information down the tree; signals axe inserted into the root node, which is always kept in internal memory. When a signal arrives in a node it may create, delete or modify an element kept in this node; the signal itself may be discarded, altered or remain unchanged. Non-discarded signals are stored until the capacity of the node's buffer is exceeded; then they are sent down the tree towards the unique leaf node its associated element is mapped to. The following amortized bound has been shown:

Lemma 1 (I/O-TT [8]). *Any sequence of z delete, deletemin and update operations on an I/O-efficient tournament tree with V elements requires at most $O(z/B \cdot \log(V/M)) = O(z/B \cdot \log(V/B))$ I/Os.*

2.2 Multi-tournament-Trees and Concurrent SSSP Computations

We first consider how to bundle I/O-TTs in order to support I/O-efficient APSP. Then we present our new approach, **Fast-APSP.** Note that the bound of Lemma 1 is obtained by choosing $M' = \Theta(B)$. Setting $M' \ll B$ will result in a worse bound of $O(z/M' \cdot \log(V/M'))$ I/Os, which is usually not desired. However, such a choice allows us to bundle the corresponding nodes of a number of I/O-TTs in one disk block. Concretely speaking, we consider *I/O-efficient multi-tournament-trees,* I/O-MTTs for short. Let $L \geq 2$ a parameter to be fixed later. An I/O-MTT consists of L independent I/O-TTs $T_0, \ldots, T_{L-1}$ with parameter $M' = \Theta(B/L)$. In particular, this means that the root nodes of all L bundled I/O-TTs can be kept in one block in internal memory while performing operations on the different I/O-TTs (unless refilling occurs which, however, is already accounted for in the amortized I/O-bound of $O(\log(V/M')/M') = O(L \log V/B)$ I/Os for single operations on each of the bundled I/O-TTs).

Our new approach, **Fast-APSP,** uses **KS-SSSP** (Section 2.1) as a building block. However, it solves APSP by working on all V underlying SSSP problems concurrently. This requires V priority-queue pairs (Q_i, Q_i^*), $0 \leq i < V$, where the entries of (Q_i, Q_i^*) belong to the i-th SSSP problem. The set of priority queues is implemented using I/O-MTTs. This requires $O(V^2)$ space.

Fast-APSP proceeds in $O(V)$ rounds. In each round it loads the roots of all its I/O-MTTs and extracts a settled graph node with smallest distance from each of the *L/2* priority queue pairs (Q_i, Q_i^*) of the current I/O-MTT. Note that for each pair (Q_i, Q_i^*) this may require some initial *deletemin* operations on Q_i^* (coupled with *delete* operations on Q_i) before a settled graph node can be extracted from Q_i. Instead of accessing the required adjacency lists of settled nodes for each SSSP problem separately, **Fast-APSP** creates a node sequence $\mathcal{K}$ of the extracted settled vertices from this round, sorted according to these nodes' indices. Then it applies a parallel scan of the graph representation in order to retrieve the adjacency lists of all nodes in $\mathcal{K}$. Finally, another sorting and parallel scanning step is applied to move these adjacency lists back to the priority-queue pairs of the SSSP problems they originated from in order to perform the necessary priority queue update operations there. This requires another

cycling through all I/O-MTTs, which can be overlapped with the beginning of the next round. During each round the total size of data scanned and sorted is bounded by $O(V^2)$; summed over all rounds it is bounded by $O(V \cdot E)$. All computed distance values can be trivially appended to an output list of size $O(V^2)$, which is eventually sorted in order to produce the final distance matrix.

Fast-APSP replaces $\Omega(V^2)$ I/Os for separate adjacency-list accesses by $O(\text{sort}(V \cdot E))$ I/Os for joint accesses. The I/O complexity for the priority-queue operations is as follows: over $O(V)$ rounds, the cycling through the roots of the $O(V/L)$ I/O-MTTs takes $O(V^2/L)$ I/Os. On each of the bundled $2 \cdot V$ I/O-TTs we perform $O(E)$ priority queue operations, or $O(VE \cdot L \log V/B)$ I/Os in total. Sorting the output list requires $O(\text{sort}(V^2))$ I/Os. The sum of these terms is optimized by choosing $L = 2 \cdot \sqrt{B \cdot V/(E \cdot \log V)} \geq 2$. We have obtained the following:

Theorem 1. *APSP on undirected sparse graphs with non-negative edge weights can be solved using* $O\left(V \cdot \left(\sqrt{(V \cdot E \cdot \log V)/B} + sort(E)\right)\right)$ *I/Os and* $O(V^2)$ *space whenever* $E/V \leq B/\log V$.

3 AP-BFS and Unweighted Diameter on General Graphs

In this section we give improved algorithms for APSP and diameter on general undirected graphs with *unweighted* edges (AP-BFS). We first present an $O(V \cdot \text{sort}(E))$ I/O solution based on the BFS algorithm in [11]. In the worst case, however, this approach requires $\Theta(V^2)$ space even if we omit producing the output matrix (i.e., in the case of diameter computation). Therefore, we propose an alternative approach, which also serves as a basis for our I/O-space tradeoffs. In this section we assume $E = O(V \cdot B)$ since for dense graphs V times BFS [14] will trivially require $O(V \cdot \text{sort}(E))$ I/Os.

3.1 AP-BFS and Unweighted Diameter Using $O(V \cdot \text{sort}(E))$ I/Os

Our new AP-BFS algorithm, **Fast-AP-BFS,** builds on the **Fast-BFS** algorithm of [11]. **Fast-BFS** operates in two phases. Let μ be a parameter $\mu \in [0, 1]$ to be fixed later. The first phase partitions G into $O(\mu \cdot V)$ disjoint subgraphs (clusters) S_i using a spanning tree/Euler tour method, such that the distance between any two vertices of S_i is at most $1/\mu$ in G. Each S_i consists of at most $1/\mu$ vertices. For each cluster S_i, it creates a list $\mathcal{F}_i$ containing the adjacency lists of all vertices in S_i. The second phase of **Fast-BFS** is a modified version of the BFS-approach of [14] in the sense that it keeps a *hot pool* $\mathcal{H}$ of adjacency lists. $\mathcal{H}$ prevents the algorithm from performing V random accesses to the adjacency lists. The hot pool consists of all adjacency lists $\mathcal{F}_i$ such that the current level $L(t)$ of the BFS-tree contains a vertex in S_i. $\mathcal{H}$ is kept sorted by vertex indices. To construct the next level $L(t+1)$ of the BFS-tree it scans $\mathcal{H}$ and $L(t)$ in parallel: if the adjacency list of a vertex $u \in L(t)$ is found in $\mathcal{H}$ then it is extracted from $\mathcal{H}$; otherwise, if it is not in $\mathcal{H}$, the data from $\mathcal{F}_i$ corresponding to the cluster S_i

containing vertex u is merged into the hot pool. After all the adjacency lists of the vertices in $L(t)$ have been obtained, the tentative next level $L(t+1)$ can be generated, duplicates removed and vertices that appear in $L(t-1)$ or $L(t)$ discarded, just like in [14]. The remaining vertices constitute $L(t+1)$.

The main idea of the I/O-analysis is as follows: Scanning and sorting all BFS levels and merging each $\mathcal{F}_i$ into the pool takes $O(\text{sort}(E))$ I/Os in total. Each list $\mathcal{F}_i$ is copied into the pool precisely once. Since there are $O(\mu \cdot V)$ clusters, this takes $O(\mu \cdot V + \text{scan}(E))$ I/Os in total. Once $\mathcal{F}_i$ is brought in the hot pool, the adjacency list of a vertex in S_i stays in the pool until the BFS-tree reaches the vertex and deletes its adjacency list from the pool. Because the shortest path between any two vertices of S_i in G is $O(1/\mu)$ the adjacency list of a vertex may stay in the pool for at most $O(1/\mu)$ levels. Thus every adjacency list is scanned $O(1/\mu)$ times. Scanning the adjacency lists of all vertices in the graph throughout the algorithm takes $O(1/\mu \cdot E/B)$ I/Os in total. The total number of I/Os for the two stages is minimized by choosing $\mu = \sqrt{E/(V \cdot B)}$. The total resulting bound for **Fast-BFS** is $O(\sqrt{V \cdot E/B} + \text{sort}(E) \cdot \log\log(V \cdot B/E))$ I/Os [11].

Fast-AP-BFS: The straightforward way of performing AP-BFS is to use **FAST-BFS** on each vertex of the graph using $O(V \cdot \sqrt{V \cdot E/B} + \text{sort}(E) \cdot \log\log(V \cdot B/E))$ I/Os in total. This bound can be improved by running **Fast-BFS** in parallel *from all source vertices in the same cluster* and considering source clusters one after the other. Hence, **Fast-AP-BFS** initially computes (once and for all) the clustering like **Fast-BFS.** Then it iterates through all clusters: given a fixed source cluster S it runs **Fast-BFS**(s) for all source nodes $s \in S$ in parallel using a *common* hot pool $\mathcal{H}$. Growing level $L(t)$ in all BFS-trees in turn is called a *round.* Just like in **FAST-BFS,** each list $\mathcal{F}_i$ is brought into the hot pool precisely once for each source cluster. However, once brought in the pool, the adjacency list of every vertex $v \in S_i$ will have to remain in the pool until *all* the BFS-trees of the current source cluster S reach vertex v. This takes at most $O(1/\mu)$ rounds (and simple counting methods are sufficient to remove the respective lists from $\mathcal{H}$ after that time): let the exploration of the node $v \in S_i$ with BFS-level k in the BFS-tree of $s \in S$ have caused merging $\mathcal{F}_i$ into $\mathcal{H}$. Now it is easy to see that the BFS-level of any other node $v' \in S_i$ concerning any other BFS-tree with source $s' \in S$ is at most $k + 2/\mu$. This is a direct consequence of our clustering, which ensures that the shortest-paths $\delta(s, s')$ and $\delta(v, v')$ in G are bounded by $1/\mu$ each.

I/O-complexity of ***Fast-AP-BFS:*** For each of the $O(\mu \cdot V)$ source clusters we have the following contributions: Scanning and sorting all BFS levels and merging all $\mathcal{F}_i$ into the pool still takes $O(\text{sort}(E))$ I/Os per source vertex, or $O((1/\mu) \cdot \text{sort}(E))$ I/Os in total for the source cluster. Also, the number of I/Os to load in the pool all adjacency lists is still $O(\mu \cdot V + \text{scan}(E))$. Finally, each adjacency list stays in $\mathcal{H}$ for at most $2/\mu$ rounds, which accounts for other $O(\text{scan}(E/\mu))$ I/Os. Summing over all source clusters, adding the single pre-processing phase, and recalling $\mu = \sqrt{E/(V \cdot B)}$, we see that **Fast-AP-BFS**

requires $O(\mu \cdot V \cdot (\mu \cdot V + 1/\mu \cdot \text{sort}(E)) + \text{sort}(E) \cdot \log\log(V \cdot B/E)))$ I/Os $= O(V \cdot \text{sort}(E))$ I/Os.

Theorem 2. *Undirected AP-BFS can be computed using $O(V \cdot sort(E))$ I/Os and $O(V^2)$ space.*

Fast-AP-BFS can be used to compute the (unweighted) diameter in the same I/O-bound and with $O((V+E)/\mu) = O(\sqrt{V \cdot E \cdot B})$ space. For sparse graphs this is $O(V \cdot \sqrt{B})$ space.

3.2 I/O-Space Tradeoffs for Unweighted Diameter Computations

In this section we sketch how to reduce the space use at the cost of increasing the I/O use. We concentrate on an I/O-space tradeoff for the practically most relevant case $E = O(V)$. Our diameter algorithm makes space a top priority: it requires $O(\text{sort}(k \cdot V^2 \cdot B^{1/k}))$ I/Os with $O(k \cdot V)$ space for any integer k, $3 \le k \le \log B$. In the full version of this paper we also describe another approach that prioritizes I/O and results in $O(\text{sort}(k \cdot V^2))$ I/Os using $O(k \cdot V \cdot B^{1/k})$ space.

Space Priority Approach. Let us first give the intuition for $k = 3$. We are still using a clustering but now the clusters have reduced size $O(B^{1/3})$. Instead of running BFS for *all* vertices of a source cluster S *in parallel* we just select *one* arbitrary node $s \in S$ and use **Fast-BFS** to compute $\text{BFS}(s)$ in order to find the maximum distance from s to any other node in G. While doing so we append the cluster lists $\mathcal{F}_i$ to some sequence $\mathcal{R}$ in the order they are merged into the hot pool $\mathcal{H}$. For each $\mathcal{F}_i$ we also record in which round it was added to $\mathcal{H}$. Using $\mathcal{R}$ we can subsequently execute **Fast-BFS**(s') for each vertex $s' \in S \setminus \{s\}$ separately (and update the global diameter value) without any random I/O: again, we exploit the observation from Section 3.1 that the BFS level of any vertex v changes by at most $O(|S|)$ when we consider different source nodes from the same source cluster S. Therefore, by simply scanning $\mathcal{R}$ we can bring all required $\mathcal{F}_i$ to the hot pool in time while not performing unstructured I/O. Hence, for each source cluster S the dominating I/O-costs are: $O(V/B^{1/3} + \text{sort}(V))$ I/Os for $\text{BFS}(s)$ with reduced cluster size and $O(B^{1/3} \cdot (\text{scan}(V \cdot B^{1/3}) + \text{sort}(V))$ I/Os for the $O(B^{1/3})$ other BFS computations with sources in S. Summed over all $O(V/B^{1/3})$ source clusters the total I/O is bounded by $O(\text{sort}(V^2 \cdot B^{1/3}))$. As we only execute one BFS at a time the working space is bounded by $O(E)$.

In the following we extend this approach to $k-1 > 2$ levels. In a precomputing step we create a $(k-2)$-level clustering using the spanning tree/ Euler tour method of [11]: level $1 \le i \le k-2$ comprises clusters of (at most) $B^{i/k}$ nodes, Level-$(i-1)$ clusters are derived from level-i clusters by chopping them into (at most) $B^{1/k}$ pieces. Then we iterate over all level-$(k-2)$ source clusters S and for each of them proceed as follows: for an arbitrary vertex $s \in S$ we run **Fast-BFS** using the level-1 clusters and creating the sorted sequence $\mathcal{R}_{k-2}$, which contains the level-1 clusters in sorted order of visiting during **Fast-BFS**(s). Then we call the procedure $A(k-3, S)$ (see below) and after returning from $A()$ remove $\mathcal{R}_{k-2}$.

The procedure $A(\text{int } i, \text{Cluster } C)$ does the following: if $i = 0$ then it computes BFS one-by-one for all nodes v_j in C using $\mathcal{R}_{k-2}$ without random I/O and possibly update the global diameter. Otherwise $(i \geq 1)$ it performs the following for each level-i subcluster C' of C: Run **Fast-BFS**(v) from a single arbitrary node v in C' using $\mathcal{R}_{i+1}$ and record from what time on the level-1 clusters are actually needed in the hot pool; this gives a more appropriate sequence $\mathcal{R}_i$ for C' which will be used in the recursive call $A(i-1, C')$. Finally, after returning from $A(i-1, C')$ the sequence $\mathcal{R}_i$ is removed.

Hence, the main difference when going from $k = 3$ to general $k > 3$ is that we apply a stepwise refinement from $\mathcal{R}_{i+1}$ to $\mathcal{R}_i$ whenever we recurse on a subcluster. For each level-$(k-2)$ source cluster we run one **Fast-BFS** computation with random I/O based on lists $\mathcal{F}_j$ hosting at most $B^{1/k}$ adjacency lists each; this takes $O(V/B^{(k-2/k} \cdot (V/B^{1/k} + \text{sort}(V \cdot B^{1/k}))$ I/Os. On level-i source clusters we compute BFS without random I/O (using $\mathcal{R}_{i+1}$) but adjacency lists may stay in the hot pool for up to $\Theta(B^{(i-2)/k})$ rounds—this is due to the fact that $\mathcal{R}_{i+1}$ was recorded for a source node that may be at distance $\Theta(B^{(i-2)/k})$ from the current source node. Fortunately, on level i, there are only $O(V/B^{i/k})$ such BFS computations during the whole algorithm, each of which takes $O(\text{sort}(V \cdot B^{(i+1)/k}))$ I/Os. Hence, the whole diameter computation can be accomplished using $O(\text{sort}(k \cdot V^2 \cdot B^{1/k}))$ I/Os. The space bound follows from the observation that at all times the algorithm keeps at most $k-2$ sequences $\mathcal{R}_i$.

Theorem 3. *The diameter of unweighted undirected graphs with $E = O(V)$ edges can be found using $O(\text{sort}(k \cdot V^2 \cdot B^{1/k}))$ I/Os and $O(k \cdot V)$ space for any integer k, $3 \leq k \leq \log B$.*

4 APSP for Planar Digraphs

In this section we show how to compute APSP on planar digraphs in optimal $O(\text{scan}(N^2))$ I/Os while improving the straightforward bound of $O(\text{sort}(N^2))$ I/Os. In Section 4.1 we first give some preliminaries and a review of the known planar SSSP algorithm [2,3]. Then we describe the new the APSP/diameter algorithm in Section 4.2.

4.1 Preliminaries

Let $G = (V, E)$ be a planar graph with N vertices. An $f(N)$-*separator* of G is a subset S of the vertices of G of size $f(N)$ such that the removal of S disconnects G into two subgraphs G_1 and G_2, each of size at most $2N/3$. Lipton and Tarjan [9] proved that any planar graph has an $O(\sqrt{N})$-separator. Using this result recursively, Frederickson [7] showed that for any parameter $R \in [1, N]$ a planar graph can be partitioned into $\Theta(N/R)$ subgraphs (clusters) G_i of size $O(R)$ each such that there are $\Theta(N/\sqrt{R})$ separator vertices in total and each cluster is adjacent to $O(\sqrt{R})$ separator vertices (called the *boundary vertices* of G_i or simply the *boundary* ∂G_i of G_i). Denote this partitioning an *R-partition.*

Denote $G_i \cup \partial G_i$ as $\overline{G_i}$. The set of separator vertices can be partitioned into maximal subsets so that the vertices in each subset are adjacent to the same set of subgraphs G_i. These sets are called the *boundary sets* of the partition. If the graph has bounded degree, which can be ensured for planar graphs using a simple transformation [7], there exists an R-partition that, in addition to the above properties, has only $O(N/R)$ boundary sets [7]. For planar directed graphs R-partitions can be defined and computed in the same way ignoring the direction of the edges.

The main idea of the planar SSSP algorithms [2,3] is to compute a B^2-partition of G and use it to reduce the SSSP problem on G to the SSSP problem on a substitute graph G^R having as vertices the $O(N/B)$ separator vertices and $O(N)$ edges. The substitute graph is constructed using the following observation: let $\delta(s,t)$ be the shortest path from s to t in G, and let α, β be two vertices on the path such that α, β are on the boundary ∂G_i of some cluster G_i and all vertices between α and β on the path are in G_i; then the subpath of $\delta(s,t)$ from α to β is the shortest path from u to v in $\overline{G_i}$. Using this observation the substitute graph is defined so that it contains all separator vertices and the edges between them G; and, for every cluster G_i, for every pair of separator vertices α, β on ∂G_i, it contains an edge (α, β) of weight equal the the weight of the shortest path $\delta_{\overline{G_i}}(\alpha, \beta)$. It can be shown that for any $\alpha, \beta \in G^R$ the shortest path $\delta_{G^R}(\alpha, \beta) = \delta(\alpha, \beta)$. Thus G^R preserves shortest paths in G. Given G^R, SSSP in G^R can be computed exploiting that there are $O(N/B)$ vertices (and thus one can afford to use $O(1)$ I/Os per vertex) and that the $O(N)$ accesses to the edges can be grouped into $O(N/B)$ accesses to boundary sets.

4.2 Planar I/O-Efficient APSP

To compute APSP we use the same technique as the planar SSSP algorithm, namely computing a substitute graph G^R and computing shortest paths in G^R. The extra ingredient is the idea to compute shortest paths between *all the vertices in a cluster* and the vertices of the graph while the cluster is in memory. The basic steps are the following:

1. Compute a B^2-partition of G into $O(N/B)$ separator vertices and $O(N/B^2)$ clusters of size $O(B^2)$ each and $O(B)$ boundary vertices.
2. Compute the substitute graph G^R defined as in Section 4.1.
3. For each cluster G_i
   a) For every vertex α in ∂G_i compute $\mathrm{SSSP}(\alpha)$ in G^R using the planar SSSP algorithm in [2,3].
   b) For every vertex in $u \in \overline{G_i}$ compute $\mathrm{SSSP}(u)$ in G as follows: Load $\overline{G_i}$ in memory. For every cluster G_j load in memory $\overline{G_j}$ and the distances from ∂G_i to ∂G_j and, for any vertices $u \in \overline{G_i}$ and $v \in \overline{G_j}$, compute the shortest path from u to v as $d(u,v) = \min_{\alpha \in \partial G_i, \beta \in \partial G_j} \{\delta_{\overline{G_i}}(u, \alpha) + \delta_{G^R}(\alpha, \beta) + \delta_{\overline{G_j}}(\beta, v)\}$. If u, v are in the same cluster then $d(u,v)$ is the smaller of $d(u,v)$ computed as above and $\delta_{G_i}(u,v)$.

It can be shown that $d(u,v)$ is indeed the shortest path from u to v in G. We now discuss in more detail the steps and analyze their I/O-complexity. A B^2-partition can be computed in $O(\text{sort}(N))$ I/Os [10]. The substitute graph G^R can be computed in $O(\text{scan}(N))$ I/Os as in [2,3]. Let V_σ be the list of vertices of G in the following order: all separator vertices are at the front of V_σ grouped by boundary set, and within the same boundary set in order of their vertex index; then follow the vertices in the clusters G_i, grouped by cluster and within the same cluster grouped by vertex index. Given that we know for each separator vertex the boundary set which contains it and for every non-separator vertex the cluster which contains it, we can produce V_σ in $O(\text{sort}(N))$ I/Os. Moreover, also in sorting time, we can associate to each vertex v its position $\sigma(v)$ in V_σ.

For each cluster G_i, computing SSSP in G^R from all the vertices on the boundary of G_i uses $O(B \cdot \text{sort}(N))$ I/Os and $O(B \cdot N/B) = O(N)$ space. Let L_i be the resulting list of distances $L_i = \{\delta(\alpha,\beta) | \alpha \in \partial G_i, \beta \in G^R\}$. The next phase of the algorithm will access L_i to retrieve the distances to from ∂G_i to ∂G_j for any cluster G_j. In order to make these accesses efficient we store L_i such that the distances to vertices in the same boundary set are adjacent. We can obtain this representation of L_i by sorting the distances $\delta(\alpha,\beta)$ in L_i primarily by $\sigma(\alpha)$ and secondarily by $\sigma(\beta)$.

For each cluster G_i, loading a cluster G_j in memory takes $O(|G_i|/B) = O(B)$ I/Os; loading its boundary ∂G_j takes $O(1)$ I/Os per boundary set of ∂G_j (since each boundary set has size $O(B)$). Because G has bounded degree, each boundary set can be boundary to $O(1)$ clusters. Thus for each cluster G_i, loading all clusters G_j and their boundaries takes $O(N/B)$ I/Os. Retrieving the distances from a vertex in ∂G_i to all vertices in ∂G_j from the list L_i takes $O(1)$ I/Os per boundary set of ∂G_j since distances to the same boundary set are stored consecutively in L_i. For each vertex in ∂G_i, summed over all clusters G_j, each boundary set is accessed $O(1)$ times, in total $O(N/B^2)$ I/Os. Thus for each cluster G_i, retrieving the distances from ∂G_i to ∂G_j from L_i summed over all clusters G_j takes $O(B \cdot N/B^2) = O(N/B)$ I/Os. Once clusters $\overline{G_i}$, $\overline{G_j}$ and the $|\partial G_i| \cdot |\partial G_j| = O(B^2)$ distances from ∂G_i to ∂G_j are loaded in main memory we can compute the shortest paths $d(u,v) = \min_{\alpha \in \partial G_i, \beta \in \partial G_j} \{\delta_{\overline{G_i}}(u,\alpha) + \delta_{G^R}(\alpha,\beta) + \delta_{\overline{G_j}}(\beta,v)\}$ from vertices $u \in \overline{G_i}$ to vertices $v \in \overline{G_j}$ without further I/Os using a standard internal memory APSP algorithm. There are $|\overline{G_i}| \cdot |\overline{G_j}| = O(B^4)$ distances computed from cluster $\overline{G_i}$ to a cluster $\overline{G_j}$. For each cluster G_i, summed over all clusters G_j, there are $O(B^4 \cdot N/B^2) = O(B^2 \cdot N)$ distances computed in total. Assume that as we compute the output distances from $\overline{G_i}$ to $\overline{G_j}$ we write them to a list L_{ij} with $O(\text{scan}(B^4))$ I/Os; for each cluster G_i this takes $O(\text{scan}(B^2 \cdot N))$ I/Os in total.

Thus, for each cluster G_i, computing the shortest paths from vertices in $\overline{G_i}$ to all vertices in G takes $O(B \cdot sort(N) + N/B) = O(B \cdot sort(N))$ I/Os and writing the output distances to lists L_{ij} takes $O(\text{scan}(B^2 \cdot N))$ I/Os. Summed over all clusters G_i the computation of APSP takes $O(N/B^2 \cdot (B \cdot \text{sort}(N) + \text{scan}(B^2 \cdot N))) = O(\text{sort}(N^2)/B + \text{scan}(N^2)) = O(\text{scan}(N^2))$ I/Os. It is interesting to note that

computing APSP is faster than *outputting* (writing) the shortest paths since $O(\text{sort}(N^2)/B) \ll \text{scan}(N^2)$. We have:

Theorem 4. *APSP in a a planar digraph can be computed using* $O(scan(N^2))$ *I/Os and* $O(N^2)$ *space.*

The planar APSP algorithm can be used to compute the diameter of the graph. The difference is that to compute the diameter it is not necessary to output *all* shortest paths, but only keep track of the maximum distance encountered so far. As a result the diameter algorithm does not incur the $O(\text{scan}(N^2))$ I/O-cost and $O(N^2)$ space-cost of writing all paths.

Theorem 5. *The diameter of a planar digraph can be computed using* $O(N)$ *space and* $O(sort(N^2)/B)$ *I/Os.*

References

1. A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM,* 31(9):1116–1127, 1988.
2. L. Arge, G. S. Brodal, and L. Toma. On external memory MST, SSSP and multi-way planar graph separation. In *Proc. SWAT, LNCS 1851,* pages 433–447, 2000.
3. L. Arge, L. Toma, and N. Zeh. I/O-efficient topological sorting of planar DAGs. In *Proc. ACM Symposium on Parallel Algorithms and Architectures,* 2003.
4. A. L. Buchsbaum, M. Goldwasser, S. Venkatasubramanian, and J. R. Westbrook. On external memory graph traversal. In *Proc. SODA,* pages 859–860, 2000.
5. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proc. SODA,* pages 139–149, 1995.
6. E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik,* 1969.
7. G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing,* 16:1004–1022, 1987.
8. V. Kumar and E. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proc. SPDP,* pages 169–177, 1996.
9. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Math.,* 36:177–189, 1979.
10. A. Maheshwari and N. Zeh. I/O-optimal algorithms for planar graphs using separators. In *Proc. SODA 2002,* pages 372–381. ACM–SIAM, 2002.
11. K. Mehlhorn and U. Meyer. External-memory breadth-first search with sublinear I/O. In *Proc. ESA 2002,* volume 2461 of *LNCS,* pages 723–735. Springer, 2002.
12. U. Meyer, P. Sanders, and J. F. Sibeyn, editors. *Algorithms for Memory Hierarchies,* volume 2625 of *LNCS.* Springer, 2003.
13. U. Meyer and N. Zeh. I/O-efficient undirected shortest paths. In *Proc. ESA 2003,* volume 2832 of *LNCS,* pages 434–445. Springer, 2003.
14. K. Munagala and A. Ranade. I/O-complexity of graph algorithms. In *Proc. SODA,* pages 687–694, 1999.
15. N. Zeh. I/O-efficient graph algorithms. In *Proc. EFF summer school on massive data sets,* LNCS. Springer, 2004, to appear.
16. U. Zwick. Exact and approximate distances in graphs - a survey. In *Proc. ESA 2001,* number 2161 in LNCS, pages 33–48. Springer, 2001.
17. U. Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM,* 49:289–317, 2002.

A λ-Calculus for Resource Separation

Robert Atkey

LFCS, School of Informatics, University of Edinburgh
Mayfield Rd, Edinburgh EH9 3JZ, UK
bob.atkey@ed.ac.uk

Abstract. We present a typed λ-calculus for recording resource separation constraints between terms. The calculus contains a novel way of manipulating nested multi-place contexts augmented with constraints, allowing a concise presentation of the typing rules. It is an extension of the affine αλ-calculus. We give a semantics based on sets indexed by resources, and show how the calculus may be extended to handle non-symmetric relations with application to allowable information flow. Finally, we mention some future directions and questions we have about the calculus.

1 Introduction

Functional programming languages present the programmer with the neat abstraction that they are dealing with pure values. The programmer is lead into the comfortable illusion that these values have no physical presence, that they may be created and discarded as one creates and discards thoughts. However, on a real computer these values occupy memory space. Different values may share sections of memory, potentially inhibiting techniques which speed up functional code by using imperative techniques [5,7], see also the benign sharing condition in [6]. The same information can often be useful for the programmer in imperative languages for reasoning about aliasing [15,10].

This paper presents a system of typed λ-calculus, λ_{sep}, which attempts to record in the typing judgements the separation between the resources used by the values of the system. We adapt the techniques used by other substructural type systems such as the αλ-calculus [9,12] and Reynolds' SCI [15,16] to record and enforce the separation required between the values of the system.

The recording of separation constraints allows us to give the basic operations of the language types that enforce the constraints. For example, consider an application where we construct jobs to be run on two items of data in parallel, but we require that the items of data occupy separate regions of memory to allow for temporarily destructive operations:

$$\texttt{mkJob} : [1\#2](D, D) \longrightarrow J$$

The notation [1#2] expresses that `mkJob` takes two arguments which must occupy separate regions of memory. We call such specifications *separation relations.*

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 158–170, 2004.

To incorporate separation relations in the typing judgements we adopt a strategy, inspired by that of the $\alpha\lambda$-calculus, of introducing new ways of forming contexts. We no longer think of the context as a list or set of type assignments. Rather, we now regard the context as an undirected graph of type assignments, with edges recording the required separation between members. To allow the piecemeal construction of larger contexts, we also consider sub-contexts that have a uniform relationship to the rest of the context. The allowable manipulations on contexts, the structural rules, correspond to separation constraint preserving manipulations. The context also represents the union of all the resources occupied by the term.

The $\alpha\lambda$-calculus uses two different context formers, represented by the comma and semicolon. Both are binary constructors used to construct contexts from nested "bunches" of type assignments. The two constructors obey different structural rules; the comma disallowing everything except reordering, and the semicolon allowing the full range of intuitionistic structural rules. The two constructors may then given different semantics; a common one is that the comma combines two contexts which use separate resources, the semicolon combines two contexts which may use overlapping resources. In this way, the system can express relationships between objects. The system presented here, $\lambda_{\mathbf{sep}}$, generalises this situation to n places with attached binary relations expressing separation constraints between members. An example typing judgement is:

$$[1\#2, 1\#3](a : D, b : D, c : D) \vdash (\mathtt{mkJob}(a, b), \mathtt{mkJob}(a, c)) : (J, J) \qquad (1)$$

The separation between a and b and a and c recorded in the context is induced by the separation required by `mkJob`. This separation configuration can be expressed in the $\alpha\lambda$-calculus as $(a : D, (b : D; c : D))$.

Our extension to n-place separation relations on contexts, rather than binary bunches of contexts, is justified by looking at the possible graphs of separation between members expressible by both schemes. Obviously, any bunched context may be translated into an n-place context by writing out all the induced separations in full. Conversely, however, binary context formers only allow the expression of the series-parallel graphs; graphs that are constructed recursively from the one point graph by two operations of either complete non-separation, or complete separation. The following fact [2,18] shows that this does not cover all graphs:

Fact 1 (SP-graph Characterisation) *A graph is series-parallel iff its restriction to any four vertices is not equal to* $(\{a, b, c, d\}, \{(a, b), (b, c), (c, d)\})$.

To see an example of how this separation relation may occur, consider the context required for typing the construction of 3 jobs in sequence over 4 items of data:

$$[1\#2, 2\#3, 3\#4](a : D, b : D, c : D, d : D) \vdash (\mathtt{mkJob}(a, b), \mathtt{mkJob}(b, c), \mathtt{mkJob}(c, d))$$

Another example of this configuration is shown in section 4.

As mentioned above, the structural rules of the calculus correspond to separation preserving manipulations of the context. To allow concise presentation of

complex separation relations we permit nested contexts. For example, judgement (1) may also be written as:

$$[1\#2](a : D, [](b : D, c : D)) \vdash (\mathtt{mkJob}(a, b), \mathtt{mkJob}(a, c)) : (J, J) \tag{2}$$

Since a was required to be separate from both b and c, we may group b and c into a nested context with uniform separation from a. The equivalence of these two contexts is justified by considering that an object is separate from a group of objects if and only if it is separate from them all individually. This operation is encoded in the type system by the formalisation of substitution of separation relations into separation relations.

To preserve the correctness of separation relations when variables are used multiple times in a term we restrict the use of the rule of contraction. We only allow contraction between two variables when they are not required to be separate. Hence, given judgement (2) we may infer:

$$[1\#2](a : D, b : D) \vdash (\mathtt{mkJob}(a, b), \mathtt{mkJob}(a, b)) : (J, J) \tag{3}$$

This step is justified by thinking of contraction as semantically inducing the duplication of references to resources. Obviously two references to the same resource cannot be considered separate, so the typing judgement may not rely on them being so.

2 The Type System

Separation Relations. We first introduce separation relations; these represent the relationships of relative separation between objects.

Definition 1. *A* separation relation *of size n is a binary, irreflexive, symmetric relation on the set $\{0, \ldots, n-1\}$.*

We write $|\mathrm{S}|$ for the size of a separation relation. Define the relation $\mathrm{S} \subseteq \mathrm{S}'$ between two separation relations to hold if and only if $|\mathrm{S}| = |\mathrm{S}'|$ and, for all x and y, $x\mathrm{S}y$ implies $x\mathrm{S}'y$. We write specific relationship specifiers as lists of related pairs $[r_1, \ldots, r_k]_n$, where each r_i is of the form $x\#y$, denoting the related pairs of the relation, and n is the size.

Definition 2 (Relation Substitution). *For separation relations* S *and* S′*, with sizes n and n' respectively, define the operation of substitution* $\mathrm{S}\{\mathrm{S}'/i\}$*, where $0 \leq i < n$, as:*

$$(x, y) \in \mathrm{S}\{\mathrm{S}'/i\} \text{ iff } \begin{cases} (x - i, y - i) \in \mathrm{S}' & \text{if } \mathrm{norm}^i_{n'}(x) = \mathrm{norm}^i_{n'}(y) = i \\ (\mathrm{norm}^i_{n'}(x), \mathrm{norm}^i_{n'}(y)) \in \mathrm{S} & \text{otherwise} \end{cases}$$

where:

$$\mathrm{norm}^i_{n'}(x) = \begin{cases} x & \text{if } x < i \\ i & \text{if } i \leq x < i + n' \\ x - n' + 1 & \text{if } x \geq i + n' \end{cases}$$

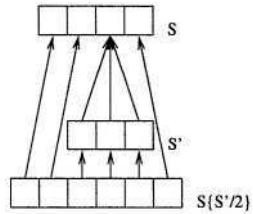


Fig. 1. Substitution of Separation Relations

Substitution of relations may be visualised as in figure 1. For a pair of positions x and y in $S\{S'/i\}$, either both x and y are in the range of S', or at least one of them is in the range of S. In the first case we use the relation S'; otherwise, we map the positions back to S (up the diagram) and use S to judge whether x and y are related. The function $\mathbf{norm}_n^i$ does the mapping back to S. Note that if a member of S is related to any member of S' then it is related to all of them.

Lemma 1. *The following properties hold of substitution:*

1. $S\{S'/i\}$ *is a separation relation;*
2. $S\{[]_1/i\} = S$;
3. $S\{S_1/i\}\{S_2/j + n_1 - 1\} = S\{S_2/j\}\{S_1/i\}$, *where* $i < j$ *and* $|S_1| = n_1$;
4. $S\{S_1\{S_2/j\}/i\} = S\{S_1/i\}\{S_2/i + j\}$.

Properties 3 and 4 ensure that if we perform two non-interfering substitutions in two different orders then we always finish in the same state. This is useful for reasoning about the allowable manipulations of contexts, since a nested context may always be substituted out to a single flat context. The 0-place separation relation $[]_0$ acts as a unit under substitution, since substituting $[]_0$ into a position i effectively causes that position to be removed.

Types and Contexts. The types of the calculus are generated by the following grammar, given a set of primitive types $\mathcal{T}$:

$$A, B \quad ::= \quad X \in \mathcal{T} \quad | \quad A_1, \ldots, A_n \xrightarrow{S} B \quad | \quad S(A_1, \ldots, A_n)$$

where S is a separation relation of size $n + 1$ for function types and size n for tuple types. The extra place in the function types represents the resources used by the body of the function. The types then generate the contexts, using the nested structure described in the introduction:

$$\Gamma, \Delta \quad ::= \quad x : A \quad | \quad S(\Gamma_1, \ldots, \Gamma_n)$$

where S is a separation relation of size n, A is a type and no variable x appears more than once in a context. We identify the context $x : A$ with the one-place context $[]_1(x : A)$. We define $\mathbf{v}(\Gamma)$ to be the list of variables in Γ built from a depth-first, left-to-right traversal. The notation $\Gamma(-)$ represents a context with a

$$\frac{}{x : A \vdash x : A}\ (\textsc{Id}) \qquad \frac{\Gamma(\Delta') \vdash e : A \qquad \Delta \overset{\phi}{\Rightarrow} \Delta'}{\Gamma(\Delta) \vdash \phi(e) : A}\ (\textsc{Struct})$$

$$\frac{\Gamma_1 \vdash e_1 : A_1 \qquad \ldots \qquad \Gamma_n \vdash e_n : A_n}{\mathrm{S}(\Gamma_1, \ldots, \Gamma_n) \vdash \mathrm{S}(e_1, \ldots, e_n) : \mathrm{S}(A_1, \ldots, A_n)}\ (\text{S-I})$$

$$\frac{\Gamma \vdash e_1 : \mathrm{S}(A_1, \ldots, A_n) \qquad \Delta(\mathrm{S}(x_1 : A_1, \ldots, x_n : A_n)) \vdash e_2 : B}{\Delta(\Gamma) \vdash \text{let } \mathrm{S}(x_1, \ldots, x_n) = e_1 \text{ in } e_2 : B}\ (\text{S-E})$$

$$\frac{\mathrm{S}(\Gamma, x_1 : A_1, \ldots, x_n : A_n) \vdash e : B}{\Gamma \vdash \lambda^{\mathrm{S}}(x_1, \ldots, x_n).e : A_1, \ldots, A_n \xrightarrow{\mathrm{S}} B}\ (\to\text{-I})$$

$$\frac{\Gamma \vdash f : A_1, \ldots, A_n \xrightarrow{\mathrm{S}} B \qquad \text{for } 1 \leq i \leq n.\ \ \Delta_i \vdash a_i : A_i}{\mathrm{S}(\Gamma, \Delta_1, \ldots, \Delta_n) \vdash f@_{\mathrm{S}}(a_1, \ldots, a_n) : B}\ (\to\text{-E})$$

$$\frac{\Gamma \vdash e : A \qquad f : A \longrightarrow B \in \mathcal{F}}{\Gamma \vdash fe : B}\ (\textsc{Prim})$$

Fig. 2. Typing Rules

"hole" at some position in the tree, and $\Gamma(\Delta)$ is the context with Δ substituted for the hole. This will be used for selecting nested sub-contexts for particular attention. We write $\Gamma \equiv_\alpha \Gamma'$ to denote equivalence of contexts up to renaming of variables.

A context Γ determines a separation relation, S_Γ, on all the variables it contains, by substituting out all the nested separation relations. It is unique by Lemma 1.

Structural Rules. We give the structural rules of the calculus in a uniform fashion, collected in the typing rule STRUCT in figure 2. This rule is parameterised by labelled transitions $\Delta \overset{\phi}{\Rightarrow} \Delta'$, where Δ and Δ' are contexts and ϕ is an operation on terms. The allowable transitions are shown in figure 3.

Transition	Effect on the context	$\phi(e)$
FLATTEN	$\mathrm{S}(\vec{\Gamma}, \mathrm{S}'(\vec{\Delta}), \vec{\Gamma'}) \Leftrightarrow \mathrm{S}\{\mathrm{S}'/\mathrm{i}\}(\vec{\Gamma}, \vec{\Delta}, \vec{\Gamma'})$	e
S-WEAK ($\mathrm{S}' \subseteq \mathrm{S}$)	$\mathrm{S}(\vec{\Gamma}) \Rightarrow \mathrm{S}'(\vec{\Gamma})$	e
CONTR ($\Gamma \equiv_\alpha \Gamma'$)	$\Gamma \Rightarrow []_2(\Gamma, \Gamma')$	$e[\mathrm{v}(\Gamma)/\mathrm{v}(\Gamma')]$
WEAK	$\Gamma \Rightarrow []_0()$	e
PERM	$\mathrm{S}(\vec{\Gamma}) \Leftrightarrow \sigma\mathrm{S}(\sigma\vec{\Gamma})$	e

Fig. 3. Structural Transitions

We can justify the structural rules by appeal to the properties of separation. The FLATTEN and CONTR transitions were justified in the introduction, here they appear in their general form. The transition S-WEAK is justified by observing that (reading the STRUCT rule from bottom to top) if we have a context which promises more separation than we require, then we may forget about the extra separation. Transitions WEAK and PERM are justified by the fact that we consider the underlying combination of values to be given by a normal product type.

Using these contexts and structural rules we can simulate the bunches of the $\alpha\lambda$-calculus. If we replace the context former "," with $[1\#2](-,-)$ and ";" with $[](-,-)$ we can rewrite an $\alpha\lambda$ context into a $\lambda_{\mathbf{sep}}$ context. The associativity of the two context formers is then a two-way derived rule formed from two applications of STRUCT with FLATTEN and its inverse ($S = []_2$ or $[1\#2]$):

$$\frac{\Gamma(S(\Delta_1, S(\Delta_2, \Delta_3))) \vdash e : A}{\Gamma(S(S(\Delta_1, \Delta_2), \Delta_3)) \vdash e : A}$$

Since we have S-WEAK and WEAK we are simulating the affine $\alpha\lambda$-calculus. Another useful derived rule is a generalised form of contraction:

$$\frac{\Gamma(x : A)(y : A) \vdash e : B \qquad \neg(x S_\Gamma y) \qquad \forall z. x S_\Gamma z \Leftrightarrow y S_\Gamma z}{\Gamma(x : A)() \vdash e[x/y] : B}$$

This is derived by completely flattening out the context $\Gamma(x : A)(y : A)$ and then grouping the pair x, y. This is possible since they have exactly the same relationship to all other members of the context. Then, since x and y are not required to be separate then we may apply CONTR. Another sequence of FLATTEN applications reconstructs the original context's structure.

Connective Rules. The typing rules for tuple and function types are also shown in figure 2. By reading the contexts as representing the resources used by the term we obtain an informal justification of the typing rules. The rule S-I uses the same relationship between the contexts on the left as for the terms on the right; therefore, if the free variables of the terms obey the required separation then so will the corresponding terms. The elimination rule for tuples, S-E, exploits the structure of the contexts. The position of the hole in $\Delta(-)$ indicates the relationships that the resources used by the variables x_i must have with the rest of Δ; by substituting Γ directly into this hole we are maintaining the same relationships.

The rules $\rightarrow$-I and $\rightarrow$-E can be understood similarly; in the introduction rule we have the nested sub-context Γ representing the resources used by the function body, treated as a single block. The required separation between the function's arguments and the function itself are recorded in S, which becomes part of the function's type. The relations are then reconstituted in the elimination rule. Using these function types we may simulate the function types of the $\alpha\lambda$-calculus: $A \rightarrow B$ becomes $A \xrightarrow{[]} B$ and $A \multimap B$ becomes $A \xrightarrow{[1\#2]} B$.

The PRIM rule incorporates a set of primitive operations $\mathcal{F}$ of the form $f : A \to B$ where A, B are types. We assume that primitive operations consume no resources themselves.

We have the usual admissible substitution rule for the calculus. We consider n-ary substitution both in order to have a strong enough induction hypothesis to handle the CONTR structural rule and because it is needed for the well-definedness of the equational theory.

Lemma 2 (Substitution). *The following rule is admissible:*

$$\frac{\Gamma(x_1 : A_1) \dots (x_n : A_n) \vdash e : B \qquad \Delta_1 \vdash e_1 : A_1 \qquad \dots \qquad \Delta_n \vdash e_n : A_n}{\Gamma(\Delta_1) \dots (\Delta_n) \vdash e[e_1/x_1, \dots, e_n/x_n] : B}$$

Definition 3 (Equational Theory). *Given a set of axioms of the form $\Gamma \vdash e_1 = e_2 : A$ such that $\Gamma \vdash e_i : A$, $i \in \{1, 2\}$, define the judgement of equation-in-context ($\Gamma \vdash e_1 = e_2 : A$) to be given by the axioms, plus: extensions of the usual $\beta\eta$ rules for linear tuple and function types; commuting conversions for tuple elimination; uniqueness for terms of type $[]_0()$; surjective pairing for $[]_2(-,-)$ types; and congruence, symmetry and transitivity rules.*

Proposition 1. *If $\Gamma \vdash e_1 = e_2 : A$ is derivable in the equational theory then $\Gamma \vdash e_i : A$, $i \in \{1, 2\}$.*

The proof of this proposition is mostly straightforward, given the substitution lemma, apart from the commuting conversion rules which, due to the syntax free structural rules, are treated using a variant of the structural extensions of [10].

3 Semantics

This section describes the semantics of the calculus. We first briefly mention the categorical semantics of the system. This fixes the structure we require and provides generic coherence and soundness results for models. The main body of the section covers a resource-indexed sets semantics which shows how the calculus models resources and their separation.

Categorical Semantics. We give the syntax a categorical semantics by requiring a category $\mathcal{C}$ with endofunctors $\underline{\mathrm{S}} : \mathcal{C}^{|\mathrm{S}|} \to \mathcal{C}$ for each separation relation S. These model the product types and contexts of the syntax. The structural rules are natural transformations between these functors, subject to several commuting conditions, corresponding to the (term syntax invisible) reordering of structural rules. Function types are modelled as functors, $[A_1, \dots, A_n \xrightarrow{S} -] : \mathcal{C} \to \mathcal{C}$, right adjoint to the $\underline{\mathrm{S}}$ functors. The semantics extends a map $\mathcal{I}$ of primitive types and operators to objects and arrows in $\mathcal{C}$ to a map $[\![-]\!]_{\mathcal{I}}$ from derivations to arrows in $\mathcal{C}$. We have the following results for the categorical semantics:

Theorem 1 (Coherence). *If* π_1 *and* π_2 *are two derivations of the judgement* $\Gamma \vdash e : A$ *then* $[\![\pi_1]\!]_{\mathcal{I}} = [\![\pi_2]\!]_{\mathcal{I}}$.

Theorem 2 (Soundness and Completeness). $\Gamma \vdash e_1 = e_2 : A$ *if and only if in all categorical models* $[\![\Gamma \vdash e_1 : A]\!]_{\mathcal{I}} = [\![\Gamma \vdash e_2 : A]\!]_{\mathcal{I}}$.

Resource indexed sets. We will model the types of the system as sets indexed by a partially ordered set of "resources". A binary relation on the resources provides our semantical interpretation of separation.

We start with a partially ordered set R that has finite joins $r_1 \vee \ldots \vee r_n$. The ordering represents the inclusion of small resources inside larger ones, and joins represent the combination of resources. To model separation we require a relation $-\#- \subseteq R \times R$ with the following properties:

1. $r_1 \# r_2$ iff $r_2 \# r_1$;
2. If $r_1 \# r_2$ and $r_1' \sqsubseteq r_1$ and $r_2' \sqsubseteq r_2$ then $r_1' \# r_2'$;
3. $r \# (r_1 \vee \ldots \vee r_n)$ iff $r \# r_1, \ldots, r \# r_n$.

Intuitively these properties are those of separation: separation is symmetric; if two resources are separate and we have two other resources contained in them then the two smaller resources are separate; and if a collection of resources is separate from a resource then they are all separate individually.

Types are modelled as functors from R to **Set,** the category of sets and functions. Terms are modelled as natural transformations between these functors. The constructions for tuple and function types are instances of Day's general construction for monoidal closed structures in functor categories [4].

Tuple and contexts $\mathrm{S}(A_1, \ldots, A_n)$ are modelled by the sets, at resource r:

$$\begin{array}{l}\{(a_1, \ldots, a_n) \in A_1 r \times \ldots \times A_n r : \\ \quad \exists r_1 \sqsubseteq r, \ldots, r_n \sqsubseteq r, a_1' \in A_1 r_1, \ldots, a_n' \in A_n r_n \text{ s.t.} \\ \quad\quad \forall i. a_i = A_i(r_i \sqsubseteq r) a_i' \text{ and } \forall i, j. i \mathrm{S} j \Rightarrow r_i \# r_j\}\end{array}$$

An element of a tuple is a tuple of elements $(a_1, \ldots, a_n)$; each one represents a value in its own resource r_i, projected forward into the containing resource r. The resources for each of the elements must be related as dictated by the separation relation. For natural transformations modelling terms $f_1, \ldots, f_n$, $\mathrm{S}(f_1, \ldots, f_n)$ is defined pointwise in the evident way.

The function types $A_1, \ldots, A_n \xrightarrow{\mathrm{S}} B$ are modelled at resource r_0 as the family of functions:

$$\Pi_{(r_1, \ldots, r_n) \in R} . A_1 r_1 \times \ldots \times A_n r_n \Rightarrow B(r_0 \vee r_1 \vee \ldots \vee r_n)$$

where $R = \{(r_1, \ldots, r_n) \in R : \forall i, j. i \mathrm{R} j \Rightarrow r_i \# r_j\}$ and $\Rightarrow$ is the set-theoretic function space. The resource r_0 represents the resources that the function itself occupies. Given resources $r_1, \ldots, r_n$ for the argument positions that satisfy the separation relation we get a function from arguments at these resources to the result using the combined resources of the function and all its arguments. This matches the justification of the typing rule using the contexts to represent the resources and their relationships.

Theorem 3. *The above definitions on the category* $[R, \mathbf{Set}]$ *for tuple and function types, together with the evident natural transformations for the structural rules, give a sound class of models for* $\lambda_{\mathbf{sep}}$.

Memory Regions. Our main example of the above construction is given by memory regions. Starting from some set of memory locations L, we take our set of resources to be the powerset of L and inclusion as the order. The relation is then defined as $r_1 \# r_2 \Leftrightarrow r_1 \cap r_2 = \emptyset$. Hence, two regions of memory are separate if they do not share any memory locations. It is easy to see that this relation obeys the required properties, and so $[\mathcal{P}(L), \mathbf{Set}]$ is a model of $\lambda_{\mathbf{sep}}$.

Representing Resources. This semantics suggests a useful way to extend the calculus via a simple application of the Yoneda embedding [8]. The Yoneda embedding allows us to represent resources $r \in R$ directly in the calculus as types Y_r. Separation with a value of type Y_r indicates separation from the fixed resource r. Following Yoneda we define:

$$Y_r(r') = \begin{cases} \{*\} & \text{if } r \sqsubseteq r' \\ \emptyset & \text{otherwise} \end{cases}$$

Thus Y_r is the empty type at resources inadequate for r and the singleton type at resources containing r.

Following on from the memory regions example, consider an example where we have a region k representing kernel memory in an operating system. Calls from the operating system kernel to user programs must not pass references to kernel memory, since it is inaccessible to user programs. This constraint may be typed as follows:

$$\texttt{callUserProgram} : [1 \# 2](Y_k, \mathit{Message}) \to \mathit{Result}$$

The representation of named resources in the calculus has a precedent in the *nominals* of hybrid logic. See, e.g. [1].

4 Non-symmetric Relations

A potentially useful variation of $\lambda_{\mathbf{sep}}$ is to allow non-symmetric relations in place of the separation relations. We can then model such constraints as allowable information flow or temporal ordering. We outline the changes required to the calculus and its semantics after a short example.

Take the base types to be *Src, Sink* and *Integer,* with primitive operations:

$$\texttt{read} : \mathit{Src} \to [](\mathit{Src}, \mathit{Integer}) \qquad \texttt{write} : [1 \triangleright 2](\mathit{Integer}, \mathit{Sink}) \to \mathit{Sink}$$

Thus, `read` takes a *Src* element and returns the next integer in that source, with the new state of the source; `write` takes a *Sink* and an integer, with the guarantee that the information in the integer may flow to the sink and returns

the new state of the sink. We write the relations with $\triangleright$s to indicate the lack of symmetry. A simple example judgement is:

$$\Gamma \vdash \text{let } [](a', i) = \texttt{read}(a) \text{ in } [](a', \texttt{write}[1 \triangleright 2](i, x)) : [](Src, Sink)$$

This types under $\Gamma = [1 \triangleright 2](a : Src, x : Sink)$, but not $\Gamma = [](a : Src, x : Sink)$ since the typing of `write` requires information to be able to flow from (the source of) the integer to the sink.

By manipulating the typing context we may set up networks of allowable information flow between sources and sinks. For example, programs that satisfy the typing judgement cannot pass data from a source to a sink that is not explicitly allowed by the context. This context sets up a network that allows information to flow from a to x and b to x and y, but nothing else:

$$[1 \triangleright 3, 2 \triangleright 3, 2 \triangleright 4](a : Src, b : Src, x : Sink, y : Sink) \vdash e : [](Src, Src, Sink, Sink)$$

Note that this network is not expressible using binary constraints since it takes the shape not expressible in series-parallel graphs (Fact 1).

The relaxation to non-symmetric constraints has almost no effect on the typing rules. However, our suggested applications of non-symmetric relations indicate an additional structural transition for the calculus; transitive closure of relations ($TC(S)$ is the transitive closure of S):

$$\textsc{TClosure} \quad | \quad \mathrm{S} \Rightarrow TC(\mathrm{S}) \quad | \quad \phi(e) = e$$

The resource indexed semantics is easily extended to the non-symmetric case by dropping the requirement of symmetry on the relation # between resources and adding a requirement of transitive closure.

An instance of the resource indexed sets semantics may be constructed as follows. Take a set Id of "identities" – people's names for instance – with a binary relation $\triangleright \subseteq Id \times Id$, denoting which people are allowed to talk to other people. This relation should be transitively closed. Extend the relation $\triangleright$ to sets of identities by $I \triangleright I'$ iff $\forall (i, i') \in I \times I'.(i \triangleright i')$. Now take $\mathcal{P}(Id)$ as the set of "resources" with this relation and union as the combining operation. By a variation of the above theorem, $[\mathcal{P}(Id), \mathbf{Set}]$ has the correct structure to model the calculus with non-symmetric relations and transitive closure.

To take this example further we wish to compare the effectiveness of this extension with type systems designed for secure information flow such as [17]. In particular, we have not considered the effect of control flow on the information flow; a large factor in type systems for security.

5 Conclusions

We have presented the calculus $\lambda_{\mathbf{sep}}$ and shown how the separation relations in the syntax may be interpreted as separation constraints on the resources used by values. We have shown how the calculus may be extended to deal with non-symmetric relations between values and how it can model properties such as allowable information flow.

The idea of augmenting contexts with a relation on the members has also been used in Retoré's Pomset Logic [14], an extension of linear logic. Pomset (Partially Ordered Multiset) logic extends linear logic by adding a "before" connective $A < B$, such that $A \otimes B \vdash A < B \vdash A \wp B$. This is interpreted, via proof nets, as being possible uni-directional communication, with $\otimes$ as no communication and $\wp$ as possible bi-directional commmunication. Retoré gives a coherence space semantics and a sequent calculus, but does not define n-ary tuple or implication formulae, nor does he consider nesting as a way of managing contexts. We believe that Retoré's coherence space semantics also works for a cut down variant of $\lambda_{\mathbf{sep}}$ with non-symmetric relations, but without contraction, weakening or function types. The "before" connective has also been considered by Reddy [13], da S. Corrêa, Haeusler and de Paiva [3] as a way of modelling temporal ordering in languages with state. da S. Corrêa *et. al.* also describe a semantics based on Dialectica Categories.

Reynolds' Syntactic Control of Interference (SCI) [15,16], a variant of Idealized Algol, also controls the aliasing of values in the system by disallowing contraction, except in the case of *passive* variables that do not update shared storage. O'Hearn [9] describes a form of SCI based on the $\alpha\lambda$-calculus.

We plan further work with this calculus and its semantics to answer some questions we have not yet been able to solve and to attempt to extend and apply this calculus in other areas. The first question is that of completeness for the resource-indexed sets semantics. It is easy to generalise the semantics to use a category of resources, and attempt to build such a category from the syntax. The primary problem is that there does not seem to be an obvious way to build the relation – # – from the syntax. The second question is that of conservativity over the $\alpha\lambda$-calculus; we conjecture this to be true, but have not yet found a proof.

We are also investigating several extensions of $\lambda_{\mathbf{sep}}$. Following SCI's passive types, we want to extend the calculus with *resource-insensitive* types. That is, types whose values do not occupy any resources and so separation constraints involving them have no meaning and can be added and removed arbitrarily.

We expect that the nested context structure of $\lambda_{\mathbf{sep}}$ will be useful for adapting the calculus to other uses. Having separation relations rather than binary context formers means that we maintain a linear ordering on the context where the $\alpha\lambda$-calculus would have to reorder context members to express some separation configurations. This means that the system should easily combine with ordered type systems for memory layout such as that of Petersen *et. al.* [11].

Lastly, we mention that the initial motivation for this calculus was to devise a higher-order version of the in-place update type system of Hofmann [5]. We have deviated from this goal in that it is not possible to directly express the separation of an object from *everything,* which is required to support in-place update. However, we note that one can express this using $\lambda_{\mathbf{sep}}$ using continuation passing style:

$$\mathtt{cons} : [1\#2, 1\#3, 2\#3, 3\#4](A, \mathit{List}(A), \diamond, \mathit{List}(A) \to R) \to R$$

where $\diamond$s represent memory locations.

Acknowledgements. I would like to thank David Aspinall and Michal Konečný for helpful discussions on the work presented here.

References

1. Carlos Areces and Patrick Blackburn. Bringing them all together. *Logic and Computation,* 11(5), 2001. Editorial of special issue on Hybrid Logics.
2. Denis Bechet, Philippe de Groote, and Christian Retoré. A complete axiomatisation for the inclusion of series-parallel partial orders. In *Proceedings of RTA '97,* volume 1232 of *Lecture Notes in Computer Science,* pages 230–240, 1997.
3. Marcelo da S. Corrêa, Edward H. Haeusler, and Valeria C. V. de Paiva. A dialectica model of state. In *CATS'96, Computing: The Australian Theory Symposium Proceedings,* January 1996.
4. B. J. Day. On closed categories of functors. In S. Mac Lane, editor, *Reports of the Midwest Category Seminar,* volume 137 of *Lecture Notes in Mathematics,* pages 1–38. Springer-Verlag, 1970.
5. Martin Hofmann. A type system for bounded space and functional in-place update. *Nordic Journal of Computing,* 7(4):258–289, 2000.
6. Martin Hofmann and Steffen Jost. Static prediction of heap space usage for first-order functional programs. In *Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages,* pages 185–197. ACM Press, 2003.
7. Michal Konečný. Functional in-place update with layered datatype sharing. In *Proceedings of TLCA 2003,* pages 195–210, 2003. LNCS 2701.
8. Saunders Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, 2nd edition, 1998.
9. P. W. O'Hearn. On bunched typing. *Journal of Functional Programming,* 13(4):747–796, 2003.
10. P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Syntactic control of interference revisited. *Theoretical Computer Science,* 228:211–252, 1999.
11. Leaf Petersen, Robert Harper, Karl Crary, and Frank Pfenning. A type theory for memory allocation and data layout. In G. Morrisett, editor, *Conference Record of the 30th Annual Symposium on Principles of Programming Languages (POPL'03),* pages 172–184, January 2003. ACM Press.
12. D. J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications,* volume 26 of *Applied Logic Series.* Kluwer Academic Publishers, 2002.
13. Uday Reddy. A linear logic model of state. Electronic manuscript: `http://www.cs.bham.ac.uk/~udr/`, October 1993.
14. Christian Retoré. Pomset logic: a non-commutative extension of classical linear logic. In *In proceedings of TLCA'97,* volume 1210 of *Lecture Notes in Computer Science,* pages 300–318, 1997.
15. John C. Reynolds. Syntactic control of interference. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages,* pages 39–46. ACM Press, 1978.
16. John C. Reynolds. Syntactic control of interference, part 2. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium,* pages 704–722. Springer-Verlag, 1989. Lecture Notes in Computer Science 372.

17. Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications,* 21(1):5–19, January 2003. Special issue on Formal Methods for Security.
18. J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series-parallel digraphs. *SIAM Journal of Computing,* 11(2):298–313, May 1982.

The Power of Verification for One-Parameter Agents*

Vincenzo Auletta, Roberto De Prisco, Paolo Penna, and Giuseppe Persiano

Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Università di Salerno, via S. Allende 2, I-84081 Baronissi (SA), Italy.
{auletta,robdep,penna,giuper}@dia.unisa.it

Abstract. We study combinatorial optimization problems involving one-parameter selfish agents considered by Archer and Tardos [FOCS 2001]. In particular, we show that, if agents can lie in one direction (that is they either overbid or underbid) then *any* (polynomial-time) c-approximation algorithm, for the optimization problem without selfish agents, can be turned into a (polynomial-time) $c(1+\epsilon)$-approximation truthful mechanism, for any $\epsilon > 0$. We then look at the $Q||C_{max}$ problem in the case of agents owning machines of different speeds. We consider the model in which payments are given to the agents only after the machines have completed the jobs assigned. This means that for each machine that receives at least one job, the mechanism can *verify* if the corresponding agent declared a greater speed. For this setting, we characterize the allocation algorithms A that admit a payment function P such that $M = (A, P)$ is a truthful mechanism. In addition, we give a $(1+\epsilon)$-approximation truthful mechanism for $Q||C_{max}$ when machine speeds are bounded by a constant. Finally, we consider the classical scheduling problem $Q||\sum w_j C_j$ which does not admit an exact mechanism if verification is not allowed. By contrast, we show that an exact mechanism for $Q||\sum w_j C_j$ exists when verification is allowed.

1 Introduction

Algorithms for solving optimization problems have been studied for decades in several models and the underlying hypothesis has been that the input is available to the algorithm (either from the beginning in off-line algorithms or during its execution in on-line algorithms). This assumption turns out to be unrealistic in the context of the Internet. Here, the various parts of the input are owned by *selfish* (but *rational*) agents and thus the optimization algorithm will have to ask the agents and then work on the *reported* inputs. It is realistic to assume that an agent will lie about her input if this leads to a solution X that she prefers even in spite of the fact that X is not globally optimal.

The field of mechanism design is the branch of Game Theory and Microeconomics that studies ways of inducing the agents to report their true type so

* Work supported by the European Project IST-2001-33135, Critical Resource Sharing for Cooperation in Complex Systems (CRESCCO).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 171–182, 2004.

that the optimization problem can be solved on the real input. We consider the following conceptual scenario with m agents identified by the integers $1, \cdots, m$. Each agent i has a private type t_i and a public valuation function v_i such that $v_i(X, t_i, \sigma)$ measures how much agent i likes solution X for instance σ. A *mechanism* is a pair $M = (A, P_A)$. A is an algorithm that, on input an instance σ and the reported types $b = (b_1, b_2, \ldots, b_n)$ (b_i is the type reported by agent i), computes a solution $X = A(b, \sigma)$. Moreover, the mechanism awards to each agent i a payment $P_A^i(b, \sigma)$, where $P_A = (P_A^1, \cdots, P_A^m)$. We define the *utility* (or *profit*) $u_M^i(b, \sigma | t_i)$ of agent i on instance σ, when the type of agent i is t_i and $b = (b_1, \cdots, b_m)$ are the declared values of the m agents in the following way

$$u_M^i(b, \sigma | t_i) := P_A^i(b, \sigma) + v^i(X, t_i, \sigma).$$

Each agent *knows* both algorithm A and payment function P_A^i and, being selfish and rational, naturally aims at maximizing her utility u^i. A mechanism is said *truthful with dominant strategies* (or simply *truthful*) if the payments P_A and the algorithm A guarantee that no agent obtains a larger utility when reporting $b_i \neq t_i$, independently of the other agents' reported types; that is, for all instances σ and for all reported types $b_{-i} = (b_1, \cdots, b_{i-1}, b_{i+1}, \cdots, b_m)$ of all the agents except i and for all possible declarations b_i of agent i it holds that

$$u_M^i((t_i, b_{-i}), \sigma | t_i) \geq u_M^i((b_i, b_{-i}), \sigma | t_i).$$

In what follows when M, σ, b_{-i} and t_i are clear from the context we will simply say $u^i(x)$ to denote the utility obtained by agent i by declaring x.

The celebrated truthful VCG mechanisms [6,9,5] are one of the classical results about mechanism design. Unfortunately, the VCG paradigm only applies to so-called *utilitarian* problems. The problem of scheduling jobs on related machines owned by selfish agents in order to minimize the makespan is a natural example of a non-utilitarian problem to which mechanism design theory can be applied. Here, the type t_i of agent i is the inverse of the speed s_i of her machine and a scheduling X has valuation $v^i(X, t_i, \sigma) = -w^i(X) \cdot t_i$ where $w^i(X)$ is the load assigned by scheduling X to machine i. The goal is to design a truthful mechanism that computes a scheduling with a good makespan. Scheduling is a special case of a large class of optimization problems for selfish agents called *one-parameter* agents. Efficient mechanisms for this class of problems have been provided by Archer and Tardos [1] that also characterized the class of allocation algorithms A that admit payments functions P for which (A, P) is a truthful mechanism. Essentially, truthful mechanisms for one-parameter agents must use so called *monotone* algorithms and, in this case, their payment functions are uniquely determined (up to an additive factor).

Summary of results. In this work, we consider mechanisms for one-parameter agents. We start by considering the case in which agents cannot lie arbitrarily, but either can only overbid (i.e., report $b_i \geq t_i$) or can only underbid (i.e., report $b_i \leq t_i$). We call such agents *restricted* agents. We prove that, in this case, for *every* algorithm A there exist payment functions P_A such that $M = (A, P_A)$ is a truthful mechanism (see Theorem 1). This is in contrast with the case of

(unrestricted) one-parameter selfish agents in which only *monotone* algorithms admit payments [1]. We also show that, under mild assumptions about the optimization problem (that, for example, hold for scheduling related machines), *any* c-approximation polynomial-time algorithm A and any $\epsilon > 0$, there exists a polynomial-time algorithm A_ϵ and polynomial-time computable functions $P_{A_\epsilon} = (P^1_{A_\epsilon}, \cdots, P^m_{A_\epsilon})$ such that A_ϵ is $c(1+\epsilon)$-approximate and $(A_\epsilon, P_{A,\epsilon})$ is a truthful mechanism (see Theorem 2).

We then consider *verifiable* agents, that is agents that may lie in reporting their types but the mechanism can verify whether agent i underbids, provided that the work $w^i(X)$ assigned to agent i by the solution X is positive. For scheduling, this naturally models the situation in which it is possible to verify that a machine has underbid (*i.e.,* it has declared to be faster than it actually is) only if at least one job has been assigned to it. This model has been studied by Nisan and Ronen [8] for the case of scheduling unrelated machines. We show that an algorithm A admits payments P_A so that (A, P_A) is truthful for verifiable agents *if and only if* A is *weakly monotone* (see Definition 3) and show that for polynomial-time weakly monotone algorithms the payment functions can be computed in polynomial time (see Theorem 3). This result tells us that the case of verifiable one-parameter selfish agents lies in between the case of restricted selfish agents (for which, by Theorem 2, all algorithms admit a payment function) and the case of (unrestricted) selfish agents in which only a subclass of the class of weakly monotone algorithms (called monotone algorithms and introduced in [1]) admit a payment function. Based on the characterization described above, in Section 4, we present a $(1+\epsilon)$-approximation truthful mechanism for $Q||C_{max}$ where machines are owned by verifiable agents with speeds bounded by a constant. The algorithm is polynomial for any number of machines. This result should be contrasted with the $(3+\epsilon)$-approximation randomized mechanism truthful in expectation [1] and the $(4+\epsilon)$-approximation deterministic truthful one [2] for *unverifiable* agents.

Finally, we consider the classical scheduling problem $Q||\sum w_j C_j$ which does not admit an exact mechanism in the general settings in which verification is not allowed [1]. By contrast, we prove the existence of an exact mechanism for $Q||\sum w_j C_j$ for verifiable agents. This shows that the class of monotone algorithms is a proper subclass of the class of weakly-monotone algorithms.

We would like to mention (and refer the reader to the full version for formal statements of the results and proofs) that similar results about truthful mechanisms with respect to *Bayesian-Nash* equilibrium can be obtained for *quasi one-parameter* agents (also studied in [3]) characterized by a valuation function $v^i(X) := -(w^i(X,\sigma)\cdot t_i + a^i(X, t_{-i}))$, where the additional term $a^i(X, t_{-i})$ does not depend on the type t_i of agent i.

Notation and model. Following the standard notation used in the study of approximation of combinatorial optimization problems (see, e.g., [4]), we consider problems defined as four-tuples $(\mathcal{I}, \mathsf{m}, \mathsf{sol}, \mathsf{goal})$, where $\mathcal{I}$ is the set of *instances* of the problem; $\mathsf{sol}(I)$ is the set of *feasible solutions* of instance I; $\mathsf{m}(X, I)$ is the *measure* of feasible solution X of instance I and goal is either min or max. Thus

the optimization problem consists in finding feasible solution X^* for instance I such that $\mathsf{m}(X^*, I) = \mathsf{opt}(I) := \mathsf{goal}_{X \in \mathsf{sol}(I)} \mathsf{m}(X, I)$.

We consider optimization problems involving *selfish* agents that *privately know* part of the input: every instance I is composed of a *private part* $t = (t_1, t_2, \ldots, t_n)$ (where t_i the *private information* of agent i) and of a *public part* σ. We assume that the set of feasible solutions $\mathsf{sol}(I)$ does not depend on t, that is, for every $I = (\sigma, t)$ and $I' = (\sigma, t')$, $\mathsf{sol}(I) = \mathsf{sol}(I')$.

2 Truthful Mechanisms for Restricted Selfish Agents

In this section we consider the problem of designing truthful mechanisms for selfish restricted agents.

Definition 1 (restricted selfish agent). *An agent i is* overbidding *(respectively,* underbidding*) if her reported type b_i always satisfies $b_i \geq t_i$ (respectively, $b_i \leq t_i$). An agent is* restricted *if it is overbidding or underbidding.*

A mechanism M is truthful for restricted selfish agents if, for every overbidding (respectively, underbidding) agent i, $u^i(t_i) \geq u^i(b_i)$, for any $b_i \in \mathcal{S}^i$ with $b_i \geq t_i$ (respectively, $b_i \leq t_i$).

Not to overburden our notation, when algorithm A, the instance σ and the declarations b_{-i} of all the agents other than i are clear from the context, we will simply write $P^i(x)$ to denote the payment awarded to agent i declaring x. Similarly, we write $w_i(x)$ instead of $w_A^i((x, b_i), \sigma)$.

We first show that any algorithm A admits a payment function P_A such that (A, P_A) is a truthful mechanism for restricted selfish agents. We give a method for computing the payment function and show that it runs in polynomial time if A is polynomial-time and for each i the set $\mathcal{S}^i$ of the possible strategies or agent i has polynomial size in the length of the input.

Theorem 1. *For any algorithm A there exists a payment function $P_A = (P_A^1, P_A^2, \ldots, P_A^n)$ such that $M = (A, P_A)$ is a truthful mechanism for restricted selfish one-parameter agents. The payment functions P_A^i for instance I can be computed in time polynomial in $|\mathcal{S}^i|$ and in the running time of algorithm A on inputs of length $|I|$.*

Proof. We prove the theorem for overbidding agents by explicitly defining the payment $P^i(x)$. Consider agent i and let her strategy set S^i be $(\alpha_1^i \leq \alpha_2^i \leq \cdots \leq \alpha_l^i)$. Fix the declarations $b_{-i} = (b_1, \cdots, b_{i-1}, b_{i+1}, \cdots, b_m)$ of the other agents. Since agent i is overbidding to enforce the truthfulness of the mechanism we have to impose that for every $b_i, t_i \in \mathcal{S}^i$, with $b_i \geq t_i$, it holds that

$$P^i(t_i) - w^i(t_i) \cdot t_i \geq P^i(b_i) - w^i(b_i) \cdot t_i.$$

Observe that if t_i is the largest value in S^i then the agent cannot lie and the above condition trivially holds for any payment. For example, we can set $P^i(\alpha_l^i) :=$

$w^i(\alpha_l^i) \cdot \alpha_l^i$ so to guarantee that the utility of agent i is non-negative. Then, for $k = l-1, l-2, \cdots, 1$, we recursively compute $P^i(\alpha_k^i) := w^i(\alpha_k^i) \cdot \alpha_k^i + \delta_k^i$, where

$$\delta_k^i := \max_{j \geq k} \{P^i(\alpha_j^i) - w^i(\alpha_j^i) \cdot \alpha_k^i\}.$$

Suppose now $t_i = \alpha_k^i$ and she declares $b_i = \alpha_j^i$, with $j > k$. By the definition of δ_k^i it follows that $P^i(b_i) - w^i(b_i) \cdot t_i \leq \delta_k^i = P^i(t_i) - w^i(t_i) \cdot t_i$. This proves the theorem.

In the sequel, we describe a class of problems such that, given a c- approximation algorithm A, a small perturbation in the input given to A has a small impact on the quality of the solution produced by the algorithm. Thus, we can consider algorithm A_ϵ that rounds the input of the problem and computes a $c(1+\epsilon)$-approximation solution. We show that if the sets $\mathcal{S}^i$'s are intervals of integers or consist of the reciprocals of an interval of integers, then we can define polynomial-time computable payment functions P_{A_ϵ} such that $(A_\epsilon, P_{A_\epsilon})$ is a truthful mechanism.

Definition 2. *Fix $\epsilon > 0$ and $\gamma > 1$. A minimization problem* $(\mathcal{I}, \mathsf{m}, \mathsf{sol}, \mathsf{goal})$ *is* (γ, ϵ)-smooth *if for any pair of instances $I = (t, \sigma)$ and $\tilde{I} = (\tilde{t}, \sigma)$ such that* $\max\{\frac{\tilde{t}_i}{t_i}, \frac{t_i}{\tilde{t}_i}\} \leq \gamma$, *for any $i = 1, 2, \cdots, m$, it holds that*

$$\forall X \in \mathsf{sol}(\sigma), \quad \mathsf{m}(X, (t, \sigma)) \leq (1+\epsilon) \cdot \mathsf{m}(X, (\tilde{t}, \sigma))),$$
$$\mathsf{opt}(t, \sigma) \geq \mathsf{opt}(\tilde{t}, \sigma).$$

A maximization problem is (γ, ϵ)-smooth if the above two inequalities hold with '$\leq$' and '$\geq$' exchanged.

The proof of the following lemma is straightforward.

Lemma 1. *Let Π be a (γ, ϵ)-smooth problem and let $I = (t, \sigma)$ and $\tilde{I} = (\tilde{t}, \sigma)$ be two instances of Π such that $\tilde{t}_i \leq \gamma t_i$, for $i = 1, 2, \ldots, m$. If X is a c-approximate solution for the instance $\tilde{I}$ then X is a $c(1+\epsilon)$-approximate solution for the instance I.*

Let Π be a (γ, ϵ)-smooth minimization problem and let A be an algorithm for Π. We define the mechanism $M_\epsilon = (A_\epsilon, P_{A_\epsilon})$ as follows. For each x let $\mathsf{up}(x) := \min\{y \geq x |\ y = \gamma^a, a \in \mathbf{N}\}$ and $\mathsf{down}(x) := \min\{y \in \mathcal{S}^i |\ \mathsf{up}(y) = \mathsf{up}(x)\}$. The algorithm A_ϵ, on input (b, σ) simply runs algorithm A on the input $(\mathsf{up}(b), \sigma)$.

Consider the set $\mathcal{S}^i$ of strategies of agent i and let $\tilde{\mathcal{S}}^i = \{\tilde{\alpha}_1^i \leq \tilde{\alpha}_2^i \leq \ldots \leq \tilde{\alpha}_h^i\}$ where $\tilde{\alpha}_j^i = \mathsf{up}(x)$ for some $x \in \mathcal{S}^i$. We next consider overbidding agents (underbidding agents can be treated similarly) and define the payment function $P_{A_\epsilon}^i(x)$ for all $x \in \tilde{\mathcal{S}}^i$ and, for $x \notin \tilde{\mathcal{S}}^i$, we set $P_{A_\epsilon}^i(x) := P_{A_\epsilon}^i(\mathsf{up}(x))$.

The values of P^i are defined recursively, starting from the largest value of $\tilde{\mathcal{S}}^i$ to the smallest one. For each $x \in \tilde{\mathcal{S}}^i$ we set

$$P_{A_\epsilon}^i(x) := \begin{cases} x \cdot w_{A_\epsilon}^i(x), & \text{if } x = \tilde{\alpha}_h^i; \\ \max\left\{P_{\mathsf{down}}^i(x), P_{\mathsf{up}}^i(x)\right\}, & \text{otherwise;} \end{cases} \tag{1}$$

where

$$P^i_{\mathsf{up}}(x) := \max_{z \in \tilde{\mathcal{S}}^i, z > x} \{P^i_{A_\epsilon}(z) + \Delta_{A_\epsilon}(x, z) \cdot \mathsf{up}(x)\} \tag{2}$$

$$P^i_{\mathsf{down}}(x) := \max_{z \in \tilde{\mathcal{S}}^i, z > x} \{P^i_{A_\epsilon}(z) + \Delta_{A_\epsilon}(x, z) \cdot \mathsf{down}(x)\}, \tag{3}$$

and

$$\Delta_{A_\epsilon}(x, y) := w^i_{A_\epsilon}(x) - w^i_{A_\epsilon}(y). \tag{4}$$

Next theorem states that $(A_\epsilon, P_{A_\epsilon})$ is an approximating truthful mechanism with respect to restricted agents.

Theorem 2. *Let Π be a (γ, ϵ)-smooth problem and let A be a c-approximate algorithm for Π. $M_\epsilon = (A_\epsilon, P_{A_\epsilon})$ is a $c(1+\epsilon)$-approximate truthful mechanisms with respect to restricted selfish agents for Π. Moreover, the payment functions P_{A_ϵ} can be computed in time $O\left(t_A(k) \cdot \sum_{i=1}^n \log^2 |\mathcal{S}^i|\right)$, where $t_A(k)$ is the worst-case running time of algorithm A on inputs of length k.*

We mention that it is possible to define a class of optimization problems for which we do not need to assume that sets $\mathcal{S}^i$ have finite size. Details are omitted from this extended abstract.

We also have the following corollary.

Corollary 1. *The mechanism $M_\epsilon = (A_\epsilon, P_{A_\epsilon})$ satisfies voluntary participation with respect to restricted overbidding agents.*

Discussion. Let us now briefly discuss the application of Theorems 1 and 2 to the problem of scheduling related machines (which is a special one-parameter problem). Here the set $\mathcal{S}^i$ consists of all speeds that agent i, the owner of the i-th machine, may declare and, in general, $\mathcal{S}^i$ coincides with the set of natural numbers. However, it is easy to see that for each machine i, there exists a threshold τ_i such that if machine i has speed τ_i or higher then the optimal schedule assigns all jobs to machine i. Thus, without loss of generality we can assume that $\mathcal{S}^i$ is the interval $[1, \tau_i]$. Then Theorem 1 tells us that *any* scheduling algorithm A can be equipped with payment functions P such that *(A, P)* is a truthful mechanism for restricted agents. The payment functions P can be computed in time polynomial in the τ_i's. Theorem 2 instead tells us that, if we round the speeds to the powers of $(1+\epsilon)$ then we lose an extra $(1+\epsilon)$ factor in approximation. But then there exist payment functions P' computable in *polynomial time* (in the length of the τ_i) that yield a truthful mechanism for restricted agents.

3 Truthful Mechanisms for Verifiable Machines

In this section we study the $\mathrm{Q}||\mathrm{C}_{\max}$ problem for the case in which each machine is owned by a selfish agent and machines are verifiable.

Let us quickly review the classical problem $\mathrm{Q}||\mathrm{C}_{\max}$. We are given m machines of speeds $s_1, s_2, \ldots, s_m$ and a set of n jobs of weights $J_1, J_2, \ldots, J_n$. We have to

assign each job to some machine, and a job of weight J_l requires J_l/s time units in order to be processed by a machine of speed s. We want to find an assignment of the jobs to the machines in order to minimize the makespan, that is, we want to minimize $\max_{1\le j\le m} w^j/s_j$, where w^j denotes the sum of the weights of all jobs assigned to machine j.

We consider the setting in which agent i owns machine i and knows the speed s_i of the machine and, for convenience, we consider $t_i = 1/s_i$ as the type of agent i. The public information is the set of weights $\sigma = (J_1, J_2, \ldots, J_m)$ of the jobs that need to be scheduled on the machines. The valuation given to a schedule of the jobs of σ by agent i is equal to the finish time of its machine. We assume that machines are *verifiable* and payments are provided *after* the execution of the jobs. If machine i receives at least one job (i.e., $w^i > 0$), then we can verify whether $b_i < t_i$ by checking the release time T_i of its last job.Indeed, if $b_i < T_i/w^i$ then the mechanism discovers that the agent has declared to be faster that it actually is. Since payments are provided *after* the execution of jobs and thus also depend on the actual time T_i, we can make it inconvenient to claim faster speeds then the real one. This is different from the previous case of restricted agents as, if a machine receives no job, there is no way of verifying the bid neither if he underbid nor if he overbid. Thus, in this case there are allocation algorithms for which no payment exists.

Next we characterize the class of algorithms A that admit a payment function P_A for which (A, P_A) is a truthful mechanism with respect to dominant strategies for *verifiable machines.*

Definition 3 (weakly monotone algorithm). *An algorithm for a scheduling problem is* weakly monotone *if, for every i, for every b_{-i} it holds that*

$$w_A^i(b_i, b_{-i}) = 0 \Rightarrow \forall b_i' > b_i, \quad w_A^i(b_i', b_{-i}) = 0. \tag{5}$$

Theorem 3. *An algorithm A for the scheduling problem Π admits a payment function P such that M = (A, P) is a truthful mechanism for verifiable machines if and only if A is weakly monotone.*

Proof. Assume that A is weakly monotone, let P' be payment functions such that $M' = (A, P')$ is a truthful mechanism for overbidding restricted agents that satisfy voluntary participation and define payment functions P in the following way.

$$P_i(b_i, T_i, b_{-i}) := \begin{cases} 0, & \text{if } w_A^i(b_i, b_{-i}) = 0 \text{ or } T_i > b_i w_A^i(b_i, b_{-i}); \\ P_i'(b_i, b_{-i}), & \text{otherwise;} \end{cases} \tag{6}$$

where T_i is the finish time of machine i. Notice that if T_i is greater than expected (that is, the bid b_i of agent i times the load assigned to it) the agent is punished and receives no payment. Let us now verify that *(A, P)* is truthful. We distinguish two cases.

1. $w_A^i(b_i) = 0$. If $b_i > t_i$, since A is weakly monotone, we have that $w_A^i(t_i) = w_A^i(b_i) = 0$. Eq. 6 implies $P_i(b_i, T_i) = P_i(t_i, T_i) = 0$. Thus the utility is the

same. If $b_i < t_i$, then $T_i > b_i w_A^i(b_i, b_{-i})$, thus the utility of the agent is 0 while by declaring $t_i$he would have a nonnegative utility.
2. $w_A^i(b_i) > 0$. If agent i reports $b_i < t_i$, then her utility is non-positive. If agent i reports $b_i \geq t_i$ it receives payment $P'(b_i)$ for a utility $u(b_i|t_i) = P'(b_i) - t_i w_A^i(b_i)$. On the other hand by reporting true type t_i, the agent would have had utility $u(t_i|t_i) = P'(t_i, b_{-i}) - t_i w_A^i(b_i, b_{-i})$. By the truthfulness of (A, P'), we have that $u(t_i|t_i) \geq u(b_i|t_i)$.
Assume now that there exists a payment function P such that *(A, P)* is truthful and, for the sake of contradiction, assume that there exist b_i and $b_i' > b_i$ such that $w_A^i(b_i) = 0$ and $w_A^i(b_i') > 0$. Since $w_A^i(b_i) = 0$, we have no way to verify whether $b_i < t_i$. Moreover, if $t_i < b_i'$, then agent i can make his/her machine finish at time $T_i' = w_A^i(b_i') \cdot b_i'$. In both cases, we cannot infer anything about the reported values b_i and b_i'. Since $M = (A, P)$ is truthful, then the following two conditions must be fulfilled:

$$t_i = b_i \;\Rightarrow\; P_i(b_i, 0) - w_A^i(b_i) \cdot b_i \geq P_i(b_i', T_i') - w_A^i(b_i') \cdot b_i,$$
$$t_i = b_i' \;\Rightarrow\; P_i(b_i', T_i') - w_A^i(b_i') \cdot b_i' \geq P_i(b_i, 0) - w_A^i(b_i) \cdot b_i'.$$

By using the fact that $w_A^i(b_i) = 0$ and combining the above two inequalities we obtain

$$\begin{aligned} P_i(b_i, 0) &\geq P_i(b_i', T_i') - w_A^i(b_i') \cdot b_i \\ &\geq P_i(b_i, 0) + w_A^i(b_i') \cdot b_i' - w_A^i(b_i') \cdot b_i = P_i(b_i, 0) + w_A^i(b_i')(b_i' - b_i) \end{aligned}$$

thus contradicting the hypothesis.

We mention (see final version for statement and proof) that, for verifiable machines, mechanism without payments perform very poorly.

4 A $(1 + \epsilon)$-Approximation Truthful Mechanism for $Q||C_{max}$

In this section we present a $(1+\epsilon)$-approximation polynomial-time truthful mechanism for $Q||C_{max}$ on selfish verifiable machines when speeds are integer and upper bounded by a constant S. Our mechanism is based on the well-known PTAS for $Q||C_{max}$ on uniform machines due to Hochbaum and Shmoys [7]. We modify their algorithm so to obtain $(1 + \epsilon)$-approximation also for machines of different speeds bounded by a constant and, more importantly, to guarantee weak monotonicity. This, combined with Theorem 3, implies the existence of a truthful polynomial-time $(1 + \epsilon)$-approximation mechanism for verifiable machines.

SCAN_ϵ takes as input (J, s) where $J = (J_1 \leq \ldots \leq J_n)$ are the jobs in nondecreasing order by weight and $s = (s_1 \leq \ldots \leq s_m)$ are the machine speeds, in nondecreasing order and performs the following steps.

1. $p := 1$;
2. if $\text{ORACLE}_\epsilon((1 + \epsilon)^p, J, s) = \text{fail}$ then
   while $\text{ORACLE}_\epsilon((1 + \epsilon)^p, J, s) = \text{fail}$ do $p := p + 1$

3. else while $\text{ORACLE}_\epsilon((1+\epsilon)^p, J, s) = \text{fail}$ do $p := p - 1$;
4. return $(\text{ORACLE}_\epsilon((1+\epsilon)^p, J, s), (1+\epsilon)^p)$

Algorithm ORACLE_ϵ is a polynomial-time algorithm that, for every $\epsilon > 0$, on input $C > 0$ and an instance (J, s) either returns success along with a feasible solution of cost at most $(1+\epsilon)C$ or it returns fail in which case no feasible solution exists of cost smaller than C.

Algorithm ORACLE_ϵ. Algorithm ORACLE_ϵ receives as input a bound C and an instance (J, s). The algorithm starts by partitioning the jobs J into two sets: the set J^S of the small jobs consisting of all the jobs of size less than ϵC and the set J^L of large jobs containing the remaining jobs. The first phase deals with the large jobs while the second phase schedules the small jobs.

Phase I: SCHEDULING LARGE JOBS. We round the weight of each job $J_h \in J^L$ to J'_h computed as the maximum value $\epsilon C(1+\epsilon)^p$ that is not greater than J_h.

Let $J' = (J'_1, \cdots, J'_n)$ be the sequence of rounded job weights, and let $\mathcal{W} = \{W_1, \cdots, W_k\}$ be the set of distinct values of J'. Denote by n_h the number of jobs in J' whose weight is W_h (clearly $\sum_{h=1}^{k} n_h = n$). We can represent the instance J' of rounded jobs by the k-tuple $(n_1, n_2, \ldots, n_k)$.

Jobs J' are then given in input to a dynamic programming algorithm EXACT that checks whether there exists a scheduling of makespan at most C and returns such a scheduling if it exists. Algorithm EXACT runs in time polynomial in the number of possible values of job weights. By the rounding performed on the weights of the large jobs the number of possible weights of the jobs is $O\left(\log_{1+\epsilon} \frac{s_m}{\epsilon}\right) = O(\log S)$, that is constant.

For $1 \le l \le m$ and for any tuple $(i_1, \ldots, i_k)$ such that $i_h \le n_h$, we define $\text{BINS}(C, i_1, i_2, \ldots, i_k, s, l)$ to take value 0 if there exists a scheduling of cost at most C for the set of jobs $(i_1, i_2, \ldots, i_k)$ using machines of speed $s_l, s_{l+1}, \ldots, s_m$; otherwise, $\text{BINS}(C, i_1, i_2, \ldots, i_k, s, l)$ equals to 1. Clearly, there exists a scheduling for the jobs J' of cost at most C if and only if $\text{BINS}(C, n_1, \ldots, n_k, s, 1) = 0$. To compute this value observe that $\text{BINS}(C, i_1, i_2, \ldots, i_k, s, m) = 0$ if and only if it is possible to execute all jobs $(i_1, i_2, \ldots, i_k)$ on machine m in time not greater than C. Instead, for $l < m$ we have

$$\text{BINS}(C, i_1, i_2, \ldots, i_k, s, l) = \min_{(q_1, q_2, \ldots, q_k)} \text{BINS}(C, i_1 - q_1, i_2 - q_2, \ldots, i_k - q_k, s, l+1)$$

where the minimum is taken over the set $\text{STORE}(C, s_l, i_1, \ldots, i_k)$ of tuples $Q = (q_1, \ldots, q_k)$ such that all jobs of Q can be executed on a machine of speed s_l and $q_h \le i_h$ for $h = 1, \ldots, k$. Observe that the cardinality of set $\text{STORE}(C, s_i, i_1, \ldots, i_k)$ is $O(n^k)$ and we can recursively compute $\text{BINS}(C, i_1, i_2, \ldots, i_k, s, 1)$ in time $O(mn^{2k})$. Moreover, algorithm EXACT also returns a scheduling of the large jobs of cost (with respect to the rounded weights) not greater than C. This is achieved by computing, for each k-tuple $(i_1, \ldots, i_k)$ and for each l, assignment $\text{X}(C, i_1, i_2, \ldots, i_k, s, l)$ to machine l, that is the lexicographically minimal k-tuple (if any) $(q_1, q_2, \ldots, q_k) \in \text{STORE}(C, s_l)$ such that $\text{BINS}(C, i_1 - q_1, i_2 - q_2, \ldots, i_k - q_k, s, l+1) = 0$.

Phase II: SCHEDULING SMALL JOBS. Let X^1 be the allocation of the large jobs computed by the algorithm EXACT and let $w_i(X^1)$ be the work assigned to

machine i by X^1 with respect to the real weights of the jobs. We have now to complete X^1 by allocating the small jobs $J^S = \{J_1, J_2, \ldots, J_k\}$ not considered in the previous phase. We assign job J_i to the machine that minimizes its completion time with respect to the work assigned to it by X^1 and by the allocation of the small jobs $J_1, \ldots J_{i-1}$. If there are more machines that obtain the same completion time we select the slowest one. If the schedule obtained at the end of this phase has cost greater than $C(1+\epsilon)$ (with respect to the real weights of the jobs) then ORACLE$_\epsilon$ returns fail. Otherwise, ORACLE$_\epsilon$ continues with the adjustment phase.

Phase III: ADJUSTMENT. In the final adjustment phase the algorithm partitions the machine into two sets: slow machines and fast machines. The aim of this phase is to enforce that each fast machine will receive positive load. The partition is computed in the following way. We assume without loss of generality that the number of jobs is larger than the number of machines and stress that jobs are ordered by nondecreasing weight and machines are ordered by nondecreasing speed. The first m jobs are assigned each to one machine with machine i getting the J_i. Suppose that the makespan of the one-to-one scheduling obtained is greater than $C(1+\epsilon)$. Then, it is easy to observe that, all schedules that assign positive load to all machine have cost greater than $C(1+\epsilon)$. Thus we repeat the same procedure by considering the $m-1$ fastest machines and the first $m-1$ jobs. The procedure stops when we find a one-to-one scheduling of jobs $J_1, \ldots, J_{m-d}$ to the machines of speed $s_{d+1}, \ldots, s_m$ of cost not greater than $C(1+\epsilon)$. The set of slow machines will then be the set of the first d machines and the remaining machines constitute the set of fast machines. By the discussion above, it is easy to see that any scheduling that assigns positive load to more than $m-d$ machines has cost greater than $C(1+\epsilon)$. More precisely, by denoting with $\mathsf{discard}(u, J, s)$ the number d of slow machines for an instance (J, s) and bound u we have the following lemma.

Lemma 2. *Let* $X \in \mathsf{sol}(J, s)$ *be a schedule that assigns positive load to more than* $m - \mathsf{discard}(u, J, s)$ *machines. Then* $\mathsf{m}(X, (J, s)) > u$.

The adjustment phase then continues in the following way. Let X^2 be the schedule computed at the end of Phase II and remove jobs $J_1, \ldots, J_{m-d}$ from X^2. For each $i = 1, \ldots, m-d$, let ℓ_i be the machine to which X^2 assigns job J_i. If machine $(d+i)$ has no load then J_i is assigned to this machine; otherwise J_i is re-assigned to machine ℓ_i. The schedule X^3 obtained at the end of this phase is then given in output by ORACLE$_\epsilon$.

Lemma 3. *If* ORACLE$_\epsilon(C, J, s) \neq$ fail *then the schedule output by* ORACLE$_\epsilon$ *assigns positive loads exactly to the* $m - \mathsf{discard}(C, J, s)$ *fastest machines.*

The next theorem proves that either ORACLE$_\epsilon$ gives a schedule of cost at most $C(1+\epsilon)$ or no schedule of cost less than C exists.

Theorem 4. *If* ORACLE$_\epsilon(C, J, s) =$ fail, *then* $\mathsf{opt}(J, s) \geq C$.
If, instead, ORACLE$_\epsilon(C, J, s)$ *returns a schedule* X *then* $\mathsf{m}(X, (J, s)) \leq C(1+\epsilon)$.

Next theorem proves that ORACLE$_\epsilon$ is stable.

Definition 4 (stable algorithm). *An algorithm A is stable if, for every* b_i, b_{-i} *such that* $w_A^i(b_i, b_{-i}) = 0$, *it holds that, for every* $b_i' > b_i$, $A(b_i', b_i) = A(b_i, b_{-i})$.

Theorem 5. *Algorithm* ORACLE_ϵ *is stable.*

We are now in a position to prove that algorithm SCAN_ϵ is weakly monotone.

Theorem 6. *The algorithm* SCAN_ϵ *is weakly monotone.*

Proof. Let (J, s) and (J, s') be two instances such that $s' = (s_i', s_{-i})$ and $s_i' < s_i$. Let (C, X) and (C', X') be the output of $\text{SCAN}_\epsilon(J, s)$ and $\text{SCAN}_\epsilon(J, s')$, respectively. We show that if $w^i(X) = 0$ then $w^i(X') = 0$. We consider three cases, depending on the values of C and C'.

Consider first the case $C = C'$. In this case both X and X' are schedules computed by algorithm ORACLE_ϵ with respect to the bound C. By Theorem 5 the algorithm ORACLE_ϵ i stable and thus if $w^i(X) = 0$ then $X = X'$ and in particular $w^i(X') = 0$.

Consider now the case $C > C'$. By Lemma 3 if $w^i(X) = 0$ then $i < \mathsf{discard}(C, J, s)$ and machine i has been classified as a slow machine. Since $\mathsf{discard}(C', J, s') \geq \mathsf{discard}(C, J, s)$ and the machine has declared a slower speed then it is classified as a slow machine also when ORACLE_ϵ runs on (C, J, s'). Then $w^i(X') = 0$.

Finally, consider the case $C < C'$. We observe that this case is not possible because it implies that $\text{ORACLE}_\epsilon(C, J, s')$ returns fail, while $\text{ORACLE}_\epsilon(C, J, s)$ returns the schedule X such that $w^i(X) = 0$. By Theorem 5 $\text{ORACLE}_\epsilon(C, J, s')$ has computed a schedule equal to X and thus it cannot return fail.

Finally we prove SCAN_ϵ is a $(1 + \epsilon)$ approximation scheme for $\mathrm{Q}||\mathrm{C}_{\max}$.

Theorem 7. *Let J be a set of jobs to be scheduled on* m *selfish machines of speed* s. *Then for every* $\epsilon > 0$, *there exists a* δ *such that the cost of the scheduling computed by algorithm* SCAN_δ *is at most* $(1 + \epsilon)\mathsf{opt}(J, s)$.

Proof. Let $C = (1 + \delta)^i$ be the value returned by the searching phase of the algorithm SCAN_δ and let X be the corresponding allocation of cost at most $C(1 + \delta)^2$. Moreover, since ORACLE_ϵ returned fail on input (C, J, s) we know that $\mathsf{opt}(J, s) \geq C$. Thus, we have that

$$\mathsf{m}(X, (J, s)) \leq (1 + \delta)^2 C \leq (1 + \delta)^2 \mathsf{opt}(J, s) \leq (1 + \epsilon)\mathsf{opt}(J, s),$$

by choosing $\delta = \epsilon/3$.

By combining Theorem 3 with Theorems 6-7, we obtain a family of polynomial-time truthful $(1 + \epsilon)$-approximation mechanisms. Moreover, from the proof of Theorem 3 and from Theorem 2, it follows that also the payments can be computed in polynomial time.

5 The Power of Verification

In this section we show that the ability of verifying the agents' bids leads to approximation guarantees strictly better than what can be achieved without verification for the problem of $Q||\sum w_j C_j$. In the $Q||\sum w_j C_j$, for each job l, we are given a weight w_l and a processing requirement J_l: this job requires J_l/s units of time when processed by a machine of speed s. In addition, jobs must be processed on the same machine without being interrupted. Thus, the completion time of each job C_l also depends on the order in which the machine processed the jobs assigned to it. The goal is to minimize the weighted sum of all jobs completion times $\sum_{l=1}^{n} w_l C_l$. In [1], it is proved that, for $c < 2/\sqrt{3}$, no c-approximation algorithm for $Q||\sum w_j C_j$ is monotone. Consequently, no truthful mechanism can obtain approximation better than $2/\sqrt{3}$. Next we show that any optimal algorithm for $Q||\sum w_j C_j$ is weakly monotone. Thus, by Theorem 3, if the speeds of the machines can be verified there exists an optimal truthful mechanism.

Theorem 8. *Any exact algorithm A* for* $Q||\sum w_j C_j$ *is weakly monotone.*

Proof. Consider two instances $\mathcal{I}$ and $\mathcal{I}'$ which differ only in the speed s_i and s_i' of the i-th machine and let X and X' be the assignments computed by A^* for the two instances. Assume by contradiction that $s_i > s_i'$ and that the load l_i assigned to machine i by X is 0 whereas X' assigns load $l_i' > 0$. Now observe that since $s_i > s_i'$, we have that $m(X',\mathcal{I}') > m(X',\mathcal{I})$ and, since X is optimal for $\mathcal{I}$, we have that $m(X',\mathcal{I}) \geq m(X,\mathcal{I})$. Finally, since $l_i = 0$, we have that $m(X,\mathcal{I}) = m(X,\mathcal{I}')$ which implies that $m(X',\mathcal{I}') > m(X,\mathcal{I})$ contradicting the optimality of A^*.

References

1. A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of the IEEE Symposium on Foundations of Computer Science,* pages 482–491, 2001.
2. V. Auletta, R. De Prisco, P. Penna, and G. Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. Technical report, To appear in Proceedings of STACS 2004.
3. V. Auletta, R. De Prisco, P. Penna, and G. Persiano. How to tax and route selfish unsplittable traffic. Technical report, To appear in Proceedings of SPAA, 2004.
4. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties.* Springer Verlag, 1999.
5. E.H. Clarke. Multipart Pricing of Public Goods. *Public Choice,* pages 17–33, 1971.
6. T. Groves. Incentive in Teams. *Econometrica,* 41:617–631, 1973.
7. D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. of ACM,* 34:144–162, 1987.
8. N. Nisan and A. Ronen. Algorithmic Mechanism Design. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing,* pages 129–140, 1999.
9. W. Vickrey. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance,* pages 8–37, 1961.

Group Spreading: A Protocol for Provably Secure Distributed Name Service

Baruch Awerbuch* and Christian Scheideler**

Department of Computer Science, Johns Hopkins University,
3400 N. Charles Street, Baltimore, MD 21218, USA,
{baruch,scheideler}@cs.jhu.edu

Abstract. This paper presents a method called Group Spreading that provides a scalable distributed name service that survives even massive Byzantine attacks. To accomplish this goal, this paper introduces a new methodology that essentially maintains a random distribution of all (honest and Byzantine) peers in an overlay network for *any* sequence of arrivals and departures of peers up to a certain rate, under a reasonable assumption that Byzantine peers are a sufficient minority. The random distribution allows to proactively protect the system from *any* adversarial attack within our model.

1 Introduction

The Internet was originally designed for the purpose of being extremely robust against hardware attacks, such as natural disasters or wars. However, software attacks (such as viruses, worms, or denial-of-service attacks) have become increasingly severe over the past few years, whereas hardware attacks are negligible. Thus, for any distributed application to run reliably on the Internet, it is of utmost importance that it is robust against adversarial software attacks. This is especially important for critical applications such as name service, i.e. a service that translates names such as "machine.cs.school.edu" into IP addresses so that machines can communicate with each other.

The current way name service is provided in the Internet is server-based. However, server-based architectures are vulnerable to attacks. A much more robust alternative appears to be the recently emerged peer-to-peer paradigm with its strictly decentralized approach. Unfortunately, despite the appeal of a decentralized approach, it appears to be a daunting task to develop peer-to-peer networks that are robust against adversarial attacks. Obviously, in an open environment any attempt to keep adversarial peers out of the network is doomed to failure because a priori there are no trust relationships that would allow to distinguish adversarial peers from honest peers. So one approach has been to at least limit the number of identities adversarial peers can obtain. Here, the use of a certification authority was suggested that requires credentials, sensitive

* Supported by NSF grant ANIR-0240551 and NSF grant CCR-0311795.
** Supported by NSF grant CCR-0311121 and NSF grant CCR-0311795.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 183–195, 2004.

information, or a payment to obtain an identity that allows the peer to join the system (e.g., [3]). However, being overly restrictive here would not only prevent adversarial peers but also many honest peers from joining the system, either because they cannot provide the necessary credentials or they are not willing to reveal sensitive information or to pay for their membership. Thus, it should be clear that without being overly restrictive, a certification authority will be ineffective in limiting the number of identities adversarial peers may obtain, allowing them to start so-called Sybil attacks [6] that can cause severe problems to all structured peer-to-peer systems that have been suggested so far. Hence, new designs are needed that provide reliability despite adversarial peers with a potentially unlimited number of identities.

The goal of this paper is to demonstrate that it is possible, under certain simplifying assumptions, to design *completely open* peer-to-peer systems that are *provably* robust against adversarial peers of *arbitrary behavior* with an *unlimited* number of identities, as long as the adversarial peers in the system (or more precisely, their currently active identities) are in a sufficient minority.

1.1 Distributed Name Service

A *peer* is defined as an entity with a unique identity, i.e. each peer p is uniquely identified by a tuple $(\mathsf{Name}(p), \mathsf{IP}(p))$ where $\mathsf{Name}(p)$ represents the name of p and $\mathsf{IP}(p)$ represents the IP address of p. In order to provide a distributed name service, the following operations have to be implemented:

- $\mathsf{Join}(p)$: peer p wants to join the system.
- $\mathsf{Leave}(p)$: peer p wants to leave the system.
- $\mathsf{Lookup}(\mathsf{Name})$: returns the IP address of the peer p in the system with $\mathsf{Name}(p) = \mathsf{Name}$, or NULL if there is no such peer.

These operations must be implemented so that they can be run *concurrently* and *reliably* in an *asynchronous* environment without any trust relationships in which adversarial peers have an unlimited number of identities at their disposal and behave in an arbitrary way (i.e. we allow Byzantine peers). To formalize this goal, we need a model (see also [1] for further details and motivation).

1.2 Security Model

We consider a peer to be *adversarial* if it belongs to an adversary or it is simply unreliable. Otherwise, a peer is called *honest.* We do not assume any prior trust relationships between the peers. Hence, a priori honest peers cannot be distinguished from adversarial peers.

Certification authority. A necessary requirement for a name service as defined above to work correctly is that every possible name x has at most one peer p with $\mathsf{Name}(p) = x$, i.e. the Lookup operation provides a unique peer for a given name (if such a peer is currently in the system). To guarantee this property,

an authority is needed that resolves conflicts among the peers and that prevents peers from taking over names of other peers. Thus, we assume that a certification authority is available that issues certified names to peers that want to enter the system and that only provides such a name if no peer has registered under that name before. Certified names allow peers to prove that they are the rightful owner of a name, which prevents peers from taking over the identities of other peers.

Semantics of Join, Leave, and Lookup. Join, Leave, and Lookup are operations acting on a name service relation $\mathsf{DNS} \subseteq \mathsf{Names} \times \mathsf{IPs}$ in the following way:

- $\mathsf{Join}(p)$: if this operation was initiated by $\mathsf{IP}(p)$ and p is correctly certified then $\mathsf{DNS} \leftarrow \mathsf{DNS} \cup \{(\mathsf{Name}(p), \mathsf{IP}(p))\}$
- $\mathsf{Leave}(p)$: if this operation was initiated by $\mathsf{IP}(p)$ then $\mathsf{DNS} \leftarrow \mathsf{DNS} \setminus \{(\mathsf{Name}(p), \mathsf{IP}(p))\}$
- $\mathsf{Lookup}(\mathsf{Name})$: if there is a peer q with $(\mathsf{Name}, \mathsf{IP}(q)) \in \mathsf{DNS}$ then return $\mathsf{IP}(q)$ and otherwise return NULL

Given that the certification authority maintains a mapping $\mathsf{CA} : \mathsf{Names} \rightarrow \mathsf{IPs}$ that is well-defined at any time (i.e. each name is associated with at most one IP address), also the lookup operation will be well-defined. Indeed, if the operations above are correctly implemented and executed, then $\mathsf{DNS} \subseteq \mathsf{CA}$ at any time and DNS consists of all identities currently present in the peer-to-peer system.

Notice that there are many ways for adversarial peers to attack the correctness of DNS: adversarial peers may execute $\mathsf{Join}(p)$ for honest peers currently not in the system or $\mathsf{Leave}(p)$ for honest peers currently in the system, or may leave the system without notice. Also, adversarial peers may attempt to provide a wrong answer to a lookup operation. So countermeasures have to be taken to protect the system against these attacks.

Network model. Our basic approach is to organize peers in a scalable overlay network in which every peer may be represented by multiple logical units called *nodes*. We allow arbitrary adversaries with bounded resources, i.e. the number of adversarial nodes is at most an ϵ-fraction of the honest nodes in the system at any time. Such adversaries are called *ϵ-bounded*.

We consider asynchronous systems in which every honest peer has the same clock speed but the clocks are not synchronized and there is no global time. Since honest peers are considered reliable, we assume that at any point in time, any message sent by an honest peer p to another honest peer q will arrive at q within a unit of time. (Other message transmissions may need any amount of time.) Furthermore, honest peers have unbounded bandwidth and computational power, i.e. an honest peer can receive, process, and send out an unbounded number of messages in a unit of time. The latter assumption allows us to ignore denial-of-service attacks, but it does *not* simplify the task of securing an overlay

network against legal attacks (i.e. attacks exploiting security holes in its protocols). As long as adversarial peers do not transmit unnecessary packets, the number of messages an honest peer will have to deal with in a time unit will normally be low so that we believe that our protocols are practical despite this assumption. Designing provably secure overlay networks for honest peers with bounded bandwidth is very challenging and needs further research.

Bootstrap peers. We assume that the certification authority provides a limited number of so-called *bootstrap peers* that are always part of the overlay network. This list of peers may be downloaded by a new peer when it registers its name so that it can contact one of the bootstrap peers without contacting the certification authority again. Bootstrap peers are like normal peers. For the Join protocol to work correctly we assume that at least one of the bootstrap peers is honest. Otherwise, there is no reliable way for a new peer to join the system. However, the Leave and Lookup protocols should *not* rely on the bootstrap peers so that the system is scalable and can work correctly under ϵ-bounded adversaries even if all bootstrap peers are adversarial.

In this paper, we will assume that *all* bootstrap peers are honest.

Messages. Finally, we need some assumptions about how messages are passed. We assume that the (IP address of the) source of a message cannot be forged so that adversarial peers cannot forge messages from honest peers (which can easily be achieved). Also, a message sent between honest peers cannot be deleted or altered by the adversary (because peers normally sit at the edge of the network).

1.3 Security Goal

Recall that our security goal is to implement the Join, Leave, and Lookup operations so that they can be run concurrently and reliably in an asynchronous environment. More precisely, any of these operations executed by any of the honest peers in the system should be executed in a correct and efficient way. "In the system", "correct" and "efficient" require precise definitions.

A Join(p) (resp. Leave(p)) operation is called *completed* if any Lookup(Name(p)) operation executed afterwards by an honest peer in the system (and before another Join(p) or Leave(p) operation) returns IP(p) (resp. NULL). A peer p is called *mature* if Join(p) has been completed and Leave(p) has not been initiated yet. A Lookup(Name) operation is called *completed* once the peer initiating the request accepts the return value.

An overlay network operation is said to execute *correctly* if it completes within a finite amount of time. Furthermore, an overlay network operation is called

- *work-efficient* if it is completed using at most polylog(N) messages and
- *time-efficient* if it is completed using at most polylog(N) time,

where N be the current number of nodes in the overlay network. The following definition is central to this paper.

Definition 1. *We call an overlay network* survivable *if, when starting with a consistent system of n honest nodes (i.e. there are no pending join or leave requests) and $N \geq n$ at any time afterwards, it can guarantee the correct and (time and work) efficient execution of any overlay network operation initiated by an honest peer for* $\mathsf{poly}(n)$ *time steps, with high probability, for any* 1/polylog(N)*-bounded adversary and a join/leave rate of up to* 1/polylog(N)*, i.e. at least* N/polylog(N) *peers may join or leave the network in a time unit.*

Notice that we only require correct and efficient executions for honest peers, i.e. we do not care whether the semantics of Join, Leave, or Lookup are violated for adversarial peers. For example, a Lookup(Name) request for some Name owned by an adversarial peer q is allowed to give inconsistent answers, i.e. some honest peers may receive the answer $\mathsf{IP}(q)$ and others may receive the answer NULL.

Also, notice that we have to add the term "with high probability" above, because we said that a priori, it is not possible to distinguish between honest and adversarial peers. So no absolute guarantees can be given, unless we completely interconnect all peers, which is highly inefficient and therefore out of question.

1.4 Existing Work

Classical distributed computing methods [12,4,13,16] use Byzantine agreement and two-phase commit approaches with inherently *linear* redundancy and overhead to maintain a consistent state.

The *proactive security* approach in [15,11,10,2,9] uses different coding techniques to protect unreliable data in *reliable* networks; applying these methods in our context still yields linear overhead.

Fixed topology networks as in [8], will work only for non-Byzantine peers, and only allow fail-stop faults; the construction cannot handle malicious behavior of even a few malicious players.

The reliability of *hash-based peer-to-peer overlays* (or DHT's) such as Chord [17], Pastry [7], and Tapestry [18] hinges on the assumption that the IDs given to the nodes are pseudo-random, so that they can cope with a constant fraction of the nodes failing concurrently, with only logarithmic overhead. While this may seem to perfectly defeat massive attacks under these randomness assumptions, DHT's cannot handle even small-scale adaptive adversarial attacks involving the selection of adversarial IP addresses (to get close to desired IDs). One such "Sybil" attack is described in [6]. Remarkably, the attackers do not need to do anything complex such as inverting the hash function; all that is needed is to get hold of a handful (actually, logarithmic) number of IP addresses so that IDs can be obtained that allow to disconnect some target from the rest of the system. This can be accomplished by a linear number (i.e. $O(n)$) of offline trial/errors. For similar attacks, see [5].

Random or unpredictable placement of data in a logarithmic size subset of locations (as in Freenet) ensures that data is difficult to attack, but also makes it *difficult to retrieve.* Specifically, data retrieval of randomly placed data requires a linear number of queries, which is, definitely unscalable.

Recently, an overlay network design for robust distributed name service has been suggested [1] that satisfies all criteria of survivability apart from work-efficiency; the work overhead can be close to linear.

2 Non-survivable Overlay Networks

In this section we prove that predictable overlay networks and hash-based overlay networks (i.e. networks in which the ID of a node is determined by a hash function) are not survivable. Furthermore, we show that being able to enforce a limited lifetime is crucial for the survivability of systems based on a virtual space, like hash-based systems.

2.1 Predictable Overlay Networks

An overlay network is *predictable* if for any fixed join/leave sequence of peers the topology will always be the same in a consistent state. Notice that all hash-based overlay networks with a fixed hash function are predictable.

We start this section by demonstrating that *no* predictable overlay network can be survivable under our definition of survivability.

Theorem 1. *Consider an arbitrary predictable overlay network of maximum (peer) degree d that can handle any sequence of N join/leave requests of peers in T time units. Then there is a join/leave sequence of $2N$ peers so that an ϵ-bounded adversary with $\epsilon \geq d/N$ can isolate an honest peer in O(T) steps.*

Proof. The proof is relatively easy. First, $2N$ honest peers join, and afterwards the first N peers that joined the network leave. This takes $O(T)$ time steps. Consider now any peer in the resulting network, say v, and let $w_1, \ldots, w_d$ be its neighbors. Then, consider the join/leave sequence of honest peers that is like the sequence above but without $w_1, \ldots, w_d$. Assign the join events for $w_1, \ldots, w_d$ to the adversary. Then we arrive at the situation that v is completely surrounded by adversarial peers. This sequence *always* works because the overlay network is predictable. Hence, the theorem follows. □

2.2 Hash-Based Overlay Networks

Hash-based overlay networks are vulnerable to adversarial attacks even if the hash function is chosen at random, and it is a one-way hash function. The mere fact that peers do not change their location over time turns them into "sitting ducks". To illustrate how an attack on hash-based approaches would look like, consider the Chord system.

Suppose that we have a system currently consisting of a set V of n nodes, each representing a peer, and further suppose we have a (pseudo-)random hash function $h : \mathsf{Names} \rightarrow [0,1)$ that maps nodes to real values in the [0,1) ring. The real value a node v is mapped to is called its *identification number* or ID

and denoted by $\mathsf{ID}(v)$. The basic structure of Chord is a doubly-linked cycle, the so-called *Chord ring,* in which all nodes are ordered according to their IDs. In addition to this, every node v has edges to nodes $f_i(v)$, called *fingers,* with $f_i(v) = \text{argmin}\{w \in V \mid \mathsf{ID}(w) \geq \mathsf{ID}(v) + 1/2^i\}$ for every $i \geq 1$ ($[0,1)$ is treated as a ring here).

Now, take any node v in Chord with hash value $x \in [0,1)$. By generating a set A of adversarial nodes with hash values in $[x - \epsilon, x]$, $[x, x + \epsilon]$, and $[x + 1/2^i, x + 1/2^i + \epsilon]$ for all relevant i where $\epsilon > 0$ is sufficiently small, v will have no node pointing to it any more, and all nodes v is pointing to belong to A, with high probability. Hence, the peer p of v will effectively be isolated from the rest of the system. Notice that even a relatively modest adversary can come up with such a set A, even if the hash function is not invertible. It just has to try enough values (which is easily possible with SHA-1; the fact that the hash values may depend on IP addresses is not a limitation, because with IPv6 there will be plenty of them available – even for private users). Also, notice that an adversary just has to know the IP address of p (to compute x and) to start an attack on p.

2.3 Problems with Unlimited Lifetime

Also truly random IDs do not help as long as no node can be excluded from the system against its will, even if there is a secure mechanism for enforcing such an ID on *every* node that joins the system.

All hash-based systems are based on the concept of a virtual space. The basic idea underlying these systems is that nodes are given virtual locations in some space, and the overlay network is constructed based on these virtual locations. That is, depending on its virtual location, a node aims to maintain connections to other virtual locations and does this by establishing pointers to the nodes closest to these locations. See, e.g., [14] for a general framework behind this approach.

Thus, all an adversary has to do to attack such a system is to throw new nodes into the system at a maximum possible rate and to keep only those nodes that obtain IDs in regions the adversary intends to take over. Hence, unlimited lifetime can result in a fast degradation of randomness.

3 Outline of the Group Spreading Protocol

Finally, we give an outline of the Group Spreading Protocol that avoids the problems above. The details can be found in a full paper.

3.1 Basic Approach

We start with some basic definitions. Recall that a *peer* is an entity with a unique *name* and a *node* is a logical unit in the system with a unique ID. A peer may have multiple nodes in the system. However, honest peers will limit their nodes to $O(\log N_t)$, where N_t denotes the number of honest nodes in the system at

time t. A node is called *honest* if it belongs to an honest peer. We assume that honest nodes execute our protocols in a faithful and reliable way. Adversarial nodes may do *anything*. (Recall that we only have to worry about legal attacks because honest nodes have infinite bandwidth.) A *region* is an interval of length $1/2^r$ in [0,1) for some integer $r \geq 0$ that starts at an integer multiple of $1/2^r$. The core ideas behind the Group Spreading protocol are:

1. every honest peer aims to maintain a group of $L_p = \Theta(\log N_t)$ nodes of consecutive remaining lifetimes from 1 to L_p time steps,
2. every honest node v maintains connections to all reliable nodes in all regions of size $1/2^{r_v} = \Theta(\log N_t)$ containing $\mathsf{ID}(v) \pm 1/2^i$ for some $i \geq 0$
3. the system enforces a random ID in [0,1) on every node, and
4. the system enforces a lifetime of $O(\log N_t)$ on every node.

The reason for item 1 is that Group Spreading uses a simple ID generation mechanism that enforces the selection of a random ID if it terminates. But this mechanism may not terminate if adversarial nodes are involved in it. Thus, every honest peer keeps a group of $\Theta(\log N_t)$ nodes in the system so that, with high probability, sufficiently many nodes of a peer will be in regions without a close-by adversarial node, and therefore the ID generation mechanism can terminate in these regions.

Using this approach, we can prove the following theorem.

Theorem 2. GROUP SPREADING *survives up to a* $\Theta(1/\log N)$ *fraction of adversarial nodes with* $O(\log N)$ *time and* $O(\text{polylog}N)$ *work per operation as long as the join/leave rate of honest nodes is* $O(1/\log N)$ *and the join rate of adversarial nodes is* $O(1/\log^2 N)$.

Next we sketch the proof of this theorem. We start with some notation that we will frequently use, followed by some basic assumptions. Afterwards, we sketch the protocols and their analysis.

3.2 Notation

- r_v: *range* of a node v, selected upon creation of v by a node w so that w's *view* (i.e. the nodes it knows) of the region of size $1/2^{r_v}$ containing $\mathsf{ID}(w)$ is as close as possible to $\rho(r_v - \log r_v)$ for some fixed constant ρ
- L_v: maximum lifetime of a node v, computed as $L_v = \lambda \cdot r_v$ for some fixed constant λ
- L_p: $(3/4) \min_{v \in p} L_v$
- $\hat{L} = \max_v L_v$ and $\ell = \min_v L_v$ over all nodes in the system
- $p = (\mathsf{Name}(p), \mathsf{IP}(p))$: represents a peer
- $v = (p(v), \mathsf{ID}(v), r_v)$: represents a node, where $p(v)$ is the peer owning v
- $\Gamma_{v,t}$: all nodes v is connected to at time t
- $R_i(v)$: the unique region of size $1/2^{r_v}$ containing $\mathsf{ID}(v) + \mathrm{sgn}(i)/2^{|i|}$
- $m = (\mathsf{Source}(m), \mathsf{Dest}(m))$: represents a message
- B: set of bootstrap peers

- A_t: nodes that are part of a join operation of a peer at time t
- D_t: nodes that are part of a leave operation of a peer at time t
- C_t: nodes whose creation is started at time t
- M_t: nodes that are mature at time t (i.e. their creation is completed)
- V_t: nodes that are *legal members* at time t (i.e. they have a connection to an honest node in the system)

Given a set of nodes S, S^h denotes the set of honest nodes in S and S^a denotes the set of adversarial nodes in S. So $N_t = |V_t^h|$. Furthermore, given a set of nodes S and a region $R \subseteq [0,1)$, $S(R)$ denotes the set of nodes in S with IDs in region R. Given a set S_t and a time interval I, $S_I = \bigcup_{t \in I} S_t$.

3.3 Prerequisites

There is a sufficiently small constant $\epsilon > 0$ so that for all $t \geq 0$,

P1 $B \subseteq V_t^h$,
P2 $|A_t^h \cup D_t^h| \leq \epsilon \cdot N_t / \log N_t$,
P3 $|A_t^a| \leq \epsilon \cdot N_t / \log^2 N_t$, and
P4 $|V_t^a| \leq \epsilon \cdot N_t / \log N_t$.

Suppose that the adversary has bounded resources (concerning computational cycles and bandwidth). Then, in practice, conditions P3 and P4 could be enforced by presenting computational challenges or Turing tests to new nodes that are created via bootstrap peers and by continuously checking connections to other nodes in the system. If a peer does not respond in time, its request for creating a node is ignored by the bootstrap peer, resp. the connection to the corresponding node is removed.

3.4 Creating a New Node

Suppose that a node u wants to create a new node v. Then u calls the Create operation, which does the following. (t denotes the current time step.)

1. ID generation stage:
   a) u sends an ID generation request to all nodes in $G = \Gamma_{u,t}(R_0(u))$, then waits for $L_u/3$ steps, and afterwards asks the nodes in G to compute the ID and send it to u. If u has not received the same ID x from $\geq |G|/5$ nodes within $O(1)$ steps, it aborts the protocol. Otherwise, u continues with the authorization stage.
   b) Each node $w \in G$ receiving an ID request, generates a random x_w and sends $h(x_w)$ (for some bit commitment scheme h) to all nodes in $R_0(u)$. Afterwards, it waits for $L_u/3 + O(1)$ steps during which it accepts commitments from other nodes till u sends an ID computation request to w. If w has not heard back from u by then, it aborts the protocol.

c) Each node $w \in G$ receiving an ID computation request from u waits until $L_u/3$ steps are over since it received the ID generation request and then sends x_w to all nodes in G. If w receives all $x_{w'}$ for all $h(x_{w'})$ it received before within $O(1)$ steps, it computes $x = \bigoplus_{w'} x_{w'}$ (including x_w) and sends x to u. Otherwise, it aborts the protocol.

2. Authorization stage:
   a) u computes r_v and sends an authorization request with (x, r_v) to all nodes in G.
   b) Each node $w \in G$ that sent u the same x $O(1)$ steps before, routes its view of $R_0(u)$ to the region R_x of size $1/2^{r_v}$ containing x, and each node w' in R_x routes its view of R_x back to the nodes in $R_0(u)$, giving a set $S_w \subseteq V_I(R_x)$ for w with $I = [t - 2\log N_t, t]$ if this process took at most $2\log N_t$ steps (otherwise, w aborts the protocol). If so, w sends an authorization to the nodes in S_w and forwards S_w to u.
   c) Once u receives sets S_w from $\geq 3|G|/20$ nodes in G, it computes the set $G' = \{w' \in \bigcup_w S_w : |\{w \mid w' \in S_w\}| \geq |G|/10\} \subseteq V_I(R_x)$.

3. Integration stage:
   a) v sends an integration request to all nodes in G'. Then it waits for $O(1)$ steps to make sure that all nodes relevant for v added v to their connection table. Afterwards, v sends an integration request to all nodes relevant for it.
   b) Each node w in R_x that was authorized by sufficiently many nodes in $R_0(u)$ at most $\log N_t + O(1)$ steps ago, notifies v about all nodes relevant for v and authorizes all nodes relevant for v to integrate v in their connection table.
   c) Each node w' relevant for v that receives sufficiently many authorizations from nodes in R_x and an integration request from u within $O(1)$ steps, adds u to its connection table.

Apart from the bootstrap nodes, every node is only allowed to initiate the Create protocol 3 times during its life.

3.5 Insert and Lookup Operations

The Insert and Lookup operations use the binary search method of Chord to forward requests, with the only difference that they are region-based, i.e. messages are forwarded along a sequence of regions rather than nodes. An honest node v accepts a message m only if $v \in V_i(R_i(m))$ for some $i > 0$ and m was sent to v by at least 1/5 of the nodes in $\Gamma_{t,v}(R_{i-1}(m))$, where $R_i(m)$ is the ith region on the path of m.

Each peer p in the system calls the Insert operation every $L_p/3$ steps to store $(\mathsf{Name}(p), \mathsf{IP}(p))$ in the unique region R_p of size $1/2^{r_v}$ for some node v of p that contains $h(\mathsf{Name}(p))$ for some one-way hash function $h : \mathsf{Names} \to [0,1)$. This makes sure that sufficiently many honest nodes in R_p know p at any time. Thus, when executing a Lookup operation for $\mathsf{Name}(p)$, it will return $\mathsf{IP}(p)$ as long as p is in the system.

3.6 Join and Leave Operations

When a peer p wants to join the system, it contacts some bootstrap peer q. q will then initiate 3 Create operations via one of its nodes in each step until p has L_p nodes with remaining lifetimes from 1 to L_p. Once this is done, p is mature. Afterwards, p will initiate 3 Create operations via one of its nodes in each step (using each node only once) to keep L_p nodes in the system.

Leaving is easy. The peer p simply does not create any new nodes and waits until all of its old nodes left the system.

3.7 Safety

The correctness of the operations crucially depends on whether the system is safe. That is, we require for all regions R with $|R| = (\gamma \log N_t)/N_t$ for a sufficiently large γ that

S1 $|V_t^h(R)| \in [(2/3)\gamma \log N_t, (4/3)\gamma \log N_t]$,
S2 $|M_{t-\ell/2}^h(R) \cap M_t^h(R)| \geq (1/3)\gamma \log N_t$,
S3 $|V_I^a(R)| \leq (1/20)\gamma \log N_t$ for $I = [t - \ell/2, t]$, and
S4 for all $t' \in [t - \hat{L}, t]$, $|V_{t'}^h(R)| \in [(1-\delta)|V_t^h(R)|, (1+\delta)|V_t^h(R)|]$ for a small constant $\delta > 0$.

Suppose that the system has been safe so far. Then the following claims hold:

C1 For all $R = R_i(v)$ for some $v \in V_t^h$, $|M_{t-\ell/2}^h(R) \cap M_t^h(R)| \geq |\Gamma_{v,t}(R)|/5$.
C2 For every routed message accepted by some $v \in V_t^h(R_i(m))$ there is a $t' < t$ s.t. m was sent to v by some $u \in V_t^h(R_{i-1}(m))$ if $i > 0$, and otherwise m was sent to v by $\mathsf{Source}(m) \in \Gamma_{v,t}(R_0(m))$.
C3 For every routed message m generated at time t with $\mathsf{Source}(m) \in M_t^h$ and all $i \geq 0$, m is accepted by all $v \in M_t^h(R_i(m)) \cap M_{t'}^h(R_i(m))$ at some time $t' \in [t, t+i]$.

Using these claims, one can show the following lemma.

Lemma 1. *As long as the system is safe, any* Insert *or* Lookup *operation or* Create *operation with a successful ID generation stage initiated by a mature honest node needs* $O(\log N_t)$ *time and* $O(\text{polylog} N_t)$ *work to be completed.*

3.8 Invariants

For the safeness of the system as well as the correct execution of Join and Leave operations, we need the following invariants ($\alpha > \beta > 0$ are constants).

I1 For all $v \in V_t$ it holds for all $w \in V_t^h$ with $v \in \Gamma_t(w)$ that w's views on v match. Thus, $\mathsf{ID}(v)$ and r_v are well-defined.
I2 For all $v \in V_t$, $\mathsf{ID}(v)$ is a random value in [0,1).
I3 For all $v \in V_t$ there is a $t' \in [t - \alpha \log N_t, t - \beta \log N_t]$: $v \in A_{t'} \cup C_{t'}$.
I4 For all $v \in M_t^h$ it holds that $M_t^h(R_i(v)) \subseteq \Gamma_{v,t}(R_i(v))$ for all i.

The safeness and the invariants are shown to be true by induction:

Lemma 2. *As long as the system is safe, the invariants are fulfilled.*

Lemma 3. *As long as the invariants are fulfilled, the system is safe, with high probability.*

Lemma 4. *As long as the system is safe and the invariants hold, any* Join *or* Leave *operation executed by an honest peer needs* $O(\log N_t)$ *time and* $O(\text{polylog} N_t)$ *work to be completed, and every mature honest peer can keep* $\Theta(\log N_t)$ *nodes in the system, with high probability.*

This completes the proof of Theorem 2. The full paper will be made available at www.cs.jhu.edu/~scheideler.

References

1. B. Awerbuch and C. Scheideler. Robust distributed name service. In *Proc. of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS),* 2004.
2. R. Canetti, R. Gennaro, A. Herzberg, and D. Naor. Proactive security: Long-term protection against break-ins. *RSA CryptoBytes,* 3(1):1–8, 1997.
3. M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. of the 5th Usenix Symp. on Operating Systems Design and Implementation (OSDI),* 2002.
4. M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. of the 2nd Usenix Symp. on Operating Systems Design and Implementation (OSDI),* 1999.
5. S. Crosby and D. Wallach. Denial of service via algorithmic complexity attacks. In *Usenix Security,* 2003.
6. J. R. Douceur. The sybil attack. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS),* 2002.
7. P. Druschel and A. Rowstron. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001),* 2001.
8. A. Fiat and J. Saia. Censorship resistant peer-to-peer content addressable networks. In *Proc. of the 13th ACM Symp. on Discrete Algorithms (SODA),* 2002.
9. Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. In *Proc. of the 38th IEEE Symp. on Foundations of Computer Science (FOCS),* pages 384–393, 1997.
10. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *Proc. of the ACM Conference on Computer and Communications Security (CCS),* pages 100–110, 1997.
11. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO '95,* pages 339–352, 1995.
12. L. Lamport. The weak Byzantine generals problem. *Journal of the ACM,* 30(3):669–676, 1983.
13. L. Lamport and N. Lynch. Distributed computing. Chapter of Handbook on Theoretical Computer Science. Also, to be published as Technical Memo MIT/LCS/TM-384, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1989.

14. M. Naor and U. Wieder. Novel architectures for P2P applications: the continuous-discrete approach. In *Proc. of the 15th ACM Symp. on Parallel Algorithms and Architectures (SPAA),* 2003.
15. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proc. of the 10th ACM Symp. on Principles of Distributed Computing (PODC),* pages 51–59, 1991.
16. R. De Prisco, B. W. Lampson, and N. Lynch. Revisiting the Paxos algorithm. In *Workshop on Distributed Algorithms,* pages 111–125, 1997.
17. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of the ACM SIGCOMM '01,* 2001.
18. B.Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of California at Berkeley, Computer Science Department, 2001.

Further Improvements in Competitive Guarantees for QoS Buffering

Nikhil Bansal[1], Lisa K Fleischer[1], Tracy Kimbrel[1], Mohammad Mahdian[2], Baruch Schieber[1], and Maxim Sviridenko[1]

[1] IBM Watson Research Center, Yorktown Heights, NY 10598.
{nikhil,lkf,kimbrel,sbar,sviri}@us.ibm.com
[2] Department of Mathematics, MIT, Cambridge MA 02139.
mahdian@theory.lcs.mit.edu

Abstract. We study the behavior of algorithms for buffering packets weighted by different levels of Quality of Service (QoS) guarantees in a single queue. Buffer space is limited, and packet loss occurs when the buffer overflows. We describe a modification of the previously proposed "preemptive greedy" algorithm of for buffer management and give an analysis to show that this algorithm achieves a competitive ratio of at most 1.75. This improves upon recent work showing a 1.98 competitive ratio, and a previous result that shows a simple greedy algorithm has a competitive ratio of 2.

1 Introduction

Quality of Service guarantees for network service allow service providers to address the service requirements of their customers by providing different levels of service. In the network setting where traffic volumes may exceed network capacity, effective management of packets at network buffers is key to achieving QoS guarantees. By differentiating service levels, packets of different types of customers may be treated according to the level of service they require. The importance of this issue is reflected in the interest devoted to it in the networking community [9,12,13,22,21,23,24].

We consider the problem of buffer management for a single queue with a limited capacity buffer. Each time step, many packets may enter the buffer, some packets may be dropped, and the packet at the head of the queue is *delivered.* Packets are delivered on a FIFO basis: once packets enter the buffer, they are not reordered. We abstract the differentiated service model by attributing different values to different packets according to their service level. The goal is to deliver the set of packets with highest total value.

We study the behavior of FIFO queues, since FIFO helps to both ensure a level of fairness and prevent packets from timing out, but more importantly, FIFO ensures that packets arrive at their destination in the order they were transmitted. In video streaming, packets from the same source may belong to different service levels according to the type of information they contain [19]. FIFO queues in a QoS environment ensure that these packets arrive in order.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 196–207, 2004.

Our Contribution. In this paper, we use competitive analysis to show that a modification of the preemptive greedy algorithm of Kesselman et al. [15] achieves a competitive ratio of 1.75. The algorithm of Kesselman et al. [15] is the first algorithm to break the bound of 2. Their improvement to 1.98 is small, but the algorithmic framework is important, and we believe it is a good approach for achieving better ratios.

Packets are dropped for two reasons:

1. *Evicted* packets are packets (either already in the buffer, or just arrived) that are dropped because the buffer is too full. In this case, the minimum value packet is dropped.
2. For a given parameter $\beta > 1$, when a packet of value v arrives at the buffer, if there is a packet with value less than v/β in the buffer, then one such packet is dropped. This is a *preempted* packet. Kesselman et al. [15] propose to preempt the first such packet in FIFO order. We drop the first such packet that is a local minimum in FIFO order: the packet our algorithm drops has value strictly smaller than the value of the packet following it in the FIFO queue. To obtain the guarantee of 1.75, we use $\beta = 4$.

Previous work. Prior to [15], the best known competitive ratio for the single queue problem was 2 which was obtained by the greedy algorithm that drops minimum value packets only when the buffer is full [16]. If the decision to drop a packet can be made only when the packet arrives, and once a packet is accepted into the buffer it must be delivered, then the best possible competitive ratio is $\Theta(\log \alpha)$ where α is the ratio of the maximum valued packet to the minimum [2, 4].

Many other models and approaches to studying this problem have been proposed. A recent paper of Albers and Schmidt looks at the unweighted case for a multiqueue switch [3]. Using competitive analysis, other papers study the case with just two weights [2,18], unlimited capacity buffers [8,17], multiqueue switches [3,6], and multiple-node networks [1,7]. Other approaches to studying this problem include probabilistic models of packet injection [10,20] and adversarial queueing theory [5,11].

Recently, Azar and Richter [7] define a class of algorithms for which guarantees for the case when all packets have values in {0,1} can be extended to arbitrary values. The class of algorithms they define considers only the relative order of values of packets. Our algorithm does not fall into this class since it behaves differently when given packets of value 1 and $1 + \epsilon$ than when given packets of value 1 and $v \geq \beta + \epsilon$ for a small positive ϵ. However, Lemma 1 in our paper can be considered a generalization of the zero-one principle of Azar and Richter [7].

2 Model Description

In this section, we give a formal description of the model considered in this paper. Our model is the same as the FIFO model studied in [15] and equivalent to the one considered by [16].[1]

In our setting a switch may deliver one packet per unit time. Packets might arrive at any time, and only one packet can be delivered at the end of each time slot. There is a buffer that can be used to store B packets. The buffer is FIFO, i.e., the delivered packets need to follow the arrival order. Due to the bounded size of the buffer, sometimes packets must be dropped. A buffer management algorithm must decide at each step which of the packets to drop, which to keep in the buffer, and which to deliver, satisfying some constraints specified below. Fix a buffer management algorithm A. At the *end* of each time slot t, there is a set of packets $Q_A(t)$ such that $|Q_A(t)| \leq B$ stored in the buffer; we define $Q_A(0) = \emptyset$. Each packet p has a positive real value denoted $v(p)$. At time t, a single packet from $Q_A(t)$, denoted $A(t)$, is delivered. The algorithm is also allowed to not deliver any packet at time t, in which case we let $A(t) = \emptyset$, and define $v(\emptyset) = 0$. During the next time slot (time slot $t+1$), a set of packets $\mathrm{Arr}(t+1)$ arrives. A subset of $Q_A(t) \cup \mathrm{Arr}(t+1) \setminus \{A(t)\}$, denoted $\mathrm{Dr}_A(t)$, is *dropped,* and the rest of packets are kept in the buffer. The set of packets in the buffer at the end of time slot $t+1$ is $Q_A(t+1) = Q_A(t) \cup \mathrm{Arr}(t+1) \setminus (\mathrm{Dr}_A(t) \cup \{A(t)\})$. Notice that we allow the buffer management algorithm to drop a packet that was placed in the buffer during an earlier time slot.

We consider only instances that are finite, i.e., those in which there is a time T such that no new packet arrives after time $T-B$, and therefore we can assume without loss of generality that the algorithm will not deliver any packet after time T. The *value delivered* by the algorithm is the sum of the values of all packets delivered by the algorithm. For a set P of packets define $v(P)$ to be the total value of the packets in the set. Also, we denote the set of packets that an algorithm A delivers by $S_A = \{A(t) : t = 0, \ldots, T\}$. In this notation the value delivered by algorithm A is $v(S_A)$.

Our goal is to compare the ratio of the value of packets delivered by our online algorithm to the value of packets delivered by an optimal clairvoyant algorithm (i.e., an offline algorithm that sees the complete input sequence in advance, and using this information delivers a set of packets with maximum value), denoted OPT. An online algorithm A is called **c-competitive,** if for every instance, $v(S_{\mathrm{OPT}})$ is at most c times $v(S_A)$.

3 Algorithm

We analyze a modification which we call PG of the *preemptive greedy algorithm* of Kesselman et al. [15] with parameter $\beta > 1$. This algorithm is shown in Figure 1.

[1] The parameter B in their model is equivalent to $B + 1$ in ours.

The Modified β-Preemptive Greedy (PG) Algorithm.

When a packet p of value $v(p)$ arrives, do the following:

1. Find the first packet p' in FIFO order such that p' has value less than $v(p)/\beta$ and less than the value of the packet following p' in the buffer (if any). If such a packet exists, preempt it.
2. If there is free space in the buffer, accept p
3. Otherwise (the buffer is full), then evict a packet p' with the smallest value among $\{p\}$ and those in the buffer, and keep the rest in the buffer.

Fig. 1. The algorithm PG

Every time a new packet p arrives, our algorithm determines whether there exists a packet p' in the buffer such that $v(p') < v(p)/\beta$. If so, the algorithm finds the first such packet p' in the buffer. If the next packet in the buffer has value no greater than the value of p', we let p' be the next packet and iterate, until we find the first packet p' that has value less than $v(p)/\beta$, and less than the value of the next packet in the buffer.[2] We drop p' from the buffer, and say that p' is *preempted* by p. Next, the algorithm determines whether there is free space for p in the buffer, either because some p' was preempted or because there was free space to begin with. If so, p is added to the tail of the buffer; otherwise, a packet of smallest value among $\{p\}$ and those in the buffer is *evicted,* and the rest are kept in the buffer. At the end of each time slot, the algorithm delivers the packet at the head of the buffer (if such a packet exists).

Kesselman et al. [15] give an example that shows that the competitive ratio of their algorithm is at least $\max\{2 - 1/\beta, \beta/(\beta - 1)\}$. This example yields the same lower bound on the competitive ratio of our algorithm. The main result of our paper is to show that this lower bound is tight for $\beta \geq 4$.

Theorem 1. *For every $\beta \geq 4$, the algorithm in Figure 1 has a competitive ratio of $2 - 1/\beta$.*

Setting $\beta = 4$, the above theorem shows that the algorithm PG is a 1.75-competitive algorithm for the buffer management problem.

4 Analysis of the Algorithm

In this section, we prove Theorem 1. The proof is composed of three steps: First, we use the theory of linear programming (specifically, the fact that every linear program has a basic feasible solution) to limit the set of possible instances we need to consider to find the competitive ratio of our algorithm. Using this, we

[2] The original algorithm of Kesselman et al. [15] always preempts the first packet in the buffer of value less than $v(p)/\beta$, without comparing it to the value of the next packet in the buffer.

can focus on one such instance and describe its structure in terms of several parameters, and also compute the value delivered by our algorithm and the optimal algorithm on this instance in terms of these parameters. In the second step, we prove several inequalities relating these parameters. This allows us to bound the competitive ratio of our algorithm by solving a maximization linear program which, following the terminology of [14], we call a *factor-revealing LP*. The third and final step is to prove an upper bound on the value of an optimal solution of this linear program. By LP duality, this step is equivalent to finding a feasible solution to the dual of the factor-revealing LP.

4.1 Structure of the Worst-Case Examples

In this section we show that is that it is enough to analyze the performance of our algorithm on instances in which the value of each packet is either 0 or β^i for some $i \geq 0$, but ties are allowed to be broken by an adversary. More precisely, we have the following lemma.

Lemma 1. *Fix an instance size T. The worst case competitive ratio of the modified preemptive greedy algorithm over instances of size at most T is realized by an input sequence in which the value of each packet is zero or a power of β, and an adversary determines for every two packets p and p' and number γ, where $v(p) = \gamma v(p')$, whether the inequality $v(p) \geq \gamma v(p')$ is "true" or "false".*

Proof. Consider an arbitrary instance $\mathcal{R}$ of size T. For every two packets p and p', write all inequalities of the form "$v(p)$? $v(p')$" or "$v(p)$? $\beta v(p')$" (where ? is either >, =, or <) that are true in the instance $\mathcal{R}$. Let $\mathcal{C}$ denote the collection of these inequalities. Now, observe that PG considers the value of a packet only to compare it to either the value of another packet, β times the value of another packet, or the value of another packet divided by β. Therefore, on every instance that differs from $\mathcal{R}$ in the values of the packets, but is the same as $\mathcal{R}$ in the arrival times and satisfies the inequalities in $\mathcal{C}$, PG will behave exactly as it does on $\mathcal{R}$, and hence the subsequence of packets delivered will be the same. Next, we can write a linear program with a set of variables corresponding to the set $\{v(p)\}$ of packet values. This LP consists of the inequalities in $\mathcal{C}$ with equalities added to strict inequalities (i.e., < changed to $\leq$ and > changed to $\geq$) and an extra normalization inequality $\sum_{p \in S_{\mathrm{PG}}(\mathcal{R})} v(p) = 1$, and the objective is to maximize $\sum_{p \in S_{\mathrm{OPT}}(\mathcal{R})} v(p)$. Take an optimal basic feasible solution of this LP, and construct an instance $\mathcal{R}'$ which is the same as $\mathcal{R}$, except that values of the packets are replaced by those of the basic feasible solution of the LP. We also let the adversary break the ties in $\mathcal{R}'$ in the same direction as in $\mathcal{R}$. It is clear that the algorithm PG delivers the same subsequence of packets in $\mathcal{R}'$ as in $\mathcal{R}$, and therefore, the competitive ratio of PG on $\mathcal{R}'$ is at least as great as on $\mathcal{R}$. Let $\mathcal{L}$ denote the collection of linear programs that consist of inequalities of the form "$v(p)$? $v(p')$" or "$v(p)$? $\beta v(p')$" (where ? is either $\leq$ or $\geq$), and $\sum_{p \in S_1} v(p) = 1$ for a set $S_1 \subseteq \{1, \ldots, T\}$, and let $\mathcal{A}$ denote the set of instances that are basic feasible solutions of linear programs in $\mathcal{L}$. By the above argument,

the competitive ratio of PG is at most its competitive ratio on instances in $\mathcal{A}$, if an adversary is allowed to break the ties. Since the number of instances in $\mathcal{A}$ and the number of ways to break ties is finite, it follows that there is an instance in $\mathcal{A}$ that achieves the worst ratio. □

From now on we only consider input sequences in which each packet value is either zero or a power of β. In the rest of this section, we define a hierarchical structure of time intervals and several parameters that describe the behavior of our algorithm and of OPT on a given instance.

Intervals. An *interval of type* i is a time interval I such that at every step $t \in I$ the algorithm PG delivers a packet of value at least β^i, and I is a maximal interval with this property; i.e., in the time step immediately before the beginning and in the step immediately after the end of I the algorithm delivers either nothing, or a packet of value less than β^i. Let $\mathcal{I}_i$ denote the set of maximal intervals of type i, and let $\mathcal{I}$ be the union over all i of $\mathcal{I}_i$. Since the same interval can be in $\mathcal{I}_i$ for more than one value of i, we consider $\mathcal{I}$ as a multiset.

The *last eviction* in an interval I of type i is the latest time step t in I such that at time t a packet of value *at least* β^i is evicted from the buffer. If no such time step exists, we say that I has *no evictions.* (Note that there may be an eviction within I of a packet of value $\beta^{i'} < \beta^i$ after the last eviction of I; we will consider this to belong to the interval of type i' containing I and those containing it, but not to I.) Let $\mathcal{E}$ be the set of intervals that have evictions, and let $\mathcal{N}$ be the set of intervals that have no evictions.

The interval structure. We define an *interval structure* as a sequence of ordered rooted trees, with a partition of its vertex set into two subsets $\mathcal{E}$ and $\mathcal{N}$. The intervals in $\mathcal{I}$ can be represented using such a structure (denoted by $\mathcal{T}$) as follows. The root of each tree is an interval in $\mathcal{I}_0$, and the children of each interval $I \in \mathcal{I}_i$ are the maximal intervals of type $i+1$ that are contained in I. These children are ordered by increasing time from left to right (e.g., the leftmost child corresponds to the interval that ends before the others begin). The trees are also ordered by increasing time. A vertex is in $\mathcal{E}$ if the corresponding interval has an eviction; otherwise, it is in $\mathcal{N}$. We denote the set of children of I by child (I) and the parent of I by par(I). Also, we call the leftmost (rightmost, respectively) child of I that is in $\mathcal{E}$ the first (last, respectively) child of I, and denote it by lastch(I) (firstch(I), respectively). Notice that, despite the name, I can have other children (that have no evictions) to the left of firstch(I) or to the right of lastch(I). The set of nodes that are the last children of their parents is denoted by LC. Also, we let rsib(I) denote the set of siblings of I that are to the right of I and are in $\mathcal{E}$.

In the next section, we will show that for every interval structure $\mathcal{T}$, the competitive ratio of the algorithm PG on instances whose corresponding interval structure is $\mathcal{T}$ can be bounded by the solution of a linear program indexed by $\mathcal{T}$.

Packets assigned to each interval. Consider an interval I of type i. We assign all packets of value at least β^i delivered or evicted during I to this interval. For a

preempted packet p, we define $\mathbf{next}(p)$ as the first packet after p in arrival order that is delivered by PG. We assign p to I if p has value at least β^i and $\mathbf{next}(p)$ is assigned to I. To simplify the notation, we let $\mathbf{next}(p) = p$ for a packet p that is delivered or evicted by PG. Let P_I denote the set of packets assigned to an interval I.

Lemma 2. *For each packet p of value at least β^i there is a unique interval I of type i such that p is assigned to I.*

Proof. The statement is trivial if p is delivered by PG. If p is evicted by PG, then the packet delivered at the same time step must have a value at least that of p, and hence p must be assigned to a unique interval of type i.

Now suppose that p is preempted. In order to show that there is a unique interval of type i that p is assigned to, it is enough to show that $next(p)$ is of value at least β^i. Consider the sequence $p = p_1, p_2, \ldots, p_j = next(p)$ where for each $1 \leq k < j$, p_k is followed immediately in the buffer by p_{k+1} at the time p_k is evicted or preempted. In each case, whether an eviction or a preemption, $v(p_k) \leq v(p_{k+1})$, and the lemma follows.

□

Lemma 3. *Let I be an interval of type i with evictions, and let $t \in I$ be a time at which a packet of value at least β^i is evicted. Then every packet that is in the buffer at time t is assigned to I.*

Proof. Assume for the sake of contradiction that there is a packet p in the buffer at time t that is not assigned to I. Therefore, p must not be evicted at time t, nor should it be delivered in the interval I. All packets in the buffer at time t have value at least β^i; thus p cannot be delivered in an interval after I. Therefore p must be dropped some time after t. Let $p_1 = \mathbf{next}(p)$, and p_2 be the packet delivered at the time step after the end of I. By definition, $v(p_2) < \beta^i$, so p_2 cannot be present in the buffer at time t. Therefore p_2 arrives after p. This means that p_1 must arrive before p_2, for otherwise p_1 would not be the first packet after p that is delivered. But then p_1 must be delivered earlier than p_2, and hence $p_1 \in P_I$. Thus, $p \in P_I$, a contradiction. □

We are now ready to define several parameters for each interval I. These definitions are presented in Table 1. In the next section, we will prove inequalities relating these parameters, and then we will treat them as variables in a linear program. The following lemma states the value delivered by PG and OPT in terms of these parameters. Here for every interval I of type $i > 0$, $b(I)$ denotes $(\beta - 1)\beta^{i-1}$. If I is an interval of type 0, then $b(I) = 1$.

Lemma 4. *We have $v(S_{\mathrm{PG}}) = \sum_{I \in \mathcal{I}} b(I)A(I)$ and $v(S_{\mathrm{OPT}}) = \sum_{I \in \mathcal{I}} b(I)X(I)$.*

Proof. Let I be an interval of type i. By Lemma 2 and the definition of $A(I)$, the number of packets of value exactly β^i in $P_I \cap S_{\mathrm{PG}}$ is $A(I) - \sum_{J \in \mathrm{child}(I)} A(J)$.

Table 1. Parameters associated with an interval

Param	Definition
$A(I)$	$\lvert P_I \cap S_{\mathrm{PG}}\rvert$, or equivalently, the length of I
$X(I)$	$\lvert P_I \cap S_{\mathrm{OPT}}\rvert$
$A^e(I)$	$I \in \mathcal{E}$: # of packets in P_I delivered by PG after the last eviction of I
$R^+(I)$	# of packets in P_I that are preempted by packets not in P_I
$R^-(I)$	$I \in \mathcal{N}$: # of packets in P_I that preempt packets not in P_I $I \in \mathcal{E}$: # of packets in P_I that preempt packets in $\{p : \mathrm{next}(p) \notin P_I\}$
$R^d(I)$	$I \in \mathcal{N}$: 0 $I \in \mathcal{E}$: # of packets in P_I that preempt packets not in P_I after the last eviction of I
$R^e(I)$	$I \in \mathcal{N}$: # of packets in P_I that are preempted by packets in P_I $I \in \mathcal{E}$: # of packets in P_I that are preempted by other packets in P_I after the last eviction in I

Therefore, the total value delivered by PG is

$$\sum_i \sum_{I \in \mathcal{I}_i} \beta^i \Big(A(I) - \sum_{J \in \mathrm{child}(I)} A(J)\Big) = \sum_i \sum_{I \in \mathcal{I}_i} \beta^i A(I) - \sum_i \sum_{I \in \mathcal{I}_{i+1}} \beta^i A(I)$$
$$= \sum_{I \in \mathcal{I}} b(I) A(I).$$

A similar argument yields the expression for $v(S_{\mathrm{OPT}})$. □

4.2 Inequalities

In the following sequence of lemmas, we prove several inequalities between the parameters defined in the previous section.

Lemma 5. *For every* $I \in \mathcal{E}$, $A^e(I) + R^+(I) \geq B + R^d(I)$.

Proof. Consider the set of packets in P_I that are either in the buffer at the time of the last eviction (by Lemma 3 we know that all these packets are in P_I), or arrive after the last eviction. We construct a graph on this set of packets by forming an edge from a packet p_1 to a packet p_2 if p_1 preempts p_2. By the definition of preemption, each vertex in this graph has in-degree and out-degree at most one, and so, this graph is a union of disjoint paths. Therefore, the number of vertices of in-degree zero is equal to the number of vertices of out-degree zero in this graph. Now, notice that the vertices corresponding to the B packets that were present in the buffer at the time of the last eviction, and also the vertices corresponding to the $R^d(I)$ packets that preempt packets outside P_I have outdegree zero. Therefore, the number of vertices of outdegree zero is at least $B + R^d(I)$. On the other hand, every vertex of indegree zero corresponds to a packet in P_I that is either not preempted (and therefore is delivered), or is preempted by packets outside P_I. Thus, the number of vertices of indegree zero is at most $A^e(I) + R^+(I)$. Hence, $A^e(I) + R^+(I) \geq B + R^d(I)$. □

Lemma 6. *For every interval* $I \in \mathcal{N}$, $R^-(I) \leq A(I) + R^+(I)$.

Proof. We use the same argument as in the proof of Lemma 5, except here we consider the graph of preemptions on the set of all packets in P_I. Since I has no evictions, the number of vertices of indegree zero is equal to $A(I) + R^+(I)$, and the number of vertices of outdegree zero is at least $R^-(I)$. □

Lemma 7. *For every* $I \in \mathcal{N}$, $X(I) \leq A(I) + R^e(I) + R^+(I)$.

Proof. Since no packet in P_I is evicted, every packet must either be delivered, preempted by another packet in P_I, or preempted by a packet outside P_I. The number of such packets is $A(I)$, $R^e(I)$, and $R^+(I)$, respectively. □

Due to space constraint, the proofs of the following lemmas are left to the full version of the paper.

Lemma 8. *For every* $I \in \mathcal{E}$, *let* y_I *denote the number of packets in* P_I *that* OPT *delivers before the start of I. Then,* $X(I) \leq A(I) + R^e(I) + R^+(I) + y_I$.

Lemma 9. *For every* $I \in \mathcal{E}$, $X(I) \leq A(I) + B - R^-(I) + R^e(I) + R^+(I)$.

Lemma 10. *For every interval* $I \in \mathcal{E}$ *of type* $i > 0$,

$$X(I) \leq \frac{1}{2}\Bigg(A(I) + A(\text{par}(I)) - \sum_{J \in \text{rsib}(I)} A(J) + B - A^e(\text{par}(I)) + A^e(\text{lastch}(\text{par}(I)))\Bigg) + R^e(I) + R^+(I).$$

Lemma 11. *For every interval I of type 0, we have* $X(I) \leq A(I) + \frac{B}{2} + R^e(I) + R^+(I)$.

Lemma 12. *For every* $i \geq 0$, $\sum_{I \in \mathcal{I}_i}(R^e(I) + R^+(I)) \leq \sum_{I \in \mathcal{I}_{i+1}}(R^e(I) + R^-(I) + R^d(I))$.

The following theorem summarizes the results of this section.

Theorem 2. *Let* $\mathcal{T}$ *be an interval structure, and* $z_{\mathcal{T}}$ *denote the solution of the following maximization program (which we call a factor-revealing LP) with variables* OPT, ON, B, $A(I)$, $X(I)$, $A^e(I), R^+(I), R^-(I), R^e(I)$, *and* $R^d(I)$ *for every* $I \in \mathcal{I}$.

$$\textit{maximize} \quad \frac{\text{OPT}}{\text{ON}}$$

$$\textit{subject to} \quad \text{ON} = \sum_{I \in \mathcal{I}} b(I)A(I) \tag{1}$$

$$\text{OPT} = \sum_{I \in \mathcal{I}} b(I)X(I) \tag{2}$$

$$\forall I \in \mathcal{E}: \; A^e(I) + R^+(I) \geq B + R^d(I) \tag{3}$$

$$\forall I \in \mathcal{N}: \; R^-(I) \leq A(I) + R^+(I) \tag{4}$$

$$\forall I \in \mathcal{N}: \; X(I) \leq A(I) + R^e(I) + R^+(I) \tag{5}$$

$$\forall I \in \mathcal{E}: \; X(I) \leq A(I) + B - R^-(I) + R^e(I) + R^+(I) \tag{6}$$

$$\forall I \in \mathcal{E} \setminus \mathcal{I}_0: \; X(I) \leq \frac{1}{2}\Bigg(A(I) + A(\text{par}(I)) - \sum_{J \in \text{rsib}(I)} A(J) + B - A^e(\text{par}(I)) + A^e(\text{lastch}(\text{par}(I)))\Bigg) + R^e(I) + R^+(I) \tag{7}$$

$$\forall I \in \mathcal{I}_0: \; X(I) \leq A(I) + \frac{B}{2} + R^e(I) + R^+(I) \tag{8}$$

$$\forall i: \; \sum_{I \in \mathcal{I}_i} (R^e(I) + R^+(I)) \leq \sum_{I \in \mathcal{I}_{i+1}} (R^e(I) + R^-(I) + R^d(I)) \tag{9}$$

$$\forall I \in \mathcal{I}: \; A^e(I) \leq A(I) \tag{10}$$

$$\forall I \in \mathcal{I}: \; A(I), X(I), A^e(I), R^+(I), R^-(I), R^e(I), R^d(I) \geq 0 \tag{11}$$

Then on every instance $\mathcal{R}$ whose corresponding interval structure is $\mathcal{T}$, the ratio of the value delivered by the optimal solution to the value delivered by the algorithm PG*is at most $z_{\mathcal{T}}$.*

4.3 Analysis of the Factor-Revealing LP

The last step in the proof is to analyze the maximization program of Theorem 2, and prove that for every $\mathcal{T}$ and $\beta \geq 4$, the solution of this program is at most $2 - 1/\beta$. This program is equivalent to a linear program. Therefore, by LP duality, this step can be done by multiplying each inequality with an appropriate multiplier, and adding them up. Due to space constraint, this analysis is left to the full version of the paper.

References

1. W. Aiello, R. Ostrovsky, E. Kushilevitz, and A. Rosen. Dynamic routing on networks with fixed size buffers. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms,* pages 771–780, 2003.
2. William Aiello, Yishay Mansour, S. Rajagopolan, and Adi Rosen. Competitive queue policies for differentiated services. In *Proceedings of the IEEE INFOCOM,* pages 431–440, 2000.
3. S. Albers and M. Schmidt. On the performance of greedy algorithms in packet buffering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing,* 2004.

4. N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies for QoS switches. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms,* pages 761–770, 2003.
5. Andrews, Awerbuch, Fernandez, Kleinberg, Leighton, and Liu. Universal stability results for greedy contention-resolution protocols. In *37th IEEE Symposium on Foundations of Computer Science (FOCS),* pages 380–389, 1996.
6. Y. Azar and Y. Richter. Management of multi-queue switches in QoS netowrks. In *35th ACM Symposium on Theory of Computing,* pages 82–89, 2003.
7. Y. Azar and Y. Richter. The zero-one principle for switching networks. In *Proc. 34th ACM Symposium on Theory of Computing,* 2004.
8. Amotz Bar-Noy, Ari Freund, Shimon Landa, and Joseph (Seffi) Naor. Competitive on-line switching policies. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms,* 2002.
9. Y. Bernet, A. Smith, S. Blake, and D. Grossman. A conceptual model for diffserv routers. Internet draft, March 2000.
10. Alexander Birman, H. Richard Gail, Sidney L. Hantler, Zvi Rosberg, and Moshe Sidi. An optimal service policy for buffer systems. *Journal of the ACM,* 42(3):641–657, 1995.
11. Allan Borodin, Jon Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queuing theory. *Journal of the ACM,* 48(1):13–38, 2001.
12. D. Clark and J. Wroclawski. An approach to service allocation in the Internet. Internet draft, July 1997.
13. Constantinos Dovrolis, Dimitrios Stiliadis, and Parameswaran Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *SIGCOMM,* pages 109–120, 1999.
14. K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problem. In *Proceedings of the 34st Annual ACM Symposium on Theory of Computing,* 2002.
15. A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. In *Proc, 11th Annual European Symposium on Algorithms (ESA),* pages 361–372, 2003.
16. Alexander Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir, Baruch Schieber, and Maxim Sviridenko. Buffer overflow management in QoS switches. In *ACM Symposium on Theory of Computing,* pages 520–529, 2001.
17. H. Koga. Balanced scheudling towards loss-free packet queueing and delay fairness. In *Proc. 12th Annual International Symposium on Algorithms and Computation,* pages 61–73, 2001.
18. Z. Lotker and B. Patt-Shamir. Nearly optimal fifo buffer management for Diff-Serv. In *Proc. 21st ACM-SIAM Symposium on Principles of Distributed Computing (PODC),* pages 134–142, 2002.
19. Y. Mansour, B. Patt-Shamir, and O. Lapid. Optimal smoothing schedules for real-time streams. In *Proc. 19th ACM Symp. on Principles of Distributed Computing,* pp. 21–29, 2000.
20. Martin May, Jean-Chrysostome Bolot, Alain Jean-Marie, and Christophe Diot. Simple performance models of differentiated services schemes for the internet. In *Proc. IEEE INFOCOM,* pages 1385–1394, 1999.
21. T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. Delay differentiation and adaptation in core stateless networks. In *Proc. IEEE INFOCOM,* 2000.
22. K. Nichols, V. Jacobson, and L. Zhang. A twobit differentiated services architecture for the internet. Internet draft, 1997.

23. Nemo Semret, Raymond R.-F. Liao, Andrew T. Campbell, and Aurel A. Lazar. Peering and provisioning of differentiated internet services. In *Proc. IEEE INFOCOM,* pages 414–420, 2000.
24. Ion Stoica and Hui Zhang. Providing guaranteed services without per flow management. In *Proc. ACM SIGCOMM,* pages 81–94, 1999.

Competition-Induced Preferential Attachment

N. Berger[1], C. Borgs[1], J.T. Chayes[1], R.M. D'Souza[1], and R.D. Kleinberg[2]

[1] Microsoft Research, One Microsoft Way, Redmond WA 98052, USA
[2] M.I.T. CSAIL, 77 Massachusetts Ave, Cambridge MA 02139, USA.
Supported by a Fannie and John Hertz Foundation Fellowship.

Abstract. Models based on preferential attachment have had much success in reproducing the power law degree distributions which seem ubiquitous in both natural and engineered systems. Here, rather than assuming preferential attachment, we give an explanation of how it can arise from a more basic underlying mechanism of competition between opposing forces.

We introduce a family of one-dimensional geometric growth models, constructed iteratively by locally optimizing the tradeoffs between two competing metrics. This family admits an equivalent description as a graph process with no reference to the underlying geometry. Moreover, the resulting graph process is shown to be preferential attachment with an upper cutoff. We rigorously determine the degree distribution for the family of random graph models, showing that it obeys a power law up to a finite threshold and decays exponentially above this threshold.

We also introduce and rigorously analyze a generalized version of our graph process, with two natural parameters, one corresponding to the cutoff and the other a "fertility" parameter. Limiting cases of this process include the standard preferential attachment model (introduced by Price and by Barabási-Albert) and the uniform attachment model. In the general case, we prove that the process has a power law degree distribution up to a cutoff, and establish monotonicity of the power as a function of the two parameters.

1 Introduction

1.1 Network Growth Models

There is currently tremendous interest in understanding the mathematical structure of networks – especially as we discover how pervasive network structures are in natural and engineered systems. Much recent theoretical work has been motivated by measurements of real-world networks, indicating they have certain "scale-free" properties, such as a power-law distribution of degree sequences. For the Internet graph, in particular, both the graph of routers and the graph of autonomous systems (AS) seem to obey power laws [14,15]. However, these observed power laws hold only for a limited range of degrees, presumably due to physical constraints and the finite size of the Internet.

Many random network growth models have been proposed which give rise to power law degree distributions. Most of these models rely on a small number

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 208–221, 2004.

of basic mechanisms, mainly preferential attachment[1] [19,4] or copying [17], extending ideas known for many years [12,20,22,21] to a network context. Variants of the basic preferential attachment mechanism have also been proposed, and some of these lead to changes in the values of the exponents in the resulting power laws. For extensive reviews of work in this area, see Albert and Barabási [2], Dorogovtsev and Mendes [11], and Newman [18]; for a survey of the rather limited amount of mathematical work see [7]. Most of this work concerns network models without reference to an underlying geometric space. Nor do most of these models allow for heterogeneity of nodes, or address physical constraints on the capacity of the nodes. Thus, while such models may be quite appropriate for geometry-free networks, such as the web graph, they do not seem to be ideally suited to the description of other observed networks, *e.g.,* the Internet graph.

In this paper, instead of assuming preferential attachment, we show that it can arise from a more basic underlying process, namely competition between opposing forces. The idea that power laws can arise from competing effects, modeled as the solution of optimization problems with complex objectives, was proposed originally by Carlson and Doyle [9]. Their "highly optimized tolerance" (HOT) framework has reliable design as a primary objective. Fabrikant, Koutsoupias and Papadimitriou (FKP) [13] introduce an elegant network growth model with such a mechanism, which they called "heuristically optimized trade-offs". As in many growth models, the FKP network is grown one node at a time, with each new node choosing a previous node to which it connects. However, in contrast to the standard preferential attachment types of models, a key feature of the FKP model is the underlying geometry. The nodes are points chosen uniformly at random from some region, for example a unit square in the plane. The trade-off is between the geometric consideration that it is desirable to connect to a nearby point, and a networking consideration, that it is desirable to connect to a node that is "central" in the network as a graph. Centrality is measured by using, for example, the graph distance to the initial node. The model has a tunable, but fixed, parameter, which determines the relative weights given to the geometric distance and the graph distance.

The suggestion that competition between two metrics could be an alternative to preferential attachment for generating power law degree distributions represents an important paradigm shift. Though FKP introduced this paradigm for network growth, and FKP networks have many interesting properties, the resulting distribution is not a power law in the standard sense [5]. Instead the overwhelming majority of the nodes are leaves (degree one), and a second substantial fraction, heavily connected "stars" (hubs), producing a node degree distribution which has clear bimodal features.[2]

---

[1] As Aldous [3] points out, proportional attachment may be a more appropriate name, stressing the linear dependence of the attractiveness on the degree.

[2] In simulations of the FKP model, this can be clearly discerned by examining the probability distribution function (pdf); for the system sizes amenable to simulations, it is less prominent in the cumulative distribution function (cdf).

Here, instead of directly producing power laws as a consequence of competition between metrics, we show that such competition can give rise to a preferential attachment mechanism, which in turn gives rise to power laws. Moreover, the power laws we generate have an upper cutoff, which is more realistic in the context of many applications.

1.2 Overview of Competition-Induced Preferential Attachment

We begin by formulating a general competition model for network growth. Let $x_0, x_1, \ldots, x_t$ be a sequence of random variables with values in some space Λ. We think of the points $x_0, x_1, \ldots, x_t$ arriving one at a time according to some stochastic process. For example, we typically take Λ to be a compact subset of $\mathbb{R}^d$, x_0 to be a given point, say the origin, and $x_1, \ldots, x_t$ to be i.i.d. uniform on Λ. The network at time t will be represented by a graph, $G(t)$, on $t+1$ vertices, labeled $0, 1, \ldots, t$, and at each time step, the new node attaches to one or several nodes in the existing network. For simplicity, here we assume that each new node connects to a single node, resulting in $G(t)$ being a tree.

Given $G(t-1)$, the new node, labeled t, attaches to that node j in the existing network that minimizes a certain cost function representing the tradeoff of two competing effects, namely connection or startup cost, and routing or performance cost. The connection cost is represented by a metric, $g_{ij}(t)$, on $\{0, \ldots, t\}$ which depends on $x_0, \ldots, x_t$, but not on the current graph $G(t-1)$, while the routing cost is represented by a function, $h_j(t-1)$, on the nodes which depends on the current graph, but not on the physical locations $x_0, \ldots, x_t$ of the nodes $0, \ldots, t$. This leads to the cost function

$$c_t = \min_j \left[\alpha g_{tj}(t) + h_j(t-1)\right], \tag{1}$$

where α is a constant which determines the relative weighting between connection and routing costs. We think of the function $h_j(t-1)$ as measuring the centrality of the node j; for simplicity, we take it to be the hop distance along the graph $G(t-1)$ from j to the root 0.

To simplify the analysis of the random graph process, we will assume that nodes always choose to connect to a point which is closer to the root, i.e. they minimize the cost function

$$\tilde{c}_t = \min_{j:\|x_j\|<\|x_t\|} \left[\alpha g_{tj}(t) + h_j(t-1)\right], \tag{2}$$

where $\|\cdot\|$ is an appropriate norm.

In the original FKP model, Λ is a compact subset of $\mathbb{R}^2$, say the unit square, and the points x_i are independently uniformly distributed on Λ. The cost function is of the form (1), with $g_{ij} = d_{ij}$, the Euclidean metric (modeling the cost of building the physical transmission line), and $h_j(t)$ is the hop distance along the existing network $G(t)$ from j to the root. A rigorous analysis of the degree distribution of this two-dimensional model was given in [5], and the analogous one-dimensional problem was treated in [16].

Our model is defined as follows.

Definition 1 (Border Toll Optimization Process) *Let* $x_0 = 0$, *and let* $x_1, x_2, \ldots$ *be i.i.d., uniformly at random in the unit interval* $\Lambda = [0,1]$, *and let* $G(t)$ *be the following process: At* $t = 0$, $G(t)$ *consists of a single vertex* 0, *the root. Let* $h_j(t)$ *be the hop distance to* 0 *along* $G(t)$, *and let* $g_{ij}(t) = n_{ij}(t)$ *be the number of existing nodes between* x_i *and* x_j *at time* t, *which we refer to as the* jump cost *of* i *connecting to* j. *Given* $G(t-1)$ *at time* $t-1$, *a new vertex, labeled* t, *attaches to the node* j *which minimizes the cost function* (2). *Furthermore, if there are several nodes* j *that minimize this cost function and satisfy the constraint, we choose the one whose position* x_j *is nearest to* x_t. *The process so defined is called the* border toll optimization process (BTOP).

As in the FKP model, the routing cost is just the hop distance to the root along the existing network. However, in our model the connection cost metric measures the number of "borders" between two nodes: hence the name BTOP. Note the correspondence to the Internet, where the principal connection cost is related to the number of AS domains crossed – representing, *e.g.,* the overhead associated with BGP, monetary costs of peering agreements, etc. In order to facilitate a rigorous analysis of our model, we took the simpler cost function (2), so that the new node always attaches to a node to its left.

It is interesting to note that the ratio of the BTOP connection cost metric to that of the one-dimensional FKP model is just the local density of nodes: $n_{ij}/d_{ij} = \rho_{ij}$. Thus the transformation between the two models is equivalent to replacing the constant parameter α in the FKP model with a variable parameter $\alpha_{ij} = \alpha\rho_{ij}$ which changes as the network evolves in time. That α_{ij} is proportional to the local density of nodes in the network reflects a model with an increase in cost for local resources that are scarce or in high demand. Alternatively, it can be thought of as reflecting the economic advantages of being first to market.

Somewhat surprisingly, the BTOP is equivalent to a special case of the following process, which closely parallels the preferential attachment model and makes no reference to any underlying geometry.

Definition 2 (Generalized Preferential Attachment with Fertility and Aging) *Let* A_1, A_2 *be two positive integer-valued parameters. Let* $G(t)$ *be the following Markov process, whose states are finite rooted trees in which each node is labeled either* fertile *or* infertile. *At time* $t = 0$, $G(t)$ *consists of a single fertile vertex. Given the graph at time* t, *the new graph is formed in two steps: first, a new vertex, labeled* $t+1$ *and initialized as infertile, connects to an old vertex* j *with probability zero if* j *is infertile, and with probability*

$$Pr(t+1 \to j) = \frac{\min\{d_j(t), A_2\}}{W(t)} \tag{3}$$

if j *is fertile. Here,* $d_j(t)$ *is equal to* 1 *plus the out-degree of* j, *and* $W(t) = \sum_j' \min\{d_j(t), A_2\}$ *with the sum running over fertile vertices only. We refer to vertex* $t+1$ *as a child of* j. *If after the first step,* j *has more than* $A_1 - 1$ *infertile children, one of them, chosen uniformly at random, becomes fertile. The process*

so defined is called a generalized preferential attachment process with fertility threshold A_1 and aging threshold A_2.

The special case $A_1 = A_2$ *is called the* competition-induced preferential attachment process *with parameter* A_1.

The last definition is motivated by the following theorem, to be proved in Section 2. To state the theorem, we define a graph process as a random sequence of graphs $G(0),G(1),G(2),\ldots$ on the vertex sets $\{0\}$, $\{0,1\}$, $\{0,1,2\},\ldots$, respectively.

Theorem 1 *As a graph process, the border toll optimization process has the same distribution as the competition-induced preferential attachment process with parameter* $A = \lceil \alpha^{-1} \rceil$.

Certain other limiting cases of the generalized preferential attachment process are worth noting. If $A_1 = 1$ and $A_2 = \infty$, we recover the standard model of preferential attachment [19,4]. If $A_1 = 1$ and A_2 is finite, the model is equivalent to the standard model of preferential attachment with a cutoff. On the other hand, if $A_1 = A_2 = 1$, we get a uniform attachment model.

The degree distribution of our random trees is characterized by the following theorem, which asserts that almost surely (a.s.) the fraction of vertices having degree k converges to a specified limit q_k, and moreover that this limit obeys a power law for $k < A_2$, and decays exponentially above A_2.

Theorem 2 *Let* A_1, A_2 *be positive integers and let* $\tilde{G}(t)$ *be the generalized preferential attachment process with fertility parameter* A_1 *and aging parameter* A_2. *Let* $N_0(t)$ *be the number of infertile vertices at time* t, *and let* $N_k(t)$ *be the number of fertile vertices with* $k-1$ *children at time* t, $k \geq 1$. *Then:*

1. *There are numbers* $q_k \in [0,1]$ *such that, for all* $k \geq 0$

$$\frac{N_k(t)}{t+1} \to q_k \quad a.s., \text{ as } t \to \infty. \tag{4}$$

2. *There exists a number* $w = w(A_1, A_2) \in [0,2]$ *such that the* q_k *are determined by the following equations:*

$$q_i = \left(\prod_{k=2}^{i} \frac{k-1}{k+w} \right) q_1 \quad \text{if} \quad i \leq A_2, \tag{5}$$

$$q_i = \left(\frac{A_2}{A_2+w} \right)^{i-A_2} q_{A_2} \quad \text{if} \quad i > A_2 \tag{6}$$

$$1 = \sum_{i=0}^{\infty} q_i, \qquad \text{and} \qquad q_0 = \sum_{i=1}^{\infty} q_i \min\{i-1, A_1 - 1\}.$$

3. *There are positive constants* c_1 *and* C_1*, independent of* A_1 *and* A_2*, such that*

$$c_1 k^{-(w+1)} < q_k/q_1 < C_1 k^{-(w+1)} \tag{7}$$

for $1 \leq k \leq A_2$.

4. *If* $A_1 = A_2$*, the parameter* w *is equal to* 1*, and for general* A_1 *and* A_2*, it decreases with increasing* A_1*, and increases with increasing* A_2.

Equation (7) clearly defines a power law degree distribution with exponent $\gamma = w + 1$ for $k \leq A_2$. Note that for measurements of the Internet the value of the exponent for the power law is $\gamma \approx 2$. In our border toll optimization model, where $A_1 = A_2$, we recover $\gamma = 2$.

The convergence claim of Theorem 2 is proved using a novel method which we believe is one of the main technical contributions of this work. For preferential attachment models which have been analyzed in the past [1,6,8,10], the convergence was established using the Azuma-Hoeffding martingale inequality. To establish the bounded-differences hypothesis required by that inequality, those proofs employed a clever coupling of the random decisions made by the various edges, such that the decisions made by an edge e only influence the decisions of subsequent edges which choose to imitate e's choices. A consequence of this coupling is that if e made a different decision, it would alter the degrees of only finitely many vertices. This in turn allows the required bounded-differences hypothesis to be established. No such approach is available for our models, because the coupling fails. The random decisions made by an edge e may influence the time at which some node v crosses the fertility or aging threshold, which thereby exerts a subtle influence on the decisions of *every* future edge, not only those which choose to imitate e.

Instead we introduce a new approach based on the second moment method. The argument establishing the requisite second-moment upper bound is quite subtle; it depends on a computation involving the eigenvalues of a matrix describing the evolution of the degree sequence in a continuous-time version of the model. The details are presented in the full version of this paper. Here we consider only the evolution of the *expected* degree sequence, see Sec. 3.

2 Equivalence of the Two Models

2.1 Basic Properties of the Border Toll Optimization Process

In this section we will turn to the BTOP defined in the introduction, establishing some basic properties which will enable us to prove that it is equivalent to the competition-induced preferential attachment model. In order to avoid complications we exclude the case that some of the x_i's are identical, an event that has probability zero. We say that $j \in \{0, 1 \ldots, t\}$ lies to the right of $i \in \{0, 1 \ldots, t\}$ if $x_i < x_j$, and we say that j lies directly to the right of i if $x_i < x_j$ but there is no $k \in \{1, \ldots, t\}$ such that $x_i < x_k < x_j$. In a similar way, we say that j is the first vertex with a certain property to the right of i if j has that property and

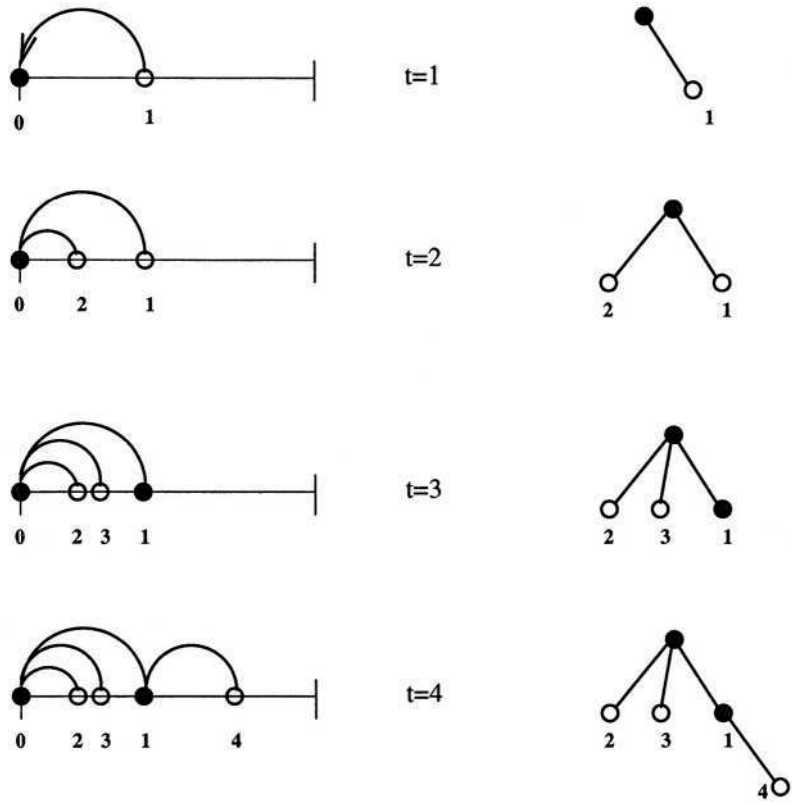


Fig. 1. A sample instance of BTOP for $A = 3$, showing the process on the unit interval (on the left), and the resulting tree (on the right). Fertile vertices are shaded, infertile ones are not. Note that vertex 1 became fertile at $t = 3$.

there exists no $k \in \{1, \ldots, t\}$ such that $x_i < x_k < x_j$ and k has the property in question. Similar notions apply with "left" in place of "right". The following definition and lemma are illustrated in Fig. 1.

Definition 3 *A vertex* i *is called* fertile at time t *if a hypothetical new point arriving at time* $t+1$ *and landing directly to the right of* x_i *would attach itself to the node* i. *Otherwise* i *is called* infertile *at time* t.

Lemma 1. *Let* $0 < \alpha < \infty$, *let* $A = \lceil \alpha^{-1} \rceil$, *and let* $0 < t < \infty$. *Then*

i) The node 0 *is fertile at time* t.

ii) Let i *be fertile at time* t. *If* i *is the rightmost fertile vertex at time* t *(case 1), let* ℓ *be the number of infertile vertices to the right of* i. *Otherwise (case 2), let* j *be the next fertile vertex to the right of* i, *and let* $\ell = n_{ij}(t)$. *Then* $0 \le \ell \le A - 1$, *and the* ℓ *infertile vertices located directly to the right of* i *are children of* i. *In case 2, if* $h_j > h_i$, *then* j *is a fertile child of* i *and* $\ell = A - 1$. *As a consequence, the hop count between two consecutive fertile vertices never*

increases by more than 1 *as we move to the right, and if it increases by* 1, *there are* $A - 1$ *infertile vertices between the two fertile ones.*

iii) Assume that the new vertex at time $t+1$ *lands between two consecutive fertile vertices* i *and* j, *and let* $\ell = n_{ij}(t)$. *Then* $t+1$ *becomes a child of* i. *If* $\ell + 1 < A$, *the new vertex is infertile at time* $t+1$, *and the fertility of all old vertices is unchanged. If* $\ell + 1 = A$ *and the new vertex lies directly to the left of* j, *the new vertex is fertile at time* $t+1$ *and the fertility of the old vertices is unchanged. If* $\ell + 1 = A$ *and the new vertex does not lie directly to the left of* j, *the new vertex is infertile at time* $t+1$, *the vertex directly to the left of* j *becomes fertile, and the fertility of all other vertices is unchanged.*

iv) If $t+1$ *lands to the right of the rightmost fertile vertex at time* t, *the statements in Hi) hold with* j *replaced by the right endpoint of* [0,1], *and* $n_{ij}(t)$ *replaced by the number of vertices to the right of* i.

v) If i *is fertile at time* t, *it is still fertile at time* $t+1$.

vi) If i *has* k *children at time* t, *the* $\ell = \min\{A-1, k\}$ *leftmost of them are infertile at time* t, *and any others are fertile.*

Proof. Statement i) is trivial, statement v) follows immediately from iii) and iv), and vi) follows immediately from ii). So we are left with ii) — iv). We proceed by induction on t. If ii) holds at time t, and iii) and iv) hold for a new vertex arriving at time $t+1$, ii) clearly also holds at time $t+1$. We therefore only have to prove that ii) at time t implies iii) and iv) for a new vertex arriving at time $t+1$. Using, in particular, the last statement of ii) as a key ingredient, the proof is straightforward but lengthy. This will appear in the full version of the paper. □

2.2 Proof of Theorem 1

In the BTOP, note that our cost function $\min_j [\alpha n_{tj}(t) + h_j(t-1)]$, and hence the graph $G(t)$, only depends on the order of the vertices $x_0, \ldots, x_t$, and not on their actual positions in the interval [0,1]. Let $\pi(t)$ be the permutation of $\{0, 1, \ldots, t\}$ which orders the vertices $x_0, \ldots, x_t$ from left to right, so that

$$x_0 = x_{\pi_0(t)} < x_{\pi_1(t)} < \cdots < x_{\pi_t(t)}. \tag{8}$$

(Recall that the vertices $x_0, x_1, \ldots, x_t$ are pairwise distinct with probability one.) We can consider a change of variables, from the x's to the length of the intervals between successive ordered vertices:

$$s_i(t) \equiv x_{\pi_{i+1}(t)} - x_{\pi_i(t)} \quad \text{if} \quad 0 \le i \le t-1 \quad \text{and} \quad s_t(t) = 1 - x_{\pi_t(t)}. \tag{9}$$

The lengths then obey the constraint: $\sum_{i=0}^{t} s_i = 1$. The set of interval lengths, $s(t)$ together with the set of permutation labels $\pi(t) = (\pi_0(t), \pi_1(t), \ldots, \pi_t(t))$ is an equivalent representation to the original set of position variables, $x(t)$.

Let us consider the process $\{\pi(t)\}_{t \ge 1}$. It is not hard to show that this process is a Markov process, with the initial permutation being the trivial permutation given by $\pi_i(1) = i$, and the permutation at time $t+1$ obtained from $\pi(t)$ by

inserting the new point $t+1$ into a uniformly random position. More explicitly, the permutation $\pi(t+1)$ is obtained from $\pi(t)$ by choosing $i_o \in \{1,\dots,t+1\}$ uniformly at random, and setting

$$\pi_i(t+1) = \begin{cases} \pi_i(t) & \text{if} \quad i \le i_0 \\ t+1 & \text{if} \quad i = i_0 \\ \pi_{i-1}(t) & \text{if} \quad i > i_0. \end{cases} \tag{10}$$

Indeed, let $I_k(t) = [x_{\pi_k(t)}, x_{\pi_{k+1}(t)}]$, and consider for a moment the process $(\pi(t), s(t))$. Then the conditional probability that the next point arrives in the k-th interval, I_k, depends only on the interval length at time t:

$$\begin{aligned} Pr\,[x_{t+1} \in I_k \,|\pi(t), s(t), \pi(t-1), s(t-1), \dots, \pi(0), s(0)] \\ = Pr\,[x_{t+1} \in I_k \,|\pi(t), s(t)] = s_k(t). \end{aligned} \tag{11}$$

Integrating out the dependence on the interval length from the above equation we get:

$$\begin{aligned} Pr\,[x_{t+1} \in I_k \,|\pi(t)] &= \int Pr\,[x_{t+1} \in I_k \,|\pi(t), s(t)]\, dP(s(t)) \\ &= \int s_k(t) dP(s(t)) = \frac{1}{t+1}, \end{aligned} \tag{12}$$

since after the arrival of t points, there exist $(t+1)$ intervals, and by symmmetry they have equal expected length. Thus the probability that the next point arrives in the k-th interval is uniform over all the intervals, proving that $\pi(t)$ is indeed a Markov chain with the transition probabilities described above.

With the help of Lemma 1, we now easily derive a description of the graph $G(t)$ which does not involve any optimization problem. To this end, let us consider a vertex i with ℓ infertile children at time t. If a new vertex falls into the interval directly to the right of i, or into one of the intervals directly to the right of an infertile child of i, it will connect to the vertex i. Since there is a total of $t+1$ intervals at time t, the probability that a vertex i with ℓ infertile children grows an offspring is $(\ell+1)/(t+1)$. By Lemma 1 (vi), this number is equal to $\min\{A, k_i\}/(t+1)$, where $k_i - 1$ is the number of children of i. Note that fertile children do not contribute to this probability, since vertices falling into an interval directly to the right of a fertile child will connect to the child, not the parent.

Assume now that i did get a new offspring, and that it had $A-1$ infertile children at time t. Then the new vertex is either born fertile, or makes one of its infertile siblings fertile. Using the principle of deferred decisions, we may assume that with probability $1/A$ the new vertex becomes fertile, and with probability $(A-1)/A$ an old one, chosen uniformly at random among the $A-1$ candidates, becomes fertile.

This finishes the proof of Theorem 1.

3 Convergence of the Degree Distribution

3.1 Overview

To characterize the behavior of the degree distribution, we derive a recursion which governs the evolution of the vector $\boldsymbol{N}(t)$, whose components are the number of vertices of each degree, at the time when there are t nodes in the network. The conditional expectation of $\boldsymbol{N}(t+1)$ is given by an evolution equation of the form

$$\mathbb{E}\left(\boldsymbol{N}(t+1) - \boldsymbol{N}(t) \mid \boldsymbol{N}(t)\right) = M(t)\boldsymbol{N}(t),$$

where $M(t)$ depends on t through the random variable $W(t)$ introduced in Definition 2. Due to the randomness of the coefficient matrix $M(t)$, the analysis of this evolution equation is not straightforward. We avoid this problem by introducing a continuous-time process, with time parameter τ, which is equivalent to the original discrete-time process up to a (random) reparametrization of the time coordinate. The evolution equation for the conditional expectations in the continuous-time process involves a coefficient matrix M that is not random and does not depend on τ. We will first prove that the *expected* degree distribution in the continuous-time model converges to a scalar multiple of the eigenvector $\hat{p}$ of M associated with the largest eigenvalue w. This is followed by the much more difficult proof that the *empirical* degree distribution converges a.s. to the same limit. Finally, we translate this continuous-time result into a rigorous convergence result for the original discrete-time system.

The key observation is that, in this continuous-time model, the number of vertices of degree k, $\widehat{N}_k(\tau)$, grows exponentially at a rate determined by the largest eigenvalue of this matrix, w, while the difference $q_j\widehat{N}_k(\tau) - q_k\widehat{N}_j(\tau)$ has an exponential growth rate which is at most the second eigenvalue; for the matrix in question this is strictly less than w. This guarantees that the ratio $\widehat{N}_k(\tau)/\widehat{N}_j(\tau)$ will converge almost surely to q_k/q_j, for all k and j. The convergence of the normalized degree sequence to the vector $(q_i)_{i=0}^{\infty}$ in the continuous-time model follows easily from this. We then translate this continuous-time result into a rigorous convergence result for the original discrete-time system.

3.2 Notation

Let $A \geq \max(A_1, A_2)$. Let $N_0(t)$ be the number of infertile vertices at (discrete) time t, and, for $k \geq 1$, let $N_k(t)$ be the number of fertile vertices with $k-1$ children at time t. Let $\widetilde{N}_A(t) = N_{\geq A}(t) = \sum_{k \geq A} N_k(t)$, let $\widetilde{N}_k(t) = N_k(t)$ if $k < A$, and let $W(t) = \sum_{k=1}^{A} \min\{k, A_2\}\widetilde{N}_k(t)$ be the combined attractiveness of all vertices. Let $n_k(t) = \frac{1}{t+1}N_k(t)$ and $\tilde{n}_k(t) = \frac{1}{t+1}\widetilde{N}_k(t)$. Finally, the vectors $(\widetilde{N}_k(t))_{k=1}^{A}$ and $(\tilde{n}_k(t))_{k=1}^{A}$ are denoted by $\widetilde{N}(t)$ and $\tilde{n}(t)$ respectively. Note that the index k runs from 1 to A, not 0 to A.

3.3 Evolution of the Expected Value

From the definition of the generalized preferential attachment model, it is easy to derive the probabilities for the various alternatives which may happen upon the arrival of the $(t+1)$-st node:

- With probability $A_2\widetilde{N}_A(t)/W(t)$, it attaches to a node of degree $\geq A$. This increments $\widetilde{N}_1$, and leaves $\widetilde{N}_A$ and all $\widetilde{N}_j$ with $1 < j < A$ unchanged.
- With probability $\min(A_2, k)\widetilde{N}_k(t)/W(t)$, it attaches to a node of degree k, where $1 \leq k < A$. This increments $\widetilde{N}_{k+1}$, decrements $\widetilde{N}_k$, increments $\widetilde{N}_0$ or $\widetilde{N}_1$ depending on whether $k < A_1$ or $k \geq A_1$, and leaves all other $\widetilde{N}_j$ with $j < A$ unchanged.

It follows that the discrete-time process $(\widetilde{N}_k(t))_{k=0}^A$ at time t is equivalent to the state of the following continuous-time stochastic process $(\widehat{N}_k(\tau))_{k=0}^A$ at the random stopping time $\tau = \tau_t$ of the t-th event.

- With rate $A_2\widehat{N}_A(\tau)$, $\widehat{N}_1$ increases by 1.
- For every $0 < k < A$, with rate $\widehat{N}_k(\tau)\min(k, A_2)$, the following happens:

$$\widehat{N}_k \to \widehat{N}_k - 1 \quad ; \quad \widehat{N}_{k+1} \to \widehat{N}_{k+1} + 1 \quad ; \quad \widehat{N}_{g(k)} \to \widehat{N}_{g(k)} + 1$$

  where $g(k) = 0$ for $k < A_1$ and $g(k) = 1$ otherwise.

Note that the above rules need to be modified if $A_1 = 1$; here the birth of a child of a degree-one vertex does not change the net number of fertile degree-one vertices, N_1. Let M be the following $A \times A$ matrix:

$$M_{i,j} = \begin{cases} -1 & \text{if } i = j = 1 < A_1 \\ -\min(j, A_2) & \text{if } 2 \leq i = j \leq A - 1 \\ \min(j, A_2) & \text{if } 2 \leq i = j + 1 \leq A \\ \min(j, A_2) & \text{if } i = 1 \text{ and } j \geq \max(A_1, 2) \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Then, for the continuous time process, for every $\tau > \sigma$, the conditional expectations of the vector $\widehat{N}(\tau) = (\widehat{N}_k(\tau))_{k=1}^A$ are given by

$$\mathbb{E}\left(\widehat{N}(\tau)|\widehat{N}(\sigma)\right) = e^{(\tau-\sigma)M}\widehat{N}(\sigma). \tag{14}$$

It is easy to see that the matrix e^M has all positive entries, and therefore (by the Perron-Frobenius Theorem) M has a unique eigenvector $\hat{p}$ of ℓ_1-norm 1 having all positive entries. Let w be the eigenvalue corresponding to $\hat{p}$. Then w is real, it has multiplicity 1, and it exceeds the real part of every other eigenvalue. Therefore, for every non-zero vector y with non-negative entries,

$$\lim_{\tau\to\infty} e^{-\tau w}e^{\tau M}y = \langle \hat{a}, y\rangle \hat{p}$$

where $\hat{a}$ is the eigenvector of M^{T} corresponding to w. Note that $\langle \hat{a}, y \rangle > 0$ because y is non-zero and non-negative, and $\hat{a}$ is positive, again by Perron-Frobenius. Therefore, the vector $\mathbb{E}\left(e^{-\tau w}\widehat{N}(\tau)\right)$ converges to a positive scalar multiple of $\hat{p}$, say $\lambda\hat{p}$, as $\tau \to \infty$. Note that this implies, in particular, that $w > 0$. We can also show that $w \leq 2$ by showing that $\|\widehat{N}(\tau)\| = \sum_{k=1}^{A} \widehat{N}_k(\tau)$ is stochastically dominated by the following process, known as the standard birth process X_τ, for which $\mathbb{E}(X_\tau) \sim e^{2\tau}$: X increases by one with rate $2X$ (a more precise definition with proof of all facts used here will come in the full version).

Intuitively, it should be clear that in the discrete time version, $\mathbb{E}\left(\frac{1}{t+1}\widetilde{N}(t)\right)$ converges to $\lambda\hat{p}$ as well. As it turns out, this does not follow immediately, and we establish it in a somewhat round-about way. After we show almost sure convergence to $\lambda\hat{p}$ in continuous time, almost sure convergence in the discrete time model follows once one shows that a.s., t is finite for all finite τ. Then, the a.s. convergence in the discrete time model yields convergence of the expected value in discrete time.

4 Power Law with a Cutoff

In the previous section, we saw that for every $A > \max\{A_1, A_2\}$, the limiting proportions up to $A - 1$ are $\lambda\hat{p}$ where $\hat{p}$ is the eigenvector corresponding to the highest eigenvalue w of the A-by-A matrix M defined in Eqn. 13. Therefore, the components $p_1, p_2, \ldots, p_A$ of the vector $\hat{p}$ satisfy the equation:

$$wp_i = -\min(i, A_2)p_i + \min(i-1, A_2)p_{i-1} \qquad i \geq 2 \tag{15}$$

where the normalization is determined by $\sum_{i=1}^{A} p_i = 1$. From (15) we get that for $i \leq A_2$,

$$p_i = \left(\prod_{k=2}^{i} \frac{k-1}{k+w}\right) p_1 \tag{16}$$

and for $i > A_2$

$$p_i = \left(\frac{A_2}{A_2+w}\right)^{i-A_2} p_{A_2} \tag{17}$$

Clearly, (17) is exponentially decaying. There are many ways to see that (16) behaves like a power-law with degree $1+w$. The simplest would probably be:

$$\frac{p_i}{p_1} = \left(\prod_{k=2}^{i} \frac{k-1}{k+w}\right) = \exp\left(\sum_{k=2}^{i} \log\left(\frac{k-1}{k+w}\right)\right) \tag{18}$$

$$= \exp\left(\sum_{k=2}^{i}\left(\frac{-1-w}{k+w}\right) + O(1)\right) = \exp\left((-1-w)\left(\sum_{k=2}^{i}(k+w)^{-1}\right) + O(1)\right)$$

$$= \exp\left((-1-w)\left(\sum_{k=2}^{i} k^{-1}\right) + O(1)\right) = \exp\left((-1-w)\left(\sum_{k=2}^{i} \log\left(\frac{k+1}{k}\right)\right) + O(1)\right)$$

$$= \exp\left((-1-w)\log(i/2) + O(1)\right) = O(1)i^{-1-w}.$$

Note that the constants implicit in the $O(\cdot)$ symbols do not depend on A_1, A_2 or i, due the fact that $0 < w \leq 2$. (18) can be stated in the following way:

Proposition 3 *There exist $0 < c < C < \infty$ such that for every A_1, A_2 and $i \leq A_2$, if $w = w(A_1, A_2)$ is as in (15), then*

$$ci^{-1-w} \leq \frac{p_i}{p_1} \leq Ci^{-1-w}. \tag{19}$$

The vector $(q_1, q_2, \ldots, q_{A-1})$ is a scalar multiple of the vector $(p_1, p_2, \ldots, p_{A-1})$, so equations (5), (6), and (7) in Theorem 2 (and the comment immediately following it) are consequences of equations (16), (17), and (19) derived above. It remains to prove the normalization conditions

$$\sum_{i=0}^{\infty} q_i = 1; \qquad q_0 = \sum_{i=1}^{\infty} q_i \min(i-1, A_1 - 1)$$

stated in Theorem 2. These follow from the equations

$$\sum_{i=0}^{\infty} N_i(t) = t + 1; \qquad N_0(t) = \sum_{i=1}^{\infty} N_i(t) \min(i-1, A_1 - 1).$$

The first of these simply says that there are $t+1$ vertices at time t; the second equation is proved by counting the number of infertile children of each fertile node.

The proof of the monotonicity properties of w asserted in part 4 of Theorem 2 is deferred to the full version of this paper.

References

1. W. Aiello, F. Chung, and L. Lu. Random evolution of massive graphs. In *Handbook of Massive Data Sets,* pages 97–122. Kluwer, 2002.
2. R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.,* 74:47–97, 2002.
3. D. J. Aldous. A stochastic complex network model. *Electron. Res. Announc. Amer. Math. Soc.,* 9:152–161, 2003.
4. A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science,* 286:509–512, 1999.
5. N. Berger, B. Bollobás, C. Borgs, J. T. Chayes, and O. Riordan. Degree distribution of the FKP network model. In *International Colloquium on Automata, Languages and Programming,* 2003.
6. B. Bollobás, C. Borgs, J. Chayes, and O. Riordan. Directed scale-free graphs. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms,* pages 132–139, 2003.
7. B. Bollobás and O. Riordan. Mathematical results on scale-free random graphs. In *Handbook of Graphs and Networks,* Berlin, 2002. Wiley-VCH.
8. B. Bollobás, O. Riordan, J. Spencer, and G. E. Tusnady. The degree sequence of a scale-free random graph process. *Random Structures and Algorithms,* 18:279–290, 2001.

9. J. M. Carlson and J. Doyle. Highly optimized tolerance: a mechanism for power laws in designed systems. *Phys. Rev. E,* 60:1412, 1999.
10. C. Cooper and A. M. Frieze. A general model of web graphs. In *Proceedings of 9th European Symposium on Algorithms,* pages 500–511, 2001.
11. S. N. Dorogovtsev and J. F. F. Mendes. Evolution of networks. *Adv. Phys.,* 51:1079, 2002.
12. F. Eggenberger and G. Pólya. Über die statistik verketteter. *Vorgänge.* Zeitschrift Agnew. Math. Mech., 3:279–289, 1923.
13. A. Fabrikant, E. Koutsoupias, and C.H. Papadimitriou. Heuristically optimized trade-offs: a new paradigm for power laws in the internet. In *International Colloquium on Automata, Languages and Programming,* pages 110–122, 2002.
14. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On the power-law relationships of the Internet topology. *Comput. Commun. Rev.,* 29:251, 1999.
15. R. Govindan and H. Tangmunarunkit. Heuristics for Internet map discovery. In *Proceedings of INFOCOM,* pages 1371–1380, 2000.
16. C. Kenyon and N. Schabanel. Personal communication.
17. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proc. 41st IEEE Symp. on Foundations of Computer Science,* pages 57–65, 2000.
18. M. E. J. Newman. The structure and function of complex networks. *SIAM Review,* 45:167–256, 2003.
19. D. J. de S. Price. A general theory of bibliometric and other cumulative advantage processes. *J. Amer. Soc. Inform. Sci.,* 27:292–306, 1976.
20. H. A. Simon. On a class of skew distribution functions. *Biometrika,* 42(3/4):425–440, 1955.
21. G. U. Yule. A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis. *Philos. Trans. Roy. Soc. London,* Ser. B 213:21–87, 1924.
22. G. K. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, Cambridge.MA, 1949.

Approximating Longest Directed Paths and Cycles

Andreas Björklund[1], Thore Husfeldt[1], and Sanjeev Khanna[2*]

[1] Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden.
thore@cs.lu.se
[2] Dept. of CIS, University of Pennsylvania, Philadelphia, PA 19104.
sanjeev@cis.upenn.edu

Abstract. We investigate the hardness of approximating the longest path and the longest cycle in directed graphs on n vertices. We show that neither of these two problems can be polynomial time approximated within $n^{1-\epsilon}$ for any $\epsilon > 0$ unless P = NP. In particular, the result holds for digraphs of constant bounded outdegree that contain a Hamiltonian cycle.

Assuming the stronger complexity conjecture that Satisfiability cannot be solved in subexponential time, we show that there is no polynomial time algorithm that finds a directed path of length $\Omega(f(n)\log^2 n)$, or a directed cycle of length $\Omega(f(n)\log n)$, for any nondecreasing, polynomial time computable function f in $\omega(1)$. With a recent algorithm for undirected graphs by Gabow, this shows that long paths and cycles are harder to find in directed graphs than in undirected graphs.

We also find a directed path of length $\Omega(\log^2 n/\log\log n)$ in Hamiltonian digraphs with bounded outdegree. With our hardness results, this shows that long directed cycles are harder to find than a long directed paths. Furthermore, we present a simple polynomial time algorithm that finds paths of length $\Omega(n)$ in directed expanders of constant bounded outdegree.

1 Introduction

Given an unweighted graph or digraph $G = (V, A)$ with $n = |V|$, the *Longest Path* problem is to find the longest sequence of distinct vertices $v_1 \cdots v_k$ such that $v_i v_{i+1} \in A$. This problem is notorious for the difficulty of understanding its approximation hardness [4]. The present paper establishes a number of upper and lower bounds for the *directed* case.

The best known polynomial time algorithms for directed graphs essentially find such structures of logarithmic length. More precisely, Alon, Yuster, and Zwick find [1] a dipath or dicycle of length exactly $c\log n$ for any constant c, provided it exists, and Gabow and Nie [7] find a dicycle of length $\log n/\log\log n$, provided it exists (such a cycle may be far longer than logarithmic).

In the present paper we show that this problem is hard to approximate. Specifically, Theorem 1 states that in directed graphs the length of the longest

* Supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Career Award CCR-0093117.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 222–233, 2004.

path cannot be polynomial time approximated within an approximation ratio of $n^{1-\epsilon}$ for any $\epsilon > 0$ unless P = NP.

We can claim a stronger bound if we make a stronger assumption called the *Exponential Time Hypothesis* (ETH), namely that Satisfiability has no subexponential time algorithms [8]. Our Theorem 2 states that if we could find a dipath of length $f(n)\log^2 n$ efficiently (for some polynomial time computable and nondecreasing f in $\omega(1)$), then there would be an deterministic algorithm for 3-Sat with s variables with running time $2^{o(s)}$, violating ETH. This is relevant to the remaining open question in [1]: "Is there a polynomial time algorithm for deciding if a given graph $G = (V, E)$ contains a path of length, say, $\log^2 n$?" Even though this question remains open, Alon, Yuster, and Zwick's choice of time bound was not as capricious as their wording may suggest: any stronger algorithm than $\log^2 n$ for Longest Dipath would be at variance with the Exponential Time Hypothesis.

Undirected Graphs versus Directed Graphs. Our hardness results under ETH are of further interest in the light of a very recent result of Gabow [6] for the undirected case, which shows how to find superpolylograthmic paths and cycles. More precisely, if the graph contains a cycle of length l through a given vertex v, then [6] finds a cycle through v of length at least $(\log l)^{c \log\log l}$ for some constant $c > 0$. (The same bound for Longest Path follows.)

This shows that paths and cycles in directed graphs are harder to find than in undirected graphs, proving (under ETH) a widely held belief.

Algorithm for Hamiltonian Digraphs. Our lower bound holds even if the input digraph is known to be Hamiltonian, which addresses the question to what extent knowledge of the presence of a long path helps in the search for one. We complement this by an algorithm in Theorem 3 to efficiently find paths of length $\Omega(\log^2 n/\log\log n)$ in Hamiltonian digraphs of constant bounded outdegree; this is close to our own $f(n)\log^2 n$ lower bound. The best previous upper bound [1] was $O(\log n)$.

Longest Path versus Longest Cycle. For the related longest *cycle* problem, where we also require $v_k v_1 \in A$, we essentially show that one cannot efficiently find a cycle of more than logarithmic length. To be precise, Theorem 2 shows that (under ETH) no polynomial time can find a cycle of length $\geq f(n)\log n$, for any nondecreasing, polynomial time computable function f in $\omega(1)$. This is no more that a factor $\log\log n$ short of the best known approximation algorithm: Recently, Gabow and Nie [7] gave a polynomial time algorithm to find a directed cycle of length $\geq \log n/\log\log n$ if one exists.

Moreover, together with the longest path guarantee $\Omega(\log^2 n/\log\log n)$ from Theorem 3, the lower bound separates the complexities of the Longest Path and Longest Cycle problems, at least for the directed, bounded outdegree, Hamiltonian case, and assuming ETH.

Long Paths in Sparse Expanders. In contrast to our worst-case inapproximability result, it is well known that almost every digraph contains a path of length $\Omega(n)$, and that this path is easy to find [3, Chap. 8]. Thus it would be interesting

to understand which natural classes of digraphs admit efficient longest path algorithms.

With Theorem 4 we observe that a very simple algorithm that always finds a path of length $\Omega(n)$ in a bounded-outdegree directed expander. This provides some insight into the structure of digraphs where long paths are hard to find: The hard instances construct in our lower bound proof have bounded outdegree as well, but can be seen to have very bad expansion properties (for any given size there is a vertex subset of that size with constant size seperators).

Related work. Among the canonical NP-hard problems, the *undirected* version of this problem has been identified as the one that is least understood [4]. However, a number of recent papers have established increasingly good approximation algorithms [14,2], culminating in the very recent result by Gabow [6] cited above. Even better bounds exist for restricted classes of graphs; for example, a recent result [4] finds cycles of length $O(l^\alpha)$ ($\alpha = \log_3 2$) in graphs of maximum degree 3.

However, it remains fair to say that in undirected graphs, the approximation hardness of Longest Path remains open. It has been conjectured [10] that the length of a longest path in undirected graphs cannot be approximated within n^α for some $\alpha > 0$ unless P = NP, a somewhat weaker bound than the one we prove for digraphs, but this is far from being proved: the quoted reference shows that the Longest Path is not in APX, and that no polynomial time algorithm can approximate the length of the longest path within $2^{\log^{1-\epsilon} n}$ for any $\epsilon > 0$ unless $\text{NP} \subseteq \text{DTIME}\left(2^{\log^{O(1/\epsilon)} n}\right)$.

Our lower bound uses a reduction to the k *Vertex Disjoint Paths* problem in digraphs. Thus there is no direct way to translate our argument to the undirected case, because the problem is known to be polynomially solvable for undirected graphs [12].

2 Preliminaries

We write uv for the arc (u, v). The vertex set V is sometimes identified with $\{1, 2, \ldots, n\}$. For a subset $W \subseteq V$ of the vertices of a graph G, we denote by $G[W]$ the graph induced by W.

Our proof starts with a reduction from a problem known to be NP-complete for over twenty years. In the k *Vertex Disjoint Paths* problem we are given a digraph G of order $n > 2k$, and we are asked whether there exists a set of k vertex disjoint paths in G such that the ith path connects vertex $2i - 1$ to vertex $2i$, for $i = 1, \ldots k$. This problem is NP-complete [5] even when $k = 2$. We need to modify this result slightly to see that it is valid even if we restrict the 'yes'-instances to be partitionable into two disjoint paths. To be precise, we define the *Two Vertex Disjoint Paths* problem (2VDP): given a digraph G of order $n \geq 4$, decide whether there exists a pair of vertex disjoint paths, one from 1 to 2 and one from 3 to 4. We study the *restricted* version of this problem (R2VDP), where the 'yes'-instances are guaranteed to contain two such paths that together exhaust all vertices of G. In other words, the graph G with the additional arcs 23 and 41 contains a Hamiltonian cycle through these arcs.

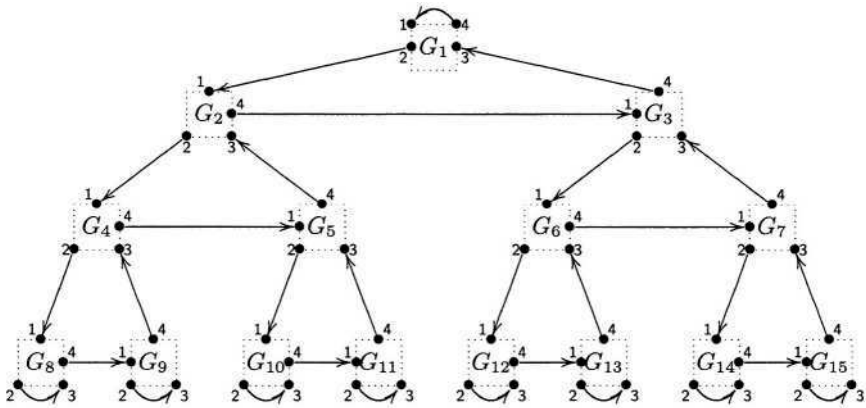


Fig. 1. $T_4[G]$.

Proposition 1 *Restricted Two Vertex Disjoint Paths is NP-complete.*

The proof is an extension of the construction in [5] and can be found in Sec. 7. It replaces a reduction from 3-Sat by a reduction from Monotone 1-in-3-Sat, and uses a more intricate clause gadget to guarantee the existence of two paths that cover all vertices. The modification is necessary to prove the lower bound for Longest Path even for Hamiltonian instances.

3 Long Paths Find Vertex Disjoint Paths

We will use instances of R2VDP to build graphs in which long paths must reveal a solution to the original problem. Given an instance $G = (V, A)$ of R2VDP, define $T_d[G]$ as a graph made up out of $m = 2^d - 1$ copies $G_1 \cdots G_m$ of G arranged in a balanced binary tree structure. For all $i < 2^{d-1}$, we say that the copies G_{2i} and G_{2i+1} are the left and right *child* of the copy G_i. The copy G_1 is the *root* of the tree, and G_i for $i \geq 2^{d-1}$ are the *leaves* of the tree. The copies of G in $T_d[G]$ are connected by additional arcs as follows. For every copy G_i having children, three arcs are added (cf. Fig. 1): One arc from 2 in G_i to 1 in G_{2i}, one arc from 4 in G_{2i} to 1 in G_{2i+1}, and one arc from 4 in G_{2i+1} to 3 in G_i. Moreover, in every leaf copy G_i $(i \geq 2^{d-1})$ we add the arc 23, and in the root G_1 we add the arc 41.

Lemma 1 *Given an instance $G = (V, A)$ of R2VDP on $n = |V|$ vertices, and any integers $m = 2^d - 1 > 3$, consider $T_d[G]$ with $N = mn$ vertices. Then*

- *If G has a solution then $T_d[G]$ contains a path of length $N - 1$.*
- *Given any path of length larger than $(4d - 5)n$ in $T_d[G]$, we can in time polynomial in N construct a solution to G.*

Proof. For the first part of the lemma, consider a solution for G consisting of two disjoint paths P and Q connecting 1 to 2 and 3 to 4, respectively, such that $P + 23 + Q + 41$ is a Hamiltonian cycle in G. The copies of P and Q in all G_is together with the added arcs constitute a Hamiltonian cycle in $T_d[G]$ of length mn and thus a path of the claimed length.

For the second part, first consider an internal copy G_i and observe that if a path traverses all of the four arcs connecting G_i to the rest of the structure then this path constitutes a solution to R2VDP for G. Thus we can restrict our attention to paths in $T_d[G]$ that avoid at least one the four external arcs of each internal G_i; we call such paths *avoiding*.

Given $T_d[G]$ define $e_d[G]$ as the length of the longest avoiding path in $T_d[G]$ ending in vertex 4 of its root copy, and $s_d[G]$ as the length of the longest avoiding path starting in vertex 1 of the root copy. Consider a path P ending in vertex 4 of the root copy, for $d > 1$. At most n vertices of P are in G_1. The path P has entered G_1 via vertex 3 from G_3's vertex 4. There are two possibilities. Either the first part of P is entirely in the subtree rooted at G_3, in which case P has length at most $n+e_{d-1}[G]$. Or it entered G_3 via 1 from the subtree rooted at G_2, in which case it may pass through at most n vertices in G_3, amounting to length at most $2n+e_{d-1}[G]$. (Especially, P cannot leave via G_3's vertex 2, because then it wouldn't be avoiding). A symmetric argument for $s_d[G]$ for $d > 1$ shows an equivalent relation. Thus we have that

$$e_1[G] \leq n, \quad e_{d+1}[G] \leq 2n + e_d[G],$$
$$s_1[G] \leq n, \quad s_{d+1}[G] \leq 2n + s_d[G].$$

Furthermore, note that a longest avoiding path in $T_d[G]$ connects a path amounting to $e_{d-1}[G]$ in the right subtree, through a bridge consisting of as many vertices as possible in the root, with a path amounting to $s_{d-1}[G]$ in the left subtree. Consequently, a typical longest avoiding path starts in a leaf copy of the right subtree, travels to its sister copy, goes up a level and over to the sister of that copy, continues straight up in this zigzag manner to the root copy, and down in the same fashion on the other side. Formally, the length of a longest avoiding path in $T_d[G]$ for $d > 1$ is bounded from above by $e_{d-1}[G] + n + s_{d-1}[G] \leq (4d-5)n$. □

Theorem 1 *There can be no deterministic, polynomial time approximation algorithm for Longest Path or Longest Cycle in a Hamiltonian directed graph on n vertices with performance ratio $n^{1-\epsilon}$ for any fixed $\epsilon > 0$, unless P = NP.*

Proof. First consider the path case. Given an instance $G = (V, A)$ of R2VDP with $n = |V|$, fix $k = 1/\epsilon$ and construct $T_d[G]$ for the smallest integers $m = 2^d - 1 \geq (4dn)^k$. Note that the graph $T_d[G]$ has order $N = n^{O(k)}$. Assume there is a deterministic algorithm finding a long path of length l_{apx} in time polynomial in N, and let l_{opt} denote the length of a longest path. Return 'yes' if and only if $l_{\text{apx}} > (4d-5)n$. To see that this works note that if G is a 'yes'-instance and if indeed $l_{\text{opt}}/l_{\text{apx}} \leq N^{1-\epsilon}$ then $l_{\text{apx}} > (4d-5)n$, so Lem. 1 gives a solution to G.

If on the other hand G is a 'no'-instance then the longest path must be *avoiding* as defined in the proof of Lem. 1, so its length is at most $(4d-5)n$. Thus we can solve the R2VDP problem in polynomial time, which by Prop. 1 requires P = NP.

For the cycle case, we may use a simpler construction. Simply connect m copies $G_1, \cdots, G_m$ of G on a string, by adding arcs from vertex 2 in G_i to vertex 1 in G_{i+1}, and arcs from vertex 4 in G_i to vertex 3 in G_{i-1}. Finally, add the arc 41 in G_1 and the arc 23 in G_m. The resulting graph has a cycle of length mn whenever G is a 'yes'-instance, but any cycle of size at least $2n+1$ must reveal a solution to G. □

4 Subexponential Algorithms for Satisfiability

In this section we show that good dipath and dicycle algorithms imply subexponential time algorithms for Satisfiability.

We need the well-known reduction from Monotone 1-in-3-Sat to 3-Sat. It can be verified that the number of variables in the construction (see also [11, Exerc. 9.5.3]) is not too large:

Lemma 2 ([13]) *Given a 3-Sat instance φ with s variables and r clauses we can construct an instance of Monotone 1-in-3-Sat with $O(r)$ clauses and variables that is satisfiable if and only if φ is.*

Lemma 3 *There is a deterministic algorithm for Monotone 1-in-3-Sat on r variables running in time $2^{o(r)}$, if there is*

1. *a polynomial time deterministic approximation algorithm A_{LP} for Longest Path in N-node Hamiltonian digraphs with guarantee $f(N)\log^2 N$, or*
2. *a polynomial time deterministic approximation algorithm A_{LC} for Longest Cycle in N-node Hamiltonian digraphs with guarantee $f(N)\log N$,*

where f is any polynomial time computable, nondecreasing function in $\omega(1)$.

Proof. We need to verify that our constructions obey the necessary size bounds. The R2VDP instance G build from the instance to Monotone 1-in-3-Sat described in Sec. 7 has size $n = O(r)$.

For the path case, set $d = 4n/f^{1/2}(n)$ and construct $T_d[G]$ as in Sec. 3. Observe that the entire construction will have $(2^d-1)n = 2^{o(n)} = 2^{o(r)}$ nodes. Running A_{LP} on a 'yes' instance instance will reveal a cycle of length $f\big(n(2^d-1)\big)\log^2\big(n(2^d-1)\big) \geq f(n)\log^2(2^{d/2}) \geq 4n^2 > (4d-5)n$, so Lem. 1 tells us how to use A_{LP} to solve the R2VDP instance, and hence the 1-in-3-Sat instance.

For the cycle case, choose the number of copies $m = 2^{3n/f(n)}$. Observe that the entire construction has size $mn = 2^{o(n)} = 2^{o(r)}$. Running A_{LC} on this graph will reveal a cycle of length $f(mn)\log(mn) \geq f(n)\log m = f(n)\cdot 3n/f(n) = 3n > 2n+1$, and the conclusion follows similarly to the proof of Theorem 1. □

Theorem 2 *There is a deterministic algorithm for 3-Sat on s variables running in time $2^{o(s)}$ if there is*

1. *a polynomial time deterministic approximation algorithm for Longest Path in N-node Hamiltonian digraphs with guarantee $f(N)\log^2 N$, or*
2. *a polynomial time deterministic approximation algorithm for Longest Cycle in N-node Hamiltonian digraphs with guarantee $f(N)\log N$,*

where f is any polynomial time computable, nondecreasing function in $\omega(1)$.

Proof. The previous two lemmas give an algorithm that runs in time $2^{o(r)}$, where r is the number of clauses in the input instance. This implies a $2^{o(s)}$-algorithm by the Sparsification Lemma of [9]. □

5 Finding Long Paths in Hamiltonian Digraphs

Vishwanathan [14] presents a polynomial time algorithm that finds a path of length $\Omega(\log^2 n/\log\log n)$ in *undirected* Hamiltonian graphs of constant bounded degree. We show in this section that with some modifications the algorithm and its analysis apply to the directed case as well.

Theorem 3 *There is a polynomial time algorithm always finding a path of length $\Omega(\log^2 n/\log\log n)$ in any Hamiltonian digraph of constant bounded outdegree on n vertices.*

To prove the theorem, we need some additional notation. Let $G = (V, A)$ be a digraph. We say that a vertex $v \in V$ *spans* the subgraph $G_v = G[V_v]$ where $V_v \subseteq V$ is the set of vertices reachable from v in G. Consider the algorithm below. It takes a digraph $G = (V, A)$ on $n = |V|$ vertices and a specified vertex $v \in V$ as input, and returns a long path starting in v.

1. Enumerate all paths in G starting in v of length $\log n$, if none return the longest found.
2. For each such path $P = (v, \cdots, w)$, let V_w be the set of vertices reachable from w in $G[V - P + \{w\}]$.
3. Compute a depth first search tree rooted at w in $G[V_w]$.
4. If the deepest path in the tree is longer than $\log^2 n$, return this path.
5. Otherwise, select the enumerated path P whose end vertex w spans as large a subgraph as possible after removal of $P - \{w\}$ from the vertex set, i.e the path maximising $|V_w|$.
6. Search recursively for a long path R starting from w in $G[V_w]$, and return $(P - \{w\}) + R$.

First note that the algorithm indeed runs in polynomial time. The enumeration of all paths of length $\log n$ takes no more than polynomial time since the outdegree is bounded by a constant k, and thus there cannot be more than $k^{\log n}$ paths. Computing a depth first search tree is also a polynomial time task, and it

is seen to be performed a polynomial number of times, since the recursion does not branch at all.

To prove that the length of the resulting path is indeed $\Omega(\log^2 n/\log\log n)$, we need to show that at each recursive call of the algorithm, there is still a long enough path starting at the current root vertex.

Lemma 4 *Let $G = (V, A)$ be a Hamiltonian digraph. Let $S \subseteq V, v \in V \setminus S$. Suppose that on removal of the vertices of S, v spans the subgraph $G_v = (V_v, A_v)$ of size t. If each vertex $w \in V_v$ is reachable from v on a path of length less than d, then there is a path of length $t/d|S|$ in G_v starting in v.*

Proof. Consider a Hamiltonian cycle C in G. The removal of S cuts C into at most $|S|$ paths $P_1 \cdots P_{|S|}$. Since each vertex in V lies on C, the subgraph G_v must contain at least $t/|S|$ vertices W from one of the paths, say P_j. In fact, G_v must contain a path of length $t/|S|$, since the vertex in W first encountered along P_j implies the presence in G_v of all the subsequent vertices on P_j, and these are at least $|W|$. Denote one such path by $P = p_0 \cdots p_{|W|-1}$, and let $R = r_0 \cdots r_{l-1}$ be a path from $r_0 = v$ to $r_{l-1} = p_0$, of length $l \leq d$. Set $s = |P \cap R|$ and enumerate the vertices on P from 0 to $|W|-1$ and let $i_1 \cdots i_s$ denote the indices of vertices in $P \cap R$, in particular $i_1 = 0$. Let $i_{s+1} = |W|$. An averaging argument shows that there exists j, such that $i_{j+1} - i_j \geq |W|/s$. Let q be the index for which $r_q = p_{i_j}$. The path along R from r_0 to r_q and continuing along P from p_{i_j+1} to $p_{i_{j+1}-1}$ has the claimed length. □

Observe that the algorithm removes no more than $\log n$ vertices from the graph at each recursive call. Thus, at call i we have removed at most $i \log n$ vertices from the original graph; the very same vertices constituting the beginning of our long path. Lemma 4 tells us that we still are in a position were it is possible to extend the path, as long as we can argue that the current end vertex of the path we are building spans large enough a subgraph. Note that whenever we stand at a vertex v starting a long path P of length $> \log n$ in step 1 of the algorithm, the path consisting of the first $\log n$ vertices of P is one of the paths of length $\log n$ being enumerated. This is our guarantee that the subgraph investigated at the next recursive call is not all that smaller than the graph considered during the previous one. It must consist of at least $|P| - \log n$ vertices. Of course, we cannot be sure that exactly this path is chosen at step 5, but this is of no concern, since it is sufficient for our purposes to assure that there are still enough vertices reachable.

Formally, let V_i denote the vertex set of the subgraph considered at the recursive call i. In the beginning, we know that regardless of the choice of start vertex v, we span the whole graph and thus $V_0 = V$, and furthermore, that a path of length n starts in v. Combining the preceding discussion with Lem. 4, we establish the following inequality for the only non-trivial case that no path of length $\log^2 n$ is ever found during step 4 of the algorithm:

$$|V_{i+1}| > \frac{|V_i|}{i \log^3 n} - \log n$$

It is readily verified that $|V_i| > 0$ for all $i < c \log n / \log \log n$ for some constant c, which completes the proof of Theorem. 3.

6 Finding Long Paths in Sparse Expanders

In this section we show that in a sparse *expander* graph, a relatively long path is easily found.

A digraph $G = (V, A)$ on n vertices is a **c-expander** if $|\delta U| \geq c(1 - \frac{|U|}{n})|U|$ for every subset $U \subset V$ where $\delta U = \{ v \notin U \mid \exists u \in U : uv \in A \}$. A standard probabilistic argument shows that with high probability a random digraphs with outdegree k $(k > 2)$, are **c_k-expanders** for some constant c_k, for large enough $n > n_k$.

We propose the following algorithm for finding a long path $p_0 \cdots p_l$ in a sparse expander.

1. Pick an arbitrary start vertex p_0, and set $i = 0$.
2. Let $G_i = (V_i, A_i)$ be the subgraph spanned by p_i in $G[V \setminus (\bigcup_{j=0}^{i-1} p_j)]$.
3. If G_i consists only of p_i, exit.
4. For each neighbour v of p_i in G_i, evaluate the size of the subgraph spanned by v in $G_i[V_i \setminus p_i]$.
5. Choose the neighbour who has the largest spanned subgraph as p_{i+1}.
6. Set $i = i + 1$ and goto 2.

Theorem 4 *The algorithm finds a path of length $\frac{c}{2(k+1)}n$ in every c-expander digraph $G = (V, A)$ with maximum outdegree k.*

Proof. Consider step i. Enumerate the neighbours of p_i in G_i as $r_1 \cdots r_{k'}$. Let $V_i[r_j]$ be the vertices reachable from r_j in $G_i[V_i - \{p_i\}]$. Now observe that the $V_i[r_j]$ either are very small or really large for small i, since the set of vertices outside $V_i[r_j]$ in G which are directly connected by an arc from a vertex in $V_i[r_j]$ must lie on the prefix path $p_0 \cdots p_i$ by definition, and there must be a lot of them because of the expander criterion. Specifically, when i is small, there must be a j for which $V_i[r_j]$ is large, since $k' \leq k$ and $\bigcup V_i[r_j] = V_i - \{p_i\}$. Observe that V_{i+1} is the largest $V_i[r_j]$, to obtain $|V_{i+1}| \geq n - \frac{2(i+1)}{c}$ whenever at least one $V_i[r_j]$ is too large to be a small subgraph, i.e. as long as $\frac{c(|V_i|-1)}{2k} \geq i + 1$, where we for the sake of simplicity have used the expansion factor $c/2$ which holds for all set sizes. Observing that $V_0 = n$, we may solve for the smallest i, when the inequality above fails to hold. This will not happen unless $i \geq \frac{c}{2(k+1)}n$, as promised. □

7 Proof of Proposition 1

We review the construction in [5], in which the *switch* gadget from Fig. 2 plays a central role. Its key property is captured in the following statement.

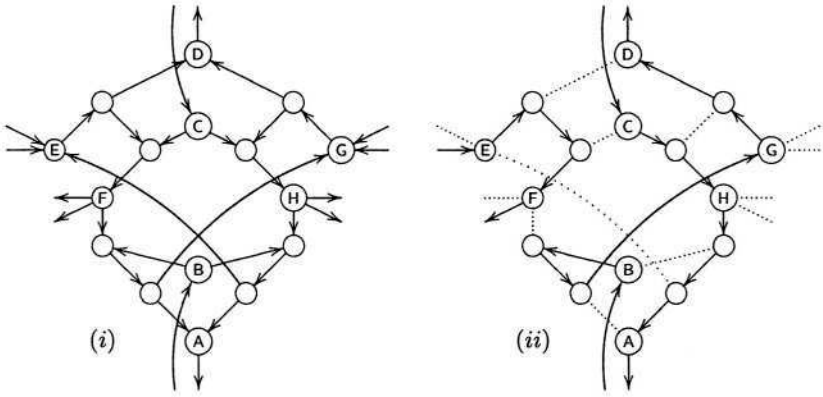


Fig. 2. (i) A switch. Only the labelled vertices are connected to the rest of the graph, as indicated by the arrows. (ii) Three vertex-disjoint paths through a switch.

Lemma 5 ([5]) *Consider the subgraph in Fig. 2. Suppose that are two vertex disjoint paths passing through the subgraph—one leaving at A and the other entering at B. Then the path leaving A must have entered at C and the path entering at B must leave at D. Furthermore, there is exactly one additional path through the subgraph and it connects either E to F or G to H, depending on the actual routing of the path leaving at A.*

Also, if one of these additional paths is present, all vertices are traversed.

To prove Prop. 1 we reduce from Monotone 1-in-3-Satisfiability, rather than 3-Satisfiability as used in [5]. An instance of 1-in-3-Sat is a Boolean expression in conjunctive normal form in which every clause has three literals. The question is if there is a truth assignment such that in every clause, exactly one literal is true. It is known that even when all literals are positive (*Monotone* 1-in-3-Sat) the problem is NP-complete [13].

Given such an instance φ with clauses $t_1, \ldots, t_m$ on variables $x_1, \ldots, x_n$, we construct and instance G_φ of R2VDP as follows.

Clause gadgets. Every clause t_i is represented by a gadget consisting of a vertex c_i and nine switches, three for every literal in t_i. Consider the clause $t_i = (x_1 \vee x_2 \vee x_3)$. The vertices c_i, c_{i+1} and the E and F vertices in the nine switches are connected as shown in Fig. 3. Thus all clause gadgets are connected on a string ending in a dummy vertex c_{m+1}.

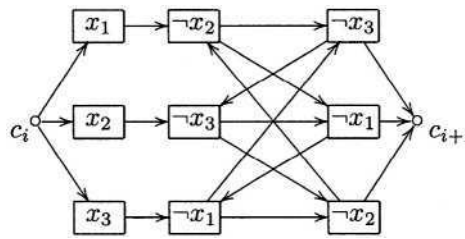


Fig. 3. A clause gadget consisting of 9 switches. Every incoming arc to a switch enters the switch's vertex E; every outgoing arc leaves the switch's vertex F.

The clause gadget has the following desirable properties: Call a path from c_i to c_{i+1} *valid* if it is consistent with a truth assignment to $\{x_1, x_2, x_3\}$ in the sense that if it passes through a switch labelled with a literal (like $\neg x_2$) then it cannot pass through its negation (like x_2). The following claims are easily verified:

Lemma 6 *Consider the construction in Fig. 3.*

1. *Every valid path from c_i to c_{i+1} corresponds to a truth assignment to $\{x_1, x_2, x_3\}$ that sets exactly one variable to true.*
2. *If there is a truth assignment to $\{x_1, x_2, x_3\}$ that sets exactly one variable to true then there is a valid path from c_i to c_{i+1} corresponding to the assignment. Moreover, there is such a valid path passing through all five switches whose labels are consistent with the assignment.*

Variable gadgets. Every variable x_i is represented by a vertex v_i. (Again, vertex v_{n+1} is a dummy vertex.) All switches in the clause gadgets representing the positive literal of the variable v_i are connected in series (the ordering of the switches on this string is not important): the vertex H in a switch is connected to vertex G of the next switch with the same label. Furthermore, there is an arc from v_i to vertex G in the first switch on its literal path, and an arc from vertex H in the last switch on the path to vertex v_{i+1}.

Likewise, all switches labelled with negated literals of this variable are connected. Thus there are two strings of switches leaving v_i: one contains all the positive literals, and one contains all the negated literals. Both end in v_{i+1}.

Also, *all* the switches are arranged on a path and connected by added arcs from vertex A in a switch to vertex C in the next one, and arcs back from vertex D in a switch to vertex B of the preceding switch. The ordering of the switches on this switch path is not important.

Finally, there is an arc from v_{n+1} to c_1 and an arc from vertex D in the first switch on the switch path to v_1.

To finish the construction of an instance of R2VDP it remains to identify the first four vertices. Vertex 1 is vertex B of the last switch on the switch path, vertex 2 is c_{m+1}, vertex 3 is vertex C of the first switch on the switch path, and vertex 4 is vertex A of the last switch on the switch path.

Lemma 7 *G_φ has two vertex disjoint paths from 1 to 2 and from 3 to 4 if and only if φ has a solution. Moreover, if G_φ contains such paths then it contains two such paths that together exhaust all its vertices.*

Proof. Assume φ can be satisfied so that exactly one variable in every clause is true. Walk through G_φ starting in vertex 1. This path is forced to traverse all switches until it reaches v_1. In general, assume that we reached v_i. To continue to v_{i+1} traverse the G–H paths of the string of negative literal switches if x_i is true; otherwise take the string of positive literal switches. Note that this forces us to avoid the E–F paths in these switches later.

Arriving at v_{n+1} continue to c_1. To travel from c_i to c_{i+1} we are forced to traverse the clause gadget of Fig. 3. Note that the truth assignment has set

exactly one of the variables to true, blocking the E–F path in the two switches labelled by its negative literal. Likewise, two of the variables are false, blocking the (two) switches labelled by their positive literal. The remaining five switches are labelled by the positive literal of the true variable or negative literals of the falsified variables. The valid path ensured by Lem. 6 passes through exactly these five switches.

Finally, the path arrives at $v_{m+1} = 2$. The path travelling from 3 to 4 is now unique. Observe that the two paths exhaust all the vertices and thus form a Hamiltonian cycle if we add 23 and 41.

Conversely, assume there are two paths from 1 to 2 and from 3 to 4. The subpaths connecting v_i to v_{i+1} ensure that all literal switches are consistent in the sense that if the E–F path in a switch labelled x_i is blocked then it is blocked in *all* such switches, and not blocked in any switch labelled $\neg x_i$. This forces the subpaths from c_i to c_{i+1} to be valid. Lem. 6 ensures that the corresponding truth assignment is satisfying and sets exactly one variable in each clause. □

Acknowledgements. The third author would like to express his thanks to Chandra Chekuri for many useful discussions on this problem. Hal Gabow suggested the formulation of the bound in Thm. 2.

References

1. N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM,* 42(4):844–856, 1995.
2. A. Björklund and T. Husfeldt. Finding a path of superlogarithmic length. *SIAM Journal on Computing,* 32(6):1395–1402, 2003.
3. Béla Bollobás. *Random graphs.* Cambridge University Press, 2nd edition, 2001.
4. T. Feder, R. Motwani, and C. Subi. Approximating the longest cycle problem in sparse graphs. *SIAM Journal on Computing,* 31(5): 1596–1607, 2002.
5. S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science,* 10:111–121, 1980.
6. H. N. Gabow. Finding paths and cycles of superlogarithmic length. In *Proc. 36th STOC,* 2004.
7. H. N. Gabow and S. Nie. Finding a long directed cycle. In *Proc. 15th SODA,* 2004.
8. R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and Systems Sciences,* 62(2):367–375, 2001.
9. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? In *Proc. 39th FOCS,* pages 653–663, 1998.
10. D. Karger, R. Motwani, and G.D.S. Ramkumar. On approximating the longest path in a graph. *Algorithmica,* 18(1):82–98, 1997.
11. C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.
12. N. Robertson and P. D. Seymour. Graph minors XIII: The disjoints paths problem. *J. Combinatorial Theory Ser. B,* 35, 1983.
13. T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th STOC,* pages 216–226, 1978.
14. S. Vishwanathan. An approximation algorithm for finding a long path in Hamiltonian graphs. In *Proc. 11th SODA,* pages 680–685, 2000.

Definitions and Bounds for Self-Healing Key Distribution Schemes*

Carlo Blundo, Paolo D'Arco, and Alfredo De Santis

Dipartimento di Informatica ed Applicazioni
Università degli Studi di Salerno, 84081, Baronissi (SA), Italy
{carblu,paodar,ads}@dia.unisa.it

Abstract. Self-healing key distribution schemes allow group managers to broadcast session keys to large and dynamic groups of users over unreliable channels. Roughly speaking, even if during a certain session some broadcast messages are lost due to network faults, the self-healing property of the scheme enables each group member to recover the key from the broadcast messages he/she has received before and after that session. Such schemes are quite suitable in supporting secure communication in wireless networks and mobile wireless ad-hoc networks. Recent papers have focused on self-healing key distribution, and have provided definitions and constructions. The contribution of this paper is the following:

- We analyse current definitions of self-healing key distribution and, for two of them, we show that no protocol can achieve the definition.
- We show that a lower bound on the size of the broadcast message, previously derived, does not hold.
- We propose a new definition of self-healing key distribution, and we show that it can be achieved by concrete schemes.
- We give some lower bounds on the resources required for implementing such schemes i.e., user memory storage and communication complexity. We prove that some of the bounds are tight.

1 Introduction

Self-healing key distribution. *Self-healing key distribution schemes,* recently introduced in [5], enable a dynamic group of users to establish a group key over an unreliable network. In such a scheme, a group manager, at the beginning of each session, in order to provide a key to each member of the group, sends packets over a broadcast channel. Every user, belonging to the group, computes the group key by using the packets and some private information. The group manager can start multiple sessions during a certain time-interval, by adding/removing users to/from the initial group. The main property of the scheme is that, if at the beginning of a certain session some broadcasted packet is lost, then users are still capable of recovering the group key for that session simply by using the packets they have received at the beginning of a previous session and the packets they will receive at the beginning of a subsequent one, without requesting additional

* Funded in part by the Network of Excellence ECRYPT EU-IST-2002-507932.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 234–245, 2004.

transmission from the group manager. Indeed, the only requirement that must be satisfied, in order for the user to recover the lost keys, is membership in the group both before and after the sessions in which the broadcast messages containing the key are sent and lost. Self-healing key distribution schemes are stateless and non-interactive, i.e., users do not need to update the secret information they receive in the setup phase, and they do not need to send any key-request message to the group manager. Some benefits of such an approach basically are: reduction of network traffic, reduction of the work load on the group manager, and a lower risk of user exposure through traffic analysis.

Applications. The relevance of self-healing key distribution has been well motivated in [5] and, later on, in [4]. Self-healing key distribution schemes can be used to achieve efficiently secure communication in wireless networks and mobile wireless ad-hoc networks. International peace operations and rescue missions, where there is no network infrastructure support and the adversary may intercept, modify, and/or partially interrupt the communication, are important applicative examples of cases in which reliability, confidentiality and authenticity of the communication is a major concern. In the above settings, all techniques developed for secure group communication in traditional networks might be used. However, some unique features of mobile and ad-hoc networks identify a new scenario: nodes/devices in mobile networks may move in and out of range frequently. Devices are powered by batteries. Hence, expensive computations like the ones required by public key cryptography are not suitable. In a battle field there could be a need for a rapid revocation of devices caught by the enemy and so on. All these aspects pose new challenges and the idea of self-healing key distribution can be of great benefit. Applications for self-healing key distribution can be also found in broadcast communication over low-cost channels: live-event transmissions (e.g., concerts, formal ceremonies, soccer games, ...) for users who have subscribed to (and paid for) the service. Electronic services delivering sensitive content/information to authorized recipients can take advantage from self-healing key distribution schemes as well. Hence, the spectrum of applicability is quite large.

Previous Work. Self-healing key distribution was introduced in [5]. Definitions, lower bounds on the resources required for implementing such schemes, and some constructions were provided. Later on, in [4], the definition given in [5], was generalised and more efficient constructions were presented. Finally, in [1], a slightly different definition was used, some efficient constructions were presented, and it was pointed that some of the constructions given in [5] have problems. The above papers have mainly considered unconditionally secure schemes. Some computationally secure constructions are given in [5,1]. Due to lack of space, we refer the interested reader to [5,4,1] for references to related works.

Our Contribution: In this paper we deal firstly with the *definitional task* of self-healing key distribution. We give some attention to the constructive task as well. We start by analysing the definition proposed in [5] and subsequently generalized in [4]. We discuss some issues related to such a formalization, and we show that no protocol can achieve some of the security requirements stated in [5,4]. Then,

we show that a lower bound on the size of the broadcast messages the group manager has to sent in order to establish session keys, proved in [5] and also used in [4], does not hold. After the analysis, we propose a new definition for self-healing key distribution, by extending and opportunely modifying the definition given in [5]. Subsequently, we give some lower bounds on the resources required for implementing such schemes, i.e., user memory storage and communication complexity, and we show that the bounds on user memory storage are tight.

2 Information Theory: Basic Notions

The *entropy* of a discrete random variable **X,** denoted by $H(\mathbf{X})$, is a real number that measures the uncertainty about the value of **X** when the underlying random experiment is carried out. It is defined by

$$H(\mathbf{X}) = -\sum_{x \in X} P_{\mathbf{X}}(x) \log P_{\mathbf{X}}(x),$$

assuming that the terms of the form $0 \log 0$ are excluded from the summation, and where the logarithm is relative to the base 2. The entropy satisfies $0 \leq H(\mathbf{X}) \leq \log |X|$, where $H(\mathbf{X}) = 0$ if and only if there exists $x_0 \in X$ such that $Pr(\mathbf{X} = x_0) = 1$; whereas, $H(\mathbf{X}) = \log |X|$ if and only if $Pr(\mathbf{X} = x) = 1/|X|$, for all $x \in X$. The deviation of the entropy $H(\mathbf{X})$ from its maximal value can be used as a measure of non-uniformity of the distribution $\{P_{\mathbf{X}}(x)\}_{x \in X}$.

Given two random variables **X** and **Y**, taking values on sets X and Y, respectively, according to a probability distribution $\{P_{\mathbf{XY}}(x, y)\}_{x \in X, y \in Y}$ on their Cartesian product, the conditional uncertainty of **X**, given the random variable **Y,** called *conditional entropy* and denoted by $H(\mathbf{X}|\mathbf{Y})$, is defined as

$$H(\mathbf{X}|\mathbf{Y}) = -\sum_{y \in Y} \sum_{x \in X} P_{\mathbf{Y}}(y) P_{\mathbf{X}|\mathbf{Y}}(x|y) \log P_{\mathbf{X}|\mathbf{Y}}(x|y).$$

Notice that the conditional entropy is not the entropy of a probability distribution but the *average* over all entropies $H(\mathbf{X}|\mathbf{Y} = y)$. Simple algebra shows that

$$H(\mathbf{X}|\mathbf{Y}) \geq 0 \tag{1}$$

with equality if and only if X is a function of Y.

The *mutual information* between **X** and **Y** is a measure of the amount of information by which the uncertainty about **X** is reduced by learning **Y,** and vice-versa. It is given by

$$I(\mathbf{X}; \mathbf{Y}) = H(\mathbf{X}) - H(\mathbf{X}|\mathbf{Y}) = H(\mathbf{Y}) - H(\mathbf{Y}|\mathbf{X}).$$

Since $I(\mathbf{X}; \mathbf{Y}) = I(\mathbf{Y}; \mathbf{X})$ and $I(\mathbf{X}; \mathbf{Y}) \geq 0$, it follows that

$$H(\mathbf{X}) \geq H(\mathbf{X}|\mathbf{Y}), \tag{2}$$

with equality if and only if $\mathbf{X}$ and $\mathbf{Y}$ are independent. Along the same lines, given three random variables, $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$, the *conditional mutual information* between $\mathbf{X}$ and $\mathbf{Y}$ given $\mathbf{Z}$ can be written as

$$I(\mathbf{X};\mathbf{Y}|\mathbf{Z}) = H(\mathbf{X}|\mathbf{Z}) - H(\mathbf{X}|\mathbf{Z},\mathbf{Y}) = H(\mathbf{Y}|\mathbf{Z}) - H(\mathbf{Y}|\mathbf{Z},\mathbf{X}) = I(\mathbf{Y};\mathbf{X}|\mathbf{Z}).$$

Since the conditional mutual information $I(\mathbf{X};\mathbf{Y}|\mathbf{Z})$ is always non-negative, it holds that

$$H(\mathbf{X}|\mathbf{Z}) \geq H(\mathbf{X}|\mathbf{Z},\mathbf{Y}). \tag{3}$$

The following lemmas are used in the proofs of our results.

Lemma 1. *Let* $\mathbf{X}$, $\mathbf{Y}$, *and* $\mathbf{Z}$ *be three random variables such that* $H(\mathbf{Z}|\mathbf{X},\mathbf{Y}) = 0$ *and* $H(\mathbf{Z}|\mathbf{Y}) = H(\mathbf{Z})$. *Then,* $H(\mathbf{X}|\mathbf{Y},\mathbf{Z}) = H(\mathbf{X}|\mathbf{Y}) - H(\mathbf{Z})$.

Lemma 2. *Let* $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ *and* $\mathbf{W}$ *be four random variables. If* $H(\mathbf{Y}|\mathbf{Z},\mathbf{W}) = 0$ *then* $H(\mathbf{X}|\mathbf{Z},\mathbf{W}) \leq H(\mathbf{X}|\mathbf{Y},\mathbf{W})$.

3 Self-Healing Key Distribution

Network Setting. Let $\mathcal{U}$ be the finite universe of users of a network. A broadcast unreliable channel is available, and time is defined by a global clock. Let GM be a group manager who sets up and manages, by means of join and revoke operations, a communication group, which is a dynamic subset of users of $\mathcal{U}$. Let $G_j \subseteq \mathcal{U}$ be the communication group established by GM in session j. Each user $U_i \in G_j$ holds a personal key S_i, received from GM before or when joining G_j. A personal key S_i can be seen as a sequence of elements from a finite set, and is "valid" as long as user U_i is not removed by GM from the group. Individual personal keys can be related.

We assume that GM can revoke at most t users during the lifetime of the scheme, and that once a user is revoked he/she is kept revoked. We denote the number of sessions, supported by the scheme, by m, the set of users revoked by GM up to session j by Rev_j, and the set of users who join the group in session j by $Join_j$. Hence, $G_j = (G_{j-1} \cup Join_j) \setminus Rev_j$. Moreover, for $j = 1, \ldots, m$, let K_j be the session key chosen by GM and communicated to the group members through a broadcast message, B_j. For each $U_i \in G_j$, the key K_j is determined by B_j and the personal key S_i.

Let $\mathbf{S}_i, \mathbf{B}_j, \mathbf{K}_j$ be the random variables representing the personal key of user U_i, the broadcast message B_j and the session key K_j for session j, respectively. Moreover, let $\mathbf{Z}_{i,j}$ be a random variable which represents information $Z_{i,j}$ that user U_i gets from the broadcast B_j and S_i.

The probability distributions according to whom the above random variables take values are determined by the key distribution scheme and the random bits used by GM. In particular, we assume that session keys K_j are chosen independently and according to the uniform distribution.

Definition. Using the entropy function, the following definition was stated:

Definition 1. *[Self-Healing Key Distribution Scheme with Revocation][5] Let $t, i \in \{1, \ldots, n\}$ be indices denoting, respectively, the maximum number of users that can be revoked by GM during the lifetime of the scheme and a generic user, and let $j \in \{1, \ldots, m\}$ be an index representing a session.*

1. *$\mathcal{D}$ is a session key distribution scheme if the following are true:*
   1.a) *For any member U_i, the key K_j is determined by $Z_{i,j}$.* Formally, it holds that:
   $$H(\mathbf{Z}_{i,j}|\mathbf{B}_j, \mathbf{S}_i) = 0 \quad \textit{and} \quad H(\mathbf{K}_j|\mathbf{Z}_{i,j}) = 0.$$
   1.b) *For any subset $F \subseteq \{U_1, \ldots U_n\}$, such that $|F| \leq t$ and $U_i \notin F$, the users in F cannot determine anything about S_i.* Formally, it holds that:
   $$H(\mathbf{S}_i|\{\mathbf{S}_{i'}\}_{U_{i'} \in F}, \mathbf{B}_1, \ldots, \mathbf{B}_m) = H(\mathbf{S}_i).$$
   1.c) *What members $U_1, \ldots, U_n$ learn from the broadcast B_j cannot be determined from the broadcast or personal keys alone.* Formally, it holds that:
   $$H(\mathbf{Z}_{i,j}|\mathbf{B}_1, \ldots, \mathbf{B}_m) = H(\mathbf{Z}_{i,j}|\mathbf{S}_1, \ldots, \mathbf{S}_n) = H(\mathbf{Z}_{i,j}).$$
2. *$\mathcal{D}$ has t-revocation capability if, given any set $R \subseteq \{U_1, \ldots, U_n\}$, where $|R| \leq t$, the group manager can generate a broadcast B_j such that, for all $U_i \notin R$, the user U_i can recover K_j but the revoked users cannot.* Formally, it holds that:
   $$H(\mathbf{K}_j|\mathbf{B}_j, \mathbf{S}_i) = 0, \quad \textit{while} \quad H(\mathbf{K}_j|\mathbf{B}_j, \{\mathbf{S}_{i'}\}_{U_{i'} \in R}) = H(\mathbf{K}_j).$$
3. *$\mathcal{D}$ is self-healing if, for any $1 \leq j_1 < j < j_2 \leq m$, the following properties are satisfied:*
   3.a) *For any U_i who is member in session j_1 and j_2, the key K_j is determined by $\{Z_{i,j_1}, Z_{i,j_2}\}$.* Formally, it holds that:
   $$H(\mathbf{K}_j|\mathbf{Z}_{i,j_1}, \mathbf{Z}_{i,j_2}) = 0.$$
   3.b) *For any two disjoint subsets $F, G \subset \{U_1, \ldots, U_n\}$, where $|F \cup G| \leq t$, the set $\{Z_{i',\ell}\}_{\{U_{i'} \in F, 1 \leq \ell \leq j_1\}} \cup \{Z_{i',\ell}\}_{\{U_{i'} \in G, j_2 \leq \ell \leq m\}}$, contains no information on K_j.* Formally, it holds that:
   $$H(\mathbf{K}_j|\{\mathbf{Z}_{i',\ell}\}_{\{U_{i'} \in F, 1 \leq \ell \leq j_1\}}, \{\mathbf{Z}_{i',\ell}\}_{\{U_{i'} \in G, j_2 \leq \ell \leq m\}}) = H(\mathbf{K}_j).$$

The definition is divided into three parts: the first and the second ones are quite easy to understand. The third one states the self-healing property and a security requirement that must hold against collusion attacks performed by coalitions of revoked and new users, who join the system in a certain session $j > 1$. More precisely, item 3.*a*) establishes that a user recovers, from two broadcast messages B_{j_1} and B_{j_2}, all session keys K_j, for $j_1 \leq j \leq j_2$. Item 3.*b*) essentially

requires that a group F of users, revoked in session j_1, and a group G of new users, who join the system in session j_2, by pooling together their personal keys and all broadcast messages, do not get any information about each key they are not entitled to receive.

Analysis. The above definition presents some problems: namely, there is *no protocol that can achieve all conditions.* We start by showing that conditions 1.*a*), 1.*b*), and 2 can be simultaneously satisfied only by a scheme where there is no uncertainty about the session keys! It turns out that the problem lies in condition 1.*b*). Indeed, condition 1.*a*) and 2 are required in order to define a basic scheme where users of the group can compute the session key and revoked users cannot. On the other hand, condition 1.*b*) implies a sort of a-posteriori security for the personal key, once given the broadcast message and the session key for a certain session i.e., $H(\mathbf{S}_i|\mathbf{K}_r, \mathbf{B}_r) = H(\mathbf{S}_i)$. Unfortunately, the proof of Theorem 1 implies that condition 1.*b*), given the other ones, holds if and only if $H(\mathbf{S}_i|\mathbf{B}_r) = H(\mathbf{S}_i)$ and $H(\mathbf{K}_r) = 0$. More precisely, we show the following result:

Theorem 1. *If conditions* 1.*a*), 1.*b*) *and* 2 *of Definition 1 hold then, for any* $1 \leq r \leq m$,

$$H(\mathbf{K}_r) = 0.$$

Proof. Let G_r be the communication group established in session r, for some $r \in \{1, \ldots, m\}$. Let F be any subset of $\mathcal{U}$ such that $|F| \leq t$, $F \cap G_r \neq \emptyset$, and $F \neq G_r$. Finally, let $U_i \in G_r \setminus F$. Assume that

$$H(\mathbf{K}_r|\mathbf{S}_i, \mathbf{B}_r) = 0, \text{ and } H(\mathbf{K}_r|\mathbf{B}_r) = H(\mathbf{K}_r).$$

Setting $\mathbf{X} = \mathbf{S}_i$, $\mathbf{Y} = \mathbf{B}_r$, and $\mathbf{Z} = \mathbf{K}_r$, and applying Lemma 1, it follows that $H(\mathbf{S}_i|\mathbf{B}_r, \mathbf{K}_r) = H(\mathbf{S}_i|\mathbf{B}_r) - H(\mathbf{K}_r)$. If condition 1.*b*) holds, we can show that $H(\mathbf{S}_i|\mathbf{B}_r, \mathbf{K}_r) = H(\mathbf{S}_i)$. Therefore, it must be $H(\mathbf{S}_i|\mathbf{B}_r) - H(\mathbf{K}_r) = H(\mathbf{S}_i)$, which holds if and only if $H(\mathbf{S}_i|\mathbf{B}_r) = H(\mathbf{S}_i)$ and $H(\mathbf{K}_r) = 0$.

Let us show the above assumptions and our claim. We start by proving that

$$H(\mathbf{K}_r|\mathbf{S}_i, \mathbf{B}_r) = 0, \text{ for any } U_i \in G_r. \tag{4}$$

From condition 1.*a*) of Definition 1, we have that $H(\mathbf{Z}_{i,j}|\mathbf{B}_r, \mathbf{S}_i) = 0$; hence, from Lemma 2, setting $\mathbf{X} = \mathbf{K}_r$, $\mathbf{Y} = \mathbf{Z}_{i,j}$, $\mathbf{Z} = \mathbf{B}_r, \mathbf{S}_i$, and $\mathbf{W}$ equals to the "empty" random variable, we get that $H(\mathbf{K}_r|\mathbf{B}_r, \mathbf{S}_i) \leq H(\mathbf{K}_r|\mathbf{Z}_{i,j})$. Since from condition 1.*a*) of Definition 1 it also holds that $H(\mathbf{K}_r|\mathbf{Z}_{i,j}) = 0$, applying (1), we have that

$$0 \leq H(\mathbf{K}_r|\mathbf{B}_r, \mathbf{S}_i) \leq H(\mathbf{K}_r|\mathbf{Z}_{i,j}) = 0.$$

Therefore, equality (4) is satisfied. To prove that $H(\mathbf{K}_r|\mathbf{B}_r) = H(\mathbf{K}_r)$, consider the following chain of equalities/inequalities.

$$\begin{aligned} H(\mathbf{K}_r) &= H(\mathbf{K}_r|\{\mathbf{S}_{i'}\}_{U_{i'} \in F}, \mathbf{B}_r) \text{ (from condition 2) of Definition 1)} \\ &\leq H(\mathbf{K}_r|\mathbf{B}_r) \text{ (applying property (3))} \\ &\leq H(\mathbf{K}_r) \text{ (applying property (2)).} \end{aligned}$$

Hence, $H(\mathbf{K}_r|\mathbf{B}_r) = H(\mathbf{K}_r)$. Finally, if condition 1.*b*) holds, then for $U_j \in F \cap G_r$ and $U_i \in G_r \setminus F$, it follows that $H(\mathbf{S}_i|\mathbf{S}_j, \mathbf{B}_r) = H(\mathbf{S}_i)$. Indeed:

$$\begin{aligned} H(\mathbf{S}_i) &= H(\mathbf{S}_i|\{\mathbf{S}_{i'}\}_{U_{i'} \in F}, \mathbf{B}_1, \ldots, \mathbf{B}_m) \text{ (from condition 1.}b\text{))} \\ &\leq H(\mathbf{S}_i|\mathbf{S}_j, \mathbf{B}_r) \text{ (applying property (3))} \\ &\leq H(\mathbf{S}_i) \text{ (applying property (2)).} \end{aligned}$$

At this point notice that, since (4) establishes that $H(\mathbf{K}_r|\mathbf{S}_j, \mathbf{B}_r) = 0$, from Lemma 2, setting $\mathbf{X} = \mathbf{S}_i$, $\mathbf{Y} = \mathbf{K}_r$, $\mathbf{Z} = \mathbf{S}_j$, and $\mathbf{W} = \mathbf{B}_r$, we get that $H(\mathbf{S}_i|\mathbf{S}_j, \mathbf{B}_r) \leq H(\mathbf{S}_i|\mathbf{K}_r, \mathbf{B}_r)$. Hence, applying (2), it holds that

$$H(\mathbf{S}_i) = H(\mathbf{S}_i|\mathbf{S}_j, \mathbf{B}_r) \leq H(\mathbf{S}_i|\mathbf{K}_r, \mathbf{B}_r) \leq H(\mathbf{S}_i),$$

i.e., $H(\mathbf{S}_i|\mathbf{K}_r, \mathbf{B}_r) = H(\mathbf{S}_i)$, and the theorem is proved. □

Notice that, the authors of [4] changed condition 1.*b*) of Definition 1. Indeed, as a side note, they pointed out that the schemes given in [5] do not meet such a condition, and a sketch of the reason was briefly provided[1]. They relaxed condition 1.*b*) and required:

> *For any subset $F \subseteq \mathcal{U}$, such that $|F| \leq t$, and for each $U_i \notin F$, the users in F have at least b bits of uncertainty about S_i.* Formally, it holds that:

$$H(\mathbf{S}_i|\{\mathbf{S}_{i'}\}_{U_{i'} \in F}, \mathbf{B}_1, \ldots, \mathbf{B}_m) \geq b. \tag{5}$$

In [4] a scheme satisfying condition (5) was presented. We notice that, given a scheme where the above condition is not satisfied, it is possible to construct a new scheme which does meet the condition still preserving all other conditions. Basically, for any b, the design strategy is to add in the new scheme b *random bits*, chosen independently of all other variables, to every $\mathbf{S}_i$. In such a case, it is easy to check that also condition (5) holds.

Definition 1 presents another problem: conditions 3.*a*) and 3.*b*) cannot be satisfied simultaneously. Consider the following situation. Let $F = \{U_s\}$ and $G = \{U_1, \ldots, U_{s-1}\}$ be two *generic* disjoint subsets of users, where $s \leq t$, and let $j_1 < j < j_2$. Condition 3.*b*) of Definition 1 implies that:

$$H(\mathbf{K}_j|\mathbf{Z}_{1,j_2}, \ldots, \mathbf{Z}_{s-1,j_2}, \mathbf{Z}_{s,j_1}) = H(\mathbf{K}_j), \tag{6}$$

while, if U_s belongs to G_{j_1} and is not revoked in session j_2, condition 3.*a*) implies that

$$H(\mathbf{K}_j|\mathbf{Z}_{s,j_2}, \mathbf{Z}_{s,j_1}) = 0. \tag{7}$$

Since the random variable $\mathbf{Z}_{i,j}$ is defined as the information user U_i gets from S_i and B_j, we suppose the users do not perform any computation, i.e., they just look at the broadcast B_j and at S_i. Hence, the first members of equalities (6) and (7) can be rewritten as

$$H(\mathbf{K}_j|\mathbf{S}_1, \ldots, \mathbf{S}_{s-1}, \mathbf{S}_s, \mathbf{B}_{j_2}, \mathbf{B}_{j_1}) \text{ and } H(\mathbf{K}_j|\mathbf{S}_s, \mathbf{B}_{j_1}, \mathbf{B}_{j_2}).$$

[1] In Theorem 1, we have shown that *it is not* due to a design problem of those schemes.

But, since (7) implies that $H(\mathbf{K}_j|\mathbf{S}_s, \mathbf{B}_{j_1}, \mathbf{B}_{j_2}) = 0$ then, from (1) and (3), we get that

$$0 \leq H(\mathbf{K}_j|\mathbf{S}_1, \ldots, \mathbf{S}_{s-1}, \mathbf{S}_s, \mathbf{B}_{j_2}, \mathbf{B}_{j_1}) \leq H(\mathbf{K}_j|\mathbf{S}_s\mathbf{B}_{j_1}, \mathbf{B}_{j_2}) = 0.$$

Hence, conditions 3.*a*) and 3.*b*) hold simultaneously only if $H(\mathbf{K}_j) = 0$, for any $j = 1, \ldots, m$.

To make conditions 3.*a*) and 3.*b*) working, Definition 1 should specify that F and G correspond to subsets of revoked and joining users. Notice that the authors of [5] informally gave such a meaning to F and G in motivating the definition, but the requirement was not formally stated (and not used).

By using conditions 3.*a*) and 3.*b*) with no constraint on F and G, a lower bound on the size of the broadcast message the group manager sends at the beginning of each session, was therein derived. Such a bound holds only if $H(\mathbf{K}_1) = \ldots = H(\mathbf{K}_m) = 0$. Assuming the "corrected" version of conditions 3.*a*) and 3.*b*), i.e., where F and G correspond to subsets of revoked and joining users, the proof of the bound given in [5] does not work, and the bound does not hold.

4 Personal Key Distribution Schemes

In all proposed self-healing key distribution schemes [5,4,1], every user has a personal key S_i, which stays the same for all the lifetime of the scheme. At the beginning of the j-th session, every user who has not been revoked, computes his/her own new key Pk_i, by using S_i and the first part of the broadcast message B_j. Then, by means of Pk_i and the second part of the broadcast message B_j, he/she computes the group session key K_j.

In Appendix C of [5] and in [4], this behavior was formalised, as an intermediate step towards the definition of a self-healing key distribution scheme, and it was referred to as *Key Distribution* in [5] and as *Personal Key Distribution* in [4]. The definition of personal key distribution, using the terminology of [4], can be stated as follows:

Definition 2. *[4] Let* $t, i \in \{1, \ldots, n\}$. *In a* personal key distribution scheme $\mathcal{D}$, *the group manager seeks to establish a new key* Pk_i *with each group member* U_i *through a broadcast message B.* $\mathcal{D}$ *is a personal key distribution scheme if*

a) For any group member U_i, *the key* Pk_i *is determined by* S_i *and B, i.e.,*

$$H(\mathbf{Pk}_i|\mathbf{B}, \mathbf{S}_i) = 0.$$

b) For any set $F \subseteq \{U_1, \ldots, U_n\}$ *such that* $|F| \leq t$, *and any* $U_i \notin F$, *the members in F are not able to learn anything about* S_i, *i.e.,*

$$H(\mathbf{Pk}_i, \mathbf{S}_i|\{\mathbf{S}_{i'}\}_{U_{i'} \in F}, \mathbf{B}) = H(\mathbf{Pk}_i, \mathbf{S}_i).$$

c) No information on $Pk_1, \ldots, Pk_n$ *is learned from either the broadcast or the personal secret alone, i.e.,*

$$H(\mathbf{Pk}_1, \ldots, \mathbf{Pk}_n|\mathbf{B}) = H(\mathbf{Pk}_1, \ldots, \mathbf{Pk}_n|\mathbf{S}_1, \ldots, \mathbf{S}_n) = H(\mathbf{Pk}_1, \ldots, \mathbf{Pk}_n).$$

The concept of the distribution of a (different) personal key to every user could be of independent interest. But we can show, along the same lines of Theorem 1, that there is no scheme achieving all conditions of Definition 2.

Theorem 2. *If conditions a), b) and c) of Definition 2 hold then, for any* $i \in \{1, \ldots, n\}$,

$$H(\mathbf{Pk}_i) = 0.$$

Notice that in both [5,4] constructions for personal key distribution schemes were provided. In the full version of the paper [2] we prove Theorem 2, and point out where the proofs for such constructions fail.

5 A New Definition of Self-Healing Key Distribution

The setting we consider is the same given at the beginning of Section 2, but we slightly change some notation. We do not use, in our formalization, the intermediate random variable $Z_{i,j}$, used in Definition 1. Then, in order to simplify the presentation, for any subset of users $Y = \{U_{i_1}, \ldots, U_{i_g}\} \subseteq \mathcal{U}$, where $i_1 < i_2 < \ldots < i_g$, we will denote the random variables $\mathbf{X}_{i_1} \ldots \mathbf{X}_{i_g}$ by means of $\mathbf{X}_Y$. Finally, we state the following definition:

Definition 3. *Let* $\mathcal{U}$ *be the universe of users of the network and, for* $i = 0, \ldots, m$, *let* $G_i \subseteq \mathcal{U}$. *The triple* $(\mathcal{R}, \mathcal{J}, G_0)$, *where* $\mathcal{R}=(Rev_1, \ldots, Rev_m)$ *and* $\mathcal{J}=(Join_1, \ldots, Join_m)$, *is an* m*-long* t-revocation-joining strategy *applied by the group manager if:*

- $Rev_i \cap Join_j = \emptyset$, *for* $1 \leq i \leq j \leq m$.
- *For* $i = 1, \ldots, m$, $Rev_i \subseteq G_{i-1} \cup \ldots \cup G_0$ *and* $Rev_{i-1} \subseteq Rev_i$.
- $|Rev_m| \leq t$.

The above definition simply states that the group manager can revoke up to t users, and specifies that once a user is revoked from the group he/she is kept revoked in the subsequent sessions[2]. In the following, we denote by $\mathcal{H}_s$ the triple $(\mathcal{R}_s, \mathcal{J}_s, G_0)$, where $\mathcal{R}_s=(Rev_1, \ldots, Rev_s)$ and $\mathcal{J}_s=(Join_1, \ldots, Join_s)$, for any $1 \leq s \leq m$. It represents the revocation-joining strategy applied until session s. Moreover, we denote by $B_j^{\mathcal{H}_j}$ the broadcast message sent by GM in session j.

Definition 4. *Let* $\mathcal{U}$ *be the universe of users of a network, and let* m *and* t *be two integers.* $\mathcal{D}(m, t, \mathcal{U})$ *is a self-healing* m*-session key distribution scheme for* $\mathcal{U}$ *with* t*-revocation capability if, for any* m*-long* t*-revocation-joining strategy* $(\mathcal{R}, \mathcal{J}, G_0)$, *the following conditions are satisfied:*

[2] Notice that such an assumption does not yield loss of generality of the model. Indeed, a revoked user that needs to re-join the group can always be treated as a new one.

1. Key Computation. *Every* $U_i \in G_j$ *computes* K_j *from* $B_j^{\mathcal{H}_j}$ *and* S_i. Formally, it holds that:
$$H(\mathbf{K}_j|\mathbf{B}_j^{\mathcal{H}_j}, \mathbf{S}_i) = 0$$
2. Self-Healing. *Let* r *and* s *be integers such that* $1 \leq r < s \leq m$. *Each* $U_i \in G_r \cap G_s$, *from the broadcast messages* $B_r^{\mathcal{H}_r}$ *and* $B_s^{\mathcal{H}_s}$ *recovers all keys* $K_s, \ldots, K_r$. Formally, it holds that:
$$H(\mathbf{K}_r, \ldots, \mathbf{K}_s|\mathbf{S}_i, \mathbf{B}_r^{\mathcal{H}_r}, \mathbf{B}_s^{\mathcal{H}_s}) = 0.$$
3. Security of future keys. *Let* s *be an integer such that* $1 \leq s \leq m$. *Users in* G_s, *by pooling together their own personal keys and broadcast messages* $B_1^{\mathcal{H}_1}$, $\ldots$, $B_{s-1}^{\mathcal{H}_{s-1}}$, *do not get any information about keys* $K_s, \ldots, K_m$. Formally, it holds that:
$$H(\mathbf{K}_s, \ldots, \mathbf{K}_m|\mathbf{S}_{G_s}\mathbf{B}_1^{\mathcal{H}_1}, \ldots, \mathbf{B}_{s-1}^{\mathcal{H}_{s-1}}) = H(\mathbf{K}_s, \ldots, \mathbf{K}_m).$$
4. Security w.r.t collusion attacks. *Let* r *and* s *be integers such that* $1 \leq r \leq s \leq m$. *Given two subsets*[3] Rev_r *and* $Join_{s+1}$ *such that* $|Rev_r \cup Join_{s+1}| \leq t$, *users in* $Rev_r \cup Join_{s+1}$, *given the sequence of broadcast messages, do not get any information about keys* $K_r, \ldots, K_s$. Formally, it holds that:
$$H(\mathbf{K}_r, \ldots, \mathbf{K}_s|\mathbf{B}_1^{\mathcal{H}_1}, \ldots, \mathbf{B}_m^{\mathcal{H}_m}, \mathbf{S}_{Rev_r}, \mathbf{S}_{Join_{s+1}}) = H(\mathbf{K}_r, \ldots, \mathbf{K}_s).$$

The definition is divided in four parts: the first states that users in the group compute the session key and the second one states the self-healing property. The third and fourth parts state the security requirements. Roughly speaking, point *3.* means that future keys are secure: even if a group of users tries to get information about new session keys by using only their own personal keys and the transcript of previous communication, they do not get anything. On the other hand, point *4.* means that a coalition of revoked and new users does not get any information about keys such users are not entitled to compute.

6 Lower Bounds and Constructions

It is easy to check that the two bounds[4] reported in [1] can still be derived from Definition 4. More precisely, it holds that:

Theorem 3. *In any* $\mathcal{D}(m, t, \mathcal{U})$, *for any* U_i *belonging to the group since session* j, *where* $j \in \{1, \ldots, m\}$, *it holds that* $H(\mathbf{S}_i) \geq (m - j + 1)H(\mathbf{K})$.

[3] Notice that, if $s = m$, then we define $Join_{s+1} = \emptyset$ since the scheme can be used for m sessions.

[4] W.l.o.g, we assume that all session keys are chosen in a finite set K. Therefore, we denote by $H(\mathbf{K})$ the entropy of a random variable $\mathbf{K}$ assuming values over a finite set K.

Theorem 4. *In any* $\mathcal{D}(m,t,\mathcal{U})$, *for any* $j = 2,\ldots,m$, *and for any* $(\mathcal{R}, \mathcal{J}, G_0)$ m-*long* t-revocation-joining strategy, *it holds that* $H(\mathbf{B}_j^{\mathcal{H}_j}) \geq (j-1)H(\mathbf{K})$. *If* $j = 1$, *it holds that* $H(\mathbf{B}_1^{\mathcal{H}_1}) \geq H(\mathbf{K})$.

It is easy to get lower bounds also on the joint entropies of the personal keys and the broadcast messages.

Theorem 5. *In any* $\mathcal{D}(m,t,\mathcal{U})$, *for any subset of users* $\{U_{i_1},\ldots,U_{i_{t+1}}\}$ *belonging to the group in session* 1, *it holds that* $H(\mathbf{S}_{i_1},\ldots,\mathbf{S}_{i_{t+1}}) \geq (t+1)\cdot m \cdot H(\mathbf{K})$.

Theorem 6. *In any* $\mathcal{D}(m,t,\mathcal{U})$, *for any* $(\mathcal{R}, \mathcal{J}, G_0)$ m-*long* t-revocation-joining strategy, *it holds that* $H(\mathbf{B}_1^{\mathcal{H}_1},\ldots,\mathbf{B}_m^{\mathcal{H}_m}) \geq m \cdot H(\mathbf{K})$.

We show that Theorems 3 and 5 are tight, while Theorems 4 and 6 are almost tight, by describing a meta-construction for $\mathcal{D}(m,t,\mathcal{U})$ self-healing key distribution schemes. Such a meta-construction uses, as a building block, two $\mathcal{D}(1,t,\mathcal{U})$ constructions, which resemble the schemes given in [3].

Let $\mathcal{U} = \{U_1,\ldots,U_n\}$ be the universe of users, let $G_0 \subset \mathcal{U}$, and let F_q, where $q > n$, be a finite prime field.

Broadcast Almost-Optimal $\mathcal{D}(1,t,\mathcal{U})$.

1. Setup Phase
   - The group manager, for each possible subset $Rev \subset \mathcal{U}$ of size at most t, chooses, uniformly at random, a value $x^{Rev} \in F_q$. We assume that the subsets Rev are listed according to a lexicographic order.
   - The group manager gives to user $U_i \in G_0$ as personal key the sequence of pairs of values $S_i = < (x^{Rev}, Rev) >_{U_i \notin Rev, \forall Rev \subset \mathcal{U}}$.
2. Broadcast Phase
   - Let Rev_1 be the subset of revoked users. The group manager, at the beginning of the session, chooses uniformly at random a key K in F_q, computes $B_1^{\mathcal{H}_1} = (K - x^{Rev_1}, Rev_1)$, and broadcasts $B_1^{\mathcal{H}_1}$.

The above $\mathcal{D}(1,t,\mathcal{U})$ is used to show that Theorems 4 and 6 are almost tight. Similarly, the following $\mathcal{D}(1,t,\mathcal{U})$ scheme is used to show that Theorems 3 and 5 are tight.

User Memory Storage Optimal $\mathcal{D}(1,t,\mathcal{U})$.

1. Setup Phase
   - The group manager, chooses uniformly at random $t+1$ values, say $a_0,\ldots,a_t \in F_q$ and computes the polynomial $P(x) = \sum_{i=0}^{t} a_i x^i$ of degree t. For each user $U_i \in G_0$, the group manager computes the value $y_i = P(i) \bmod q$.
   - The group manager gives to user $U_i \in G_0$ as personal key the value $S_i = y_i$.

2. Broadcast Phase
   - Let Rev_1 be the subset of revoked users. The group manager chooses uniformly at random a key K in F_q, computes the sequence of pairs of values $B_1^{\mathcal{H}_1} = < (K - y_i, i) >_{U_i \in G_1}$, and broadcasts $B_1^{\mathcal{H}_1}$.

It is easy to check that, in both constructions, every user belonging to G_1 recovers the session key, while revoked users do not get any information about the key. In the first construction, it holds that $H(\mathbf{S}_i) = H(\mathbf{K})$. Hence, Theorem 3 is tight in the special case of 1-session schemes.

In order to set up a $\mathcal{D}(m,t,\mathcal{U})$ scheme, the group manager operates as follows:

A Meta-Construction for $\mathcal{D}(m,t,\mathcal{U})$ schemes.

1. Setup Phase
   - The group manager generates m *independent* copies $\Sigma_1, \ldots, \Sigma_m$ of the $\mathcal{D}(1,t,\mathcal{U})$ scheme described before.
   - The group manager gives to user $U_i \in G_0$ a personal key S_i comprising the sequence of m personal keys he/she would receive from $\Sigma_1, \ldots, \Sigma_m$.
2. Broadcast Phase
   - In session $j = 1, 2$, it broadcasts $B_j^{\mathcal{H}_j}$, according to scheme Σ_j.
   - In session $j \geq 3$, it broadcasts message $B_j^{\mathcal{H}_j}$, according to scheme Σ_j, *concatenated* with broadcast messages $B_{j-1}^{\mathcal{H}_{j-1}}, \ldots, B_2^{\mathcal{H}_2}$, associated to schemes $\Sigma_{j-1}, \ldots, \Sigma_2$, respectively.

A user who joins the group in session j gets, as personal key, the sequence of personal keys associated to him by schemes $\Sigma_j, \ldots, \Sigma_m$. Notice that, we are taking into account with every broadcast message $B_j^{\mathcal{H}_j}$, the largest possible self-healing interval at that time, i.e., $r = 1$ and $s = j$.

The above meta-construction, instantiated with the two different $\mathcal{D}(1,t,\mathcal{U})$ schemes previously described, proves that the bounds given by Theorems 3 and 5 are tight, while the ones given by Theorems 4 and 6 are almost tight. An interesting open problem is to find schemes achieving a good trade-off between user memory storage and communication complexity.

References

1. C. Blundo, P. D'Arco, A. De Santis, and M. Listo. *Design of Self-healing Key Distribution Schemes,* Design, Codes, and Cryptography, N.32, pp. 15–44, 2004.
2. C. Blundo, P. D'Arco, and A. De Santis, *On Self-healing Key Distribution,* available at http://www.dia.unisa.it/~paodar/publications.html
3. A. Fiat and M. Naor, *Broadcast Encryption,* Proceedings of Crypto '93, Lecture Notes in Computer Science, Vol. 773, pp. 480-491, 1994.
4. D. Liu, P. Ning, and K. Sun, *Efficient Self-Healing Key Distribution with Revocation Capability,* Proceedings of the 10-th ACM Conference on Computer and Communications Security, October 27-31, 2003, Washington, DC, USA.
5. J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean, *Self-Healing Key Distribution with Revocation,* IEEE Symposium on Security and Privacy, May 12-15, 2002, Berkeley, California.

Tree-Walking Automata Cannot Be Determinized

Mikołaj Bojańczyk* and Thomas Colcombet**

Uniwersytet Warszawski, Wydział MIM, Banacha 2, Warszawa, Poland

Abstract. Tree-walking automata are a natural sequential model for recognizing tree languages. It is shown that deterministic tree-walking automata are weaker than nondeterministic tree-walking automata.

Introduction

A tree-walking automaton (TWA) is a natural type of finite automaton working over trees. The automaton is a finite memory device which walks around a tree, choosing what move to make according to its current state and some information about its current position in the tree. After a certain amount of walking the automaton can choose to accept the tree. Even though TWA were introduced in the early seventies by Aho and Ullman [AU71], very little is known about this model.

This situation is different from the "usual" tree automata – branching tree automata – which are a very well understood object. Both top-down and bottom-up nondeterministic branching tree automata recognize the same class of languages. Languages of this class are called *regular,* the name being so chosen because it enjoys many nice properties of the class of regular word languages. The deterministic variants of branching tree automata are similarly well understood – deterministic bottom-up automata also recognize all regular tree languages, while deterministic top-down automata recognize a strict subclass of the class of regular languages.

It is a classical result that every language recognized by a TWA is regular. However most other fundamental questions pertaining to tree-walking automata remain unanswered:

1. Is every regular language recognized by a TWA?
2. Can TWA be determinized?
3. Is the class of languages recognized by TWA closed under complementation?

It is believed that the answers to all three questions above are negative. There has been much related research, which can be roughly grouped in two categories: non-definability results for weakened models of tree-walking automata [NS00,Boj03]

* Supported by Polish KBN grant No. 4 T11C 042 25.
** Supported by the European Community Research Training Network GAMES.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 246–256, 2004.

and definability results for strengthened models of tree-walking automata [KS81, EH99,EHvB99]. The three questions stated above, however, have remained open.

In this paper we answer question 2: we prove that there exists a language which is recognized by a tree-walking automaton, but by no deterministic one.

1 Tree Walking Automata, Patterns, and the General Idea

In this section we define tree-walking automata, specify our separating language and prove it is recognized by a nondeterministic tree-walking automaton.

Preliminaries

For two integers i and j, we denote by $[i,j]$ the set $\{n : i \leq n \leq j\}$. The trees we deal with in this paper are finite, binary trees labeled by a given finite alphabet Σ. Formally, a *Σ-tree* t is a mapping from $N_t \subseteq \{1,2\}^*$ to Σ, where N_t is a finite, non-empty, prefix-closed set such that for any $v \in N_t$, $v1 \in N_t$ iff $v2 \in N_t$. Elements of the set N_t are called *nodes* of the tree. We use the set $\text{Types} = \{r,1,2\} \times \{l,f\}$ to encode the possible *types* of a node: the first component has the value r for the root, 1 for a left son and 2 for a right one; the second component is l for a leaf or else f for fathers. For $v \in N_t$, let $type_t(v) \in$ Types denote the type of this node. A *direction* is an element in [0, 2], where informally 0 stands for 'up', 1 for 'down-left' and 2 for 'down-right'. Let

$$d : N_t \times N_t \to [0,2] \cup \{\bot\}$$

be the function assigning: i to pairs of the form $(v, v{\cdot}i)$, for $i \in \{1,2\}$; 0 to pairs of the form $(v{\cdot}i, v)$, for $i \in \{1,2\}$; and $\bot$ otherwise.

Definition 1 A *tree-walking automaton* over Σ-trees is a tuple $\mathcal{A} = (Q, q_I, F, \delta)$, where Q is a finite set of *states*, $q_I \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *accepting states* and δ is the *transition relation* of the form

$$\delta \subseteq Q \times \text{Types} \times \Sigma \times Q \times [0,2].$$

A *run* of $\mathcal{A}$ over a Σ-tree t is a sequence $(v_0, q_0) \ldots (v_n, q_n)$ satisfying

$$(q_i, type_t(v_i), t(v_i), q_{i+1}, d(v_i, v_{i+1})) \in \delta \quad \text{for all } i \in [0, n-1].$$

A run is *accepting*, if $v_0 = v_n = \varepsilon$, $q_0 = q_I$ and $q_n \in F$. The automaton $\mathcal{A}$ accepts a tree if it has an accepting run over it. A set of Σ-trees L is *recognized* by $\mathcal{A}$ if $\mathcal{A}$ accepts exactly the trees in L. Finally, we say that a tree-walking automaton is *deterministic* if δ is a function from $Q \times \text{Types} \times \Sigma$ to $Q \times [0,2]$. We use TWA to denote the class of tree languages which are recognized by some TWA and DTWA for languages recognized by some deterministic TWA.

We would like to point out here that reading the type of a node is an essential feature of tree-walking automata. Indeed, Kamimura and Slutzki showed in [KS81] that TWA which do not have access to this information cannot recognize all regular languages, being incapable of even searching a tree in a systematic manner by doing a depth-first search, for instance.

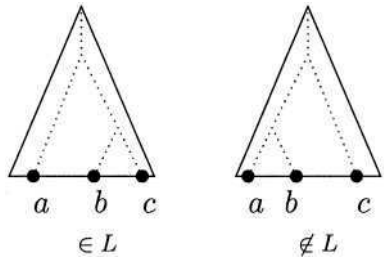


Fig. 1. The two kinds of well-formed trees

The Separating Language L

In this section we specify our separating language L, which we will eventually prove to witness the inequality $\mathbf{DTWA} \subsetneq \mathbf{TWA}$. Also in this section, we present a nondeterministic TWA which recognizes L. A proof that no DTWA can recognize L is more involved and will be spread across the subsequent sections.

The language L involves a very simple kind of trees, which we call *well-formed* trees: $\{B, a, b, c\}$-trees which have all nodes labeled by the *blank symbol* B but for three leaves: one leaf having label a; one leaf having label b and one leaf having label c. Let us simply call a (resp. b, c) the only node labeled by a (resp. by b, by c). Furthermore, in a well-formed tree we require the nodes a, b and c to be lexicographically ordered. One can check that the set of well-formed trees belongs to DTWA.

There are two possible kinds of well-formed trees: ones where the deepest common ancestor of a and b is above c; and ones where the deepest common ancestor of b and c is above a. The language L is the set of well-formed trees of the first kind. This definition is illustrated in Figure 1.

We now proceed to describe a nondeterministic tree-walking automaton which recognizes the language L.

Lemma 1. *There is a nondeterministic TWA which recognizes L.*

Proof. We will only give here an informal description of the automaton. This automaton first checks deterministically that the tree is well-formed, then reaches somehow the node labeled by c. From this node, it goes toward the root and at some point v decides (using nondeterminism) to perform a depth-first search from left to right. It then accepts the tree if in this search the first non-blank node encountered is a b, i.e. the left-most non-blank leaf below v is labeled by b.

One can verify that there exists an accepting run of this automaton if and only if the tree belongs to L. Indeed, when the tree belongs to L the automaton chooses v to be the deepest common ancestor of b and c. On the other hand, if a tree is well-formed but does not belong to L, there is no v ancestor of c such that the left-most non-blank leaf below v is labeled by b (this leaf is either a or c), and thus the automaton does not accept the tree.

2 Patterns

In this section, we introduce the key technical concept of patterns and outline how they can be used to show that L is outside DTWA. From now on we assume that a deterministic tree-walking automaton

$$\mathcal{A} = (Q, q_I, F, \delta)$$

is fixed; our aim is to prove that $\mathcal{A}$ does not recognize the language L.

Patterns and Pattern Equivalence

A *pattern* Δ is a $\{B, *\}$-tree where the symbol $*$ labels only leaves which are left sons. The i-th $*$-labeled leaf (numbered from left to right) is called the *i-th-port*. *Port* 0 stands for the root. The largest port number is called the *arity* of the pattern, and we use Pat^n to denote the set of n-ary patterns.

Given $\Delta \in \mathrm{Pat}^n$, and n patterns $\Delta_1, \ldots, \Delta_n$, the *composition* $\Delta[\Delta_1, \ldots, \Delta_n]$ is obtained from Δ by simultaneously substituting each Δ_i for the i-th port. We may use $*$ instead of some substituted patterns in a composition, the intended meaning being that the corresponding ports remain untouched. When all Δ_i's are $*$ but for D_k we simply write $\Delta[\Delta_k/k]$. If furthermore Δ is a unary pattern, we write $\Delta \cdot \Delta'$ instead of $\Delta[\Delta'/1]$. Given a set P of patterns, we denote by $\mathcal{C}(P)$ the least set of patterns which contains P and is closed under composition.

Definition 2 The *automaton's transition relation over an n-ary pattern* Δ,

$$\delta_\Delta \subseteq Q \times [0,n] \times Q \times [0,n] \ ,$$

contains a tuple (q, i, r, j) if it is possible for $\mathcal{A}$ to go from state q in port i to state r in port j in Δ. This run is assumed not to visit any port along the way but for the initial and final configurations, in which the ports are treated as having type $(1, f)$ (i.e. non-leaf left sons). In particular the port 0 is not seen as the root and non null ports are not seen as leaves (to make composition work).

From the point of view of the automaton $\mathcal{A}$, the relation δ_Δ sums up all important properties of a pattern and we consider two patterns equivalent if they induce the same relation. More precisely, for two patterns Δ and Δ' of the same arity n, we write

$$\Delta \simeq \Delta' \quad \text{iff} \quad \delta_\Delta = \delta_{\Delta'} \ .$$

The essence of this equivalence is that if one replaces a sub-pattern by an equivalent one, the automaton $\mathcal{A}$ is unable to see the difference. The following lemma shows that $\simeq$ acts as a congruence with respect to pattern composition:

Lemma 2. *For $\Delta_1 \simeq \Delta_1'$ of arity n, $\Delta_2 \simeq \Delta_2'$ and $i \in [1,n]$,*

$$\Delta_1[\Delta_2/i] \simeq \Delta_1'[\Delta_2'/i]$$

A consequence of the determinism of $\mathcal{A}$ is that for any pattern Δ of arity n, the relation δ_Δ is a partial function from $Q \times [0,n]$ to $Q \times [0,n]$ (it may be partial even if the original transition function is not since the automaton can be trapped in a loop). For this reason, we use from now and on a functional notation for δ relations.

Outline of the Proof

In order to prove that $\mathcal{A}$ cannot recognize L, we will produce three patterns: a nullary pattern Δ_0, a unary pattern Δ_1 and a binary pattern Δ_2. We then prove that compositions of these patterns satisfy several desirable properties. In particular, we ultimately show that for deterministic automata the following equivalence holds:

$$\Delta_2[*, \Delta_2] \simeq \Delta_2[\Delta_2, *] \ . \tag{1}$$

Having this equivalence, proving that $\mathcal{A}$ does not recognize L becomes a simple matter. Consider a context where a B-labeled tree is plugged for the port 0, and three trees with one a, b and c respectively are plugged into the ports 1, 2, 3. If we plug the left pattern from (1) into this context, we obtain a tree in L, and if we plug the right pattern, we obtain a tree outside L. However, since the patterns are equivalent, the automaton L cannot distinguish the two resulting trees and will either accept both or reject both, hence $\mathcal{A}$ does not recognize L.

Since the deterministic automaton $\mathcal{A}$ was chosen arbitrarily, it follows that $L \notin \mathrm{DTWA}$. Together with Lemma 1, we obtain this paper's contribution:

Theorem 1. *The class DTWA is strictly included in the class TWA.*

What remains to be done is to construct the patterns Δ_0, Δ_1 and Δ_2, which we do in Section 3; and then study properties of those patterns using the determinism of $\mathcal{A}$, which we do in Section 4. The culmination of this study is Corollary 4, from which the key equivalence (1) follows.

3 Basic Patterns

In this section, we define the patterns Δ_0, Δ_1 and Δ_2 and prove a basic property related to their composition (Lemma 4 and 5). Before we do this, we need to first a simple result concerning finite semigroups.

In order to define the patterns Δ_0, Δ_1, Δ_2 we need to state first a classical result concerning semigroups. Let us recall that a *semigroup* is a set together with an associative binary operation, which we write multiplicatively here.

Lemma 3. *For every finite semigroup S and any $u, v \in S$, there exist $u', v' \in S$ such that the elements $U = u \cdot u'$ and $V = v \cdot v'$ satisfy the following equations:*

$$U = U \cdot U = U \cdot V \qquad \text{and} \qquad V = V \cdot U = V \cdot V \ .$$

Let us now describe the construction of the patterns Δ_0, Δ_1, Δ_2 and prove Lemma 4. The insightful reader will notice that the determinism of $\mathcal{A}$ is not used in this part of the proof.

Let us denote by B_k the full binary tree of depth k. As the pattern equivalence relation $\simeq$ is of finite index, there exists m, n such that $m + 1 < n$ and $B_m \simeq B_n$. Let Δ_0 be B_m. In the tree B_n, the tree Δ_0 appears at least twice as a subtree rooted in a left son, thus there exists a binary pattern Δ_X such that $\Delta_X[\Delta_0, \Delta_0] = B_n$. Consider now the following two unary patterns:

$$\Delta_u = \Delta_X[*, \Delta_0] \quad \text{and} \quad \Delta_v = \Delta_X[\Delta_0, *] \ .$$

Let S be the semigroup whose elements are patterns in $\mathcal{C}(\{\Delta_u, \Delta_v\})$ and where the multiplication operation is the composition of unary patterns. Since S is a finite semigroup (modulo $\simeq$), there exist, by Lemma 3, unary patterns $\Delta_{u'}$ and $\Delta_{v'}$ in $\mathcal{C}(\{\Delta_u, \Delta_v\})$ such that the two patterns $\Delta_U = \Delta_u \cdot \Delta_{u'}$ and $\Delta_V = \Delta_v \cdot \Delta_{v'}$ satisfy the following equivalences:

$$\Delta_U \simeq \Delta_U \cdot \Delta_U \simeq \Delta_U \cdot \Delta_V \quad \text{and} \quad \Delta_V \simeq \Delta_V \cdot \Delta_U \simeq \Delta_V \cdot \Delta_V \ .$$

Let us define now Δ_1 to be Δ_U and Δ_2 to be $\Delta_1 \cdot \Delta_X[\Delta_{u'} \cdot \Delta_1, \Delta_{v'} \cdot \Delta_1]$. Finally, let $\mathcal{C}_\mathcal{A}$ stand for the set $\mathcal{C}(\{\Delta_0, \Delta_1, \Delta_2\})$ and let $\mathcal{C}_\mathcal{A}^n$ stand for $\mathcal{C}_\mathcal{A} \cap \mathrm{Pat}^n$. The following lemma shows that, from the point of view of the automaton $\mathcal{A}$, all patterns of a given small arity in $\mathcal{C}_\mathcal{A}$ look the same:

Lemma 4. *For all $k \in [0, 2]$ and all $\Delta \in \mathcal{C}_\mathcal{A}^k$, $\Delta \simeq \Delta_k$.*

Proof. Let us establish first the three following equivalences:

- $\Delta_1 \cdot \Delta_0 \simeq \Delta_0$. It is enough to prove by a simple induction that for all patterns Δ in $\mathcal{C}(\{\Delta_u, \Delta_v\})$, $\Delta \cdot \Delta_0 \simeq \Delta_0$.
- $\Delta_1 \cdot \Delta \simeq \Delta \simeq \Delta[\Delta_1/i]$ for all n-ary patterns $\Delta \in \mathcal{C}_\mathcal{A}$ and $i \in [1, n]$. This follows from the equivalence

$$\Delta_1 = \Delta_U \simeq \Delta_U \cdot \Delta_U = \Delta_1 \cdot \Delta_1$$

  and the definition of Δ_2, where the pattern Δ_1 appears next to every port.
- $\Delta_2[\Delta_0, *] \simeq \Delta_2[*, \Delta_0] \simeq \Delta_1$. By symmetry, we only prove one equivalence:

$$\begin{aligned} \Delta_2[\Delta_0, *] &= \Delta_1 \cdot \Delta_X[\Delta_{u'} \cdot \Delta_1 \cdot \Delta_0, \Delta_{v'} \cdot \Delta_1] \\ &\simeq \Delta_1 \cdot \Delta_X[\Delta_0, \Delta_{v'} \cdot \Delta_1] \simeq \Delta_U \cdot \Delta_V \cdot \Delta_U \simeq \Delta_U = \Delta_1 \ . \end{aligned}$$

Note now that every pattern in $\mathcal{C}_\mathcal{A}$ of arity in [0, 2] is either one of $\Delta_0, \Delta_1, \Delta_2$ or is a composition in which only patterns of arity no bigger than 2 are involved. The lemma is then established by using an induction driven by this decomposition, where each step corresponds to one of the equivalences above.

As an application of Lemma 4, we conclude this section by a description of runs that start and end in the same port of a pattern:

Lemma 5. *For $n \geq 1$, all patterns $\Delta \in \mathcal{C}^n_{\mathcal{A}}$, all states q, r and all $i \in [1,n]$:*

$$\delta_\Delta(q,0) = (r,0) \quad \text{iff} \quad \delta_{\Delta_1}(q,0) = (r,0)\ ,$$
$$\delta_\Delta(q,i) = (r,i) \quad \text{iff} \quad \delta_{\Delta_1}(q,1) = (r,1)\ .$$

Proof. The right to left implications follow from the fact that for all $\Delta \in \mathcal{C}^n_{\mathcal{A}}$ and all $i \in [1,n]$, the following equivalence holds (Lemma 4):

$$\Delta \simeq \Delta_1 \cdot \Delta \simeq \Delta[\Delta_1/i]\ .$$

The left to right implications follow from the fact that for all $\Delta \in \mathcal{C}^n_{\mathcal{A}}$ and all $i \in [1,n]$, the following equivalence holds (Lemma 4):

$$\Delta_1 \simeq \Delta[\overbrace{\Delta_0,\ldots,\Delta_0}^{i-1 \text{ times}},*,\overbrace{\Delta_0,\ldots,\Delta_0}^{n-i \text{ times}}]\ .$$

4 Swinging Removal

From now on, we will be using the fact that the automaton $\mathcal{A}$ is deterministic. We start by hiding the case when the automaton "swings", i.e. enters and exits a pattern of nonzero arity by the same port. Technically, we replace the δ functions with a higher-level construct γ, which can be considered as equivalent to δ, but furthermore has several desirable extra properties (Lemmas 7, 9 and 10).

Consider a unary pattern of the form $\Delta_1 \cdot \Delta_1$, with the nodes $v < w$ corresponding to the 1-ports of the two component Δ_1 patterns. For a state q, consider the unique maximal run of $\mathcal{A}$ which starts in (q,v) and visits neither the root nor w. If this run is finite, we call the state r in which node v is last visited the *ε-successor* $s_\varepsilon(q)$ of q, else $s_\varepsilon(q)$ is undefined.

We say q is an *upward* state if it can appear *after* $\mathcal{A}$ has traversed a pattern Δ_1 in the up direction, i. e. for some state r, $\delta_{\Delta_1}(r,1) = (q,0)$ holds. Similarly we define a *downward* state by swapping the role of ports 0 and 1. We use Q_U and Q_D to denote the sets of upward and downward states respectively.

We now introduce a new type of function which we will use instead of the δ functions. This replacement aims at eliminating the hassle involved with swinging. For $\Delta \in \mathcal{C}^n_{\mathcal{A}}$, the partial function

$$\gamma_\Delta : Q_D \times \{0\} \cup Q_U \times [1,n] \to Q_U \times \{0\} \cup Q_D \times [1,n]$$

is defined as $\gamma_\Delta(q,i) = \delta_\Delta(s_\varepsilon(q),i)$. From now on, we simplify slightly the notations by using γ_0, γ_1 and γ_2 for respectively γ_{Δ_0}, γ_{Δ_1} and γ_{Δ_2}.

We remark here, somewhat ahead of time, that the function γ_Δ is turns out to be completely defined for $\Delta \in \mathcal{C}_{\mathcal{A}}$. This because – thanks to the choice of the function's domain – we can be sure that the automaton does not loop. The intuitive reason for this is that an upward (or downward) state has already "survived" the traversal of a Δ_1 pattern, and it will not loop without good reason. The formal proofs are in Lemma 6 for patterns of nonzero arity and in Lemma 9 for the pattern Δ_0.

Lemma 6. *For any* $q \in Q_D$, $\gamma_1(q,0) = (q,1)$. *For any* $q \in Q_U$, $\gamma_1(q,1) = (q,0)$.

Proof. Let $q \in Q_D$. By definition, there is some r such that $\delta_{\Delta_1}(r,0) = (q,1)$. Consider the pattern $\Delta_1 \cdot \Delta_1$, with v labeling the interface between the two Δ_1 patterns and a run on this pattern which starts in $(r,0)$. The first time the node v is passed, state q is assumed, since $\delta_{\Delta_1}(r,0) = (q,1)$. The last time v is passed state $s_\varepsilon(q)$ is assumed, by definition of s_ε. Since $\Delta_1 \simeq \Delta_1 \cdot \Delta_1$, the run of the automaton starting with state r in port 0 of $\Delta_1 \cdot \Delta_1$ must end with state q in port 1, the last traversal of the lower Δ_1 pattern going downward from $s_\varepsilon(q)$ in v to q in port 1. Hence $\gamma_1(q,0) = \delta_{\Delta_1}(s_\varepsilon(q),0) = (q,1)$. The proof for $q \in Q_U$ is obtained by swapping the roles of ports 0 and 1.

The following lemma, which follows straight from the definition of the γ function, shows why γ allows us to ignore swinging:

Lemma 7. *For any* $\Delta \in \mathcal{C}_\mathcal{A}$ *of arity* $n \geq 1$, *states* q, r *and any port* $i \in [0,n]$, *if* $\gamma_\Delta(q,i) = (r,j)$ *then* $i \neq j$.

Proof. We only do the case where q is a downward state and hence $i = 0$. Let $q' = s_\varepsilon(q)$. Assume that the lemma is not true, hence $j = 0$. This means that $\delta_\Delta(q',0) = (r,0)$. By Lemma 5, $\delta_{\Delta_1}(q',0) = (r,0)$ and hence $\gamma_1(q,0) = (r,0)$, a contradiction with Lemma 6.

We start with a simple description of the behaviour of downward states:

Lemma 8. *For* $q \in Q_D$, *either* $\gamma_2(q,0) = (q,1)$ *or* $\gamma_2(q,0) = (q,2)$ *holds.*

Proof. Let $\gamma_2(q,0) = (r,i)$. By Lemma 7 , $i \in \{1,2\}$. Let us assume, without lessening of generality, that $i = 1$. Since $\Delta_1 \simeq \Delta_2[*,\Delta_0]$, we obtain that $\gamma_1(q,0) = (r,1)$. Hence it must be that $r = q$, since $\gamma_1(q,0) = (q,1)$ holds by Lemma 6.

Lemma 9. *For all* $\Delta \in \mathcal{C}_\mathcal{A}$, *the partial function* γ_Δ *is completely defined.*

Proof. Consider a pattern Δ of nonzero arity n and assume that $\gamma_\Delta(q,i)$ is undefined for some $i \in [0,n]$. Let us consider first the case when $i \neq 0$. If we plug all the ports of Δ but i with a Δ_0 pattern, we get a pattern equivalent to Δ_1. But this would imply that $\gamma_1(q,i)$ is undefined, a contradiction with Lemma 6. The case of $i = 0$ is proved analogously.

For the case of $\Delta = \Delta_0$, let us assume for a moment that when entering from $(q,0)$, the automaton gets lost in the pattern. But then, since $\Delta_1 \simeq \Delta_2[*,\Delta_0] \simeq \Delta_2[\Delta_0,*]$ and by Lemma 8, $\mathcal{A}$ would also get lost in Δ_1 when entering from $(q,0)$, a contradiction with Lemma 6.

The following lemma shows that to establish the equivalence of two patterns, it is enough to study the γ functions.

Lemma 10. *For* $n \geq 0$ *and all* $\Delta, \Delta' \in \mathcal{C}_\mathcal{A}^n$, $\Delta \simeq \Delta'$ *if and only if* $\gamma_\Delta = \gamma_{\Delta'}$.

Proof. The left to right implication is straightforward and, in fact, true for any pattern. The right to left implication is more involved. It is known for arities up to two from Lemma 4. Let us consider two patterns Δ and Δ' of arity at least one such that $\gamma_\Delta = \gamma_{\Delta'}$, and a state q. Three cases have to be studied.

- If $\delta_{\Delta_1}(q,0)$ is undefined then so is $\delta_\Delta(q,0)$, since $\Delta_1 \cdot \Delta \simeq \Delta$. The same goes for $\delta_{\Delta'}(q,0)$.
- If $\delta_{\Delta_1}(q,0) = (r,0)$ for some r, by Lemma 5, $\delta_\Delta(q,0) = \delta_{\Delta'}(q,0) = (r,0)$.
- Otherwise $\delta_{\Delta_1}(q,0) = (r,1)$ for some r. As Δ_1 is equivalent to Δ in which all nonzero ports but one are replaced by Δ_0, we obtain that $\delta_\Delta(q,0)$ is defined. Let (r',i) be this value. According to Lemma 5, $i \neq 0$. Let us consider now the run of the automaton in pattern $\Delta_1 \cdot \Delta \simeq \Delta$. It crosses first the junction point with state $\delta_{\Delta_1}(q,0) = (r,1)$, then after some walk, reaches the same node with state $s_\varepsilon(r)$. Finally it crosses the Δ pattern and reaches port i in state r', which means $\delta_\Delta(r,0) = (r',i)$. We obtain that

$$\delta_\Delta(q,0) = \delta_\Delta(s_\varepsilon(r),0) = \gamma_\Delta(r,0) \ .$$

  Similarly $\delta_{\Delta'}(q,0) = \gamma_{\Delta'}(r,0)$, and hence $\delta_\Delta(q,0) = \delta_{\Delta'}(q,0)$.

The same method can be applied for ports in $[1,n]$.

From now on, the γ function is used as if it was the original δ function, in particular with respect to composition.

5 Generic Behaviours

In this last part we show that, essentially, $\mathcal{A}$ can only do depth-first searches over patterns in $\mathcal{C}_\mathcal{A}$. Using this characterization, we prove the main technical lemma of this paper, Lemma 12. This characterization is obtained by analyzing the γ functions. Due to the domain of γ we need to consider two cases: downward states in port 0 and upward states in the other ports.

A good understanding of the behaviour of downward states in patterns from $\mathcal{C}_\mathcal{A}$ comes from Lemma 6: if a downward state q starts in port 0 of a pattern $\Delta \in \mathcal{C}_\mathcal{A}$, it will emerge in state q either in the leftmost or the rightmost nonzero port.

The description of the behaviour of upward states is more involved. When starting in a nonzero port, an upward state may go in the direction of the root, but it may also try to visit a neighboring nonzero port (something that does not appear in the Δ_1 pattern used to define upward states). The following definition, along with Lemma 11, gives a classification of the possible behaviours of upward states:

Definition 3 We say that a pair of states $(q,r) \in Q_D \times Q_U$ has *right to left depth-first search behaviour* if

$$\gamma_2(q,0) = (q,2), \quad \gamma_0(q,0) = (r,0), \quad \gamma_2(r,2) = (q,1), \quad \text{and } \gamma_2(r,1) = (r,0).$$

A *left to right depth-first search behaviour* is defined symmetrically by swapping the roles of ports 1 and 2. An upward state r has *ascending behaviour* if

$$\gamma_2(r,1) = \gamma_2(r,2) = (r,0) .$$

The following lemma shows that Definition 3 is exhaustive:

Lemma 11. *For any upward state r, either r has ascending behaviour, or there exists a downward state q such that the pair (q,r) has either right to left or left to right depth-first search behaviour.*

Proof. Let r be an upward state. We first show that either $\gamma_2(r,1) = (r,0)$ or $\gamma_2(r,2) = (r,0)$. Let us suppose that $\gamma_2(r,1) \neq (r,0)$ must hold. By Lemma 7 we have $\gamma_2(r,1) = (q,2)$ for some downward state q. Let r' be the downward state such that $\gamma_0(q,0) = (r',0)$. Since $\Delta_1 \simeq \Delta_2[*,\Delta_0]$ and $\gamma_1(r,1) = (r,0)$ (Lemma 6), we obtain that $\gamma_2(r',2) = (r,0)$, and hence $\gamma_1(r',1) = (r,0)$. Since $\gamma_1(r',1) = (r',0)$, we obtain $r = r'$ and $\gamma_2(r,2) = (r,0)$. Thus, either $\gamma_2(r,1) = (r,0)$ or $\gamma_2(r,2) = (r,0)$.

If both cases hold, then r has ascending behaviour. Otherwise exactly one case holds, without lessening of generality let us assume it is:

$$\gamma_2(r,2) = (r,0), \quad \text{and} \quad \gamma_2(r,1) \neq (r,0). \tag{2}$$

As in the reasoning above, let q be the state such that

$$\gamma_2(r,1) = (q,2) . \tag{3}$$

We claim that (q,r) has left to right depth-first search behaviour. As we have seen before,

$$\gamma_0(q,0) = (r,0) . \tag{4}$$

Since we already have the equations (2), (3) and (4), to establish that the pair (q,r) has left to right depth-first search behaviour we only need to prove the equation (5).

Since q is a downward state, then by Lemma 8 the value of $\gamma_2(q,0)$ must be either $(q,1)$ or $(q,2)$. But the second case cannot hold, since together with equations (2) and (4) this would give us $\gamma_{\Delta_2[*,\Delta_0]}(q,0) = (r,0)$, a contradiction with Lemma 7. This means that

$$\gamma_2(q,0) = (q,1) . \tag{5}$$

and, hence, (q,r) has left to right depth-first search behaviour. The right to left depth-first search behaviour case is obtained by a symmetric argument when the roles of ports 1 and 2 are exchanged in the assumptions (2).

Now that we know exactly how the automaton behaves for upward and downward states, we obtain the following as a simple consequence of Lemmas 8 and 11, none of whose described behaviours make it possible to distinguish patterns of the same arity:

Lemma 12. *For all n and all $\Delta, \Delta' \in \mathcal{C}^n_{\mathcal{A}}$, the functions γ_Δ and $\gamma_{\Delta'}$ are equal.*

Proof. The statement of the lemma follows from the following characterization of moves over an arbitrary pattern $\Delta \in \mathcal{C}_{\mathcal{A}}$ of arity $n \geq 1$:

- For any downward state q, by Lemma 8, two cases may happen:
  - If $\gamma_1(q,0) = (q,1)$ then $\gamma_\Delta(q,0) = (q,1)$.
  - If $\gamma_1(q,0) = (q,2)$ then $\gamma_\Delta(q,0) = (q,n)$.
- If q is an upward state then, by Lemma 11, three cases may happen:
  - If the state q has ascending behaviour, $\gamma_\Delta(q,i) = (q,0)$ for all $i \in [1,n]$.
  - If for some downward state r, the pair (q,r) has right to left depth-first-search behaviour, then $\gamma_\Delta(q,i) = (r,i+1)$ for all $i \in [1,n-1]$ and $\gamma_\Delta(q,n) = (q,0)$.
  - If for some downward state r, the pair (q,r) has right to left depth-first-search behaviour, then $\gamma_\Delta(q,0) = (q,0)$ and $\gamma_\Delta(q,i) = (r,i-1)$ for all $i \in [2,n]$.

The above lemma, together with Lemma 10, gives us the required:

Corollary 4 The equivalence $\Delta_2[\Delta_2, *] \simeq \Delta_2[*, \Delta_2]$ holds.

Acknowledgment. We would like to thank C. Löding and A. Meyer for reading previous versions of this paper.

References

[AU71] A. V. Aho and J. D. Ullman. Translations on a context-free grammar. *Information and Control,* 19(5):439–475, dec 1971.

[Boj03] M. Bojańczyk. 1-bounded TWA cannot be determinized. In *FSTTCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings,* volume 2914 of *Lecture Notes in Computer Science,* pages 62,73. Springer, 2003.

[EH99] J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In G. Paum J. Karhumaki, H. Maurer and G. Rozenberg, editors, *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa,* pages 72–83. Springer-Verlag, 1999.

[EHvB99] J. Engelfriet, H. J. Hoogeboom, and J.-P. van Best. Trips on trees. *Acta Cybernetica,* 14:51–64, 1999.

[KS81] T. Kamimura and G. Slutzki. Parallel two-way automata on directed ordered acyclic graphs. *Information and Control,* 49(1):10–51, 1981.

[NS00] F. Neven and T. Schwentick. On the power of tree-walking automata. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000,* volume 1853 of *LNCS,* 2000.

Projecting Games on Hypercoherences

Pierre Boudes

Institut de mathématiques de Luminy UMR 6206,
campus de Luminy case 907,
13288 Marseille cedex 9, France,
boudes@iml.univ-mrs.fr,
http://boudes.lautre.net

Abstract. We compare two interpretations of programming languages: game semantics (a dynamic semantics dealing with computational traces) and hypercoherences (a static semantics dealing with results of computation). We consider polarized bordered games which are Laurent's polarized games endowed with a notion of terminated computation (the border) allowing for a projection on hypercoherences. The main result is that the projection commutes to the interpretation of linear terms (exponential-free proofs of polarized linear logic). We discuss the extension to general terms.

The Curry-Howard isomorphism establishes a correspondence between proofs and programs and between formulæ and types. In this paper we adopt the logical point of view on computation and we use the sequent calculus syntax (where cut-elimination represents dynamic of computation) of Girard's linear logic [10] (LL for short). Let us recall that LL splits logic into a linear fragment, where resources are consumed when used, and an exponential fragment, allowing data copying and erasing through structural rules.

1 Introduction

In denotational semantics, an agent (a program, a term or a proof) is represented as a structure describing all its possible interactions with other agents. Static semantics (e.g. hypercoherences [7]) focus on results of interactions while dynamic semantics (e.g. game semantics) focus on interaction histories (computational traces called *plays* in game semantics). This difference is somewhat the same as the difference between a function (static) and an algorithm (dynamic).

In the fifties, Kreisel introduced partial equivalence relations (PER) to deal with higher order functions presented in an operational manner (algorithms, recursive functions, proofs). Partiality of the equivalence relation comes from the fact that a higher order algorithm can separate two algorithms which compute the same function whereas a higher order function cannot.

The hypercoherence semantics of LL [7] has been introduced to give an "extensional" account of sequentiality [6]. Hypercoherences, just as coherence spaces, are built by adding a graph structure to the objets of the *relational*

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 257–268, 2004.

model. In this model, formulæ are interpreted as sets of points (results of computation) and agents as relations between them.

In [8], Ehrhard shows that hypercoherences form the *extensional collapse* (the quotient by PERs) of sequential algorithms (a model of PCF introduced by Berry and Curien in [3] and which has been shown by Lamarche to be a game model [13]). This result has been proved again by Longley and Van Oosten, with different methods (see [16,20]) and has been independently extended to other game semantics by Laird [12] and Melliès [18]. This relates surprisingly games to hypercoherences in the simple types hierarchy and shows that hypercoherences carry an implicit representation of the dynamic of computation. Our goal is to make the dynamical content of hypercoherences more explicit, in LL and not only in the simple types hierarchy. The various proofs of Ehrhard's result we already mentioned do not give a clear understanding of this dynamical content.

In [17], P.-A. Melliès gives a new proof of Ehrhard's result which clarifies the relation between games and hypercoherences, for simple types.

In [2], Baillot, Danos, Ehrhard and Regnier present the projection of a standard game model of multiplicative exponential linear logic onto a suitable static model based on the relational semantics by means of a lax time forgetful functor. Since this functor is lax, the projection of the game interpretation of a proof is included into its static interpretation, but not the converse, in general.

Our approach to the comparaison of games and hypercoherences consists in finding a suitable framework with a projection of plays onto points of the relational model, and then working out, on top of this framework, a precise relation between the hypergraph structure of hypercoherences and the dynamical structure of games.

In section 2, inspired by the *rigid parallel unfolding* of hypercoherences of [9], we introduce *polarized bordered games* (PBG for short). PBGs are polarized games (a game model of both polarized linear logic and linear logic with polarities [15,14]) endowed with a *border* which is a set of plays to be considered as the *terminated* plays.

We present the PBG interpretation of the linear fragment (MALLpol for short) of linear logic with polarities (LLpol for short).

The terminated plays of a PBG are the plays which can be projected onto the points of the relational model. Thanks to this additional structure, the projection commutes to the interpretation of proofs of MALLpol. But, in general, the projection of a strategy (other than the interpretation of a proof) is not a clique in hypercoherences.

A peculiar reversibility property of PBGs is also presented.

We next show, in section 3, how to extend the PBG semantics to LLpol and ILL, considering two interpretations of the exponentials. One is a version "with a border" of the exponential of Berry-Curien's sequential algorithms and the other is a new kind of exponential.

In section 4, we try to relate the hypergraph structure of hypercoherences with PBGs. We briefly present an unfolding of hypercoherences into *tower trees,* generalizing a construction given in [9] with the aim of disclosing the dynamical

content of hypercoherences. This unfolding maps the hypercoherence interpretation of additive and multipicative connectives to their PBG interpretation. This is not the case, in general, for exponentials.

We end this section by recalling the notion of hypercoherence and the syntax and the rules of linear logic with polarities.

A hypercoherence X is just a hypergraph consisting of a countable set of vertices $|X|$, the *web,* together with a set of hyperedges Γ, the *coherence.* More precisely, Γ is a subset of $\mathcal{P}^*_{\text{fin}}(|X|)$, the set of non-empty finite subsets of the web. In hypercoherences, each singleton is coherent. The *strict coherence* Γ^* is just coherence without singletons, *i.e.* $\Gamma \setminus \{\{a\} \mid a \in |X|\}$. The *incoherence* $\Gamma^\perp$ is the complementary set of Γ^* in $\mathcal{P}^*_{\text{fin}}(|X|)$. A clique is a subset x of the web such that $\mathcal{P}^*_{\text{fin}}(x) \subseteq \Gamma$.

The *orthogonal* is interpreted by the exchange of coherence and incoherence.

The interpretation of LL in hypercoherences follows the pattern of its interpretation in coherence spaces (see [7]).

Linear logic with polarities, LLpol, is the fragment of LL restricted to formulæ:

$$P := 0 \mid 1 \mid P \oplus P \mid P \otimes P \mid !N \mid \alpha^\perp$$
$$N := \top \mid \bot \mid N \,\&\, N \mid N \parr N \mid ?P \mid \alpha$$

where α denotes atoms, P stands for positive formulæ and N for negative formulæ. In LLpol sequents contains at most one positive formula. We use Γ to range over contexts containing at most one positive formula and $\mathcal{N}, \mathcal{N}'$ to range over contexts made of negative formulæ only. The rules are just the ordinary rules of LL:

$$\frac{}{\vdash \alpha, \alpha^\perp}\,(ax.) \qquad \frac{}{\vdash 1}\,(one) \qquad \frac{\vdash \Gamma}{\vdash \bot, \Gamma}\,(bot) \qquad \frac{}{\vdash \top, \Gamma}\,(top)$$

$$\frac{\vdash \mathcal{N}, P_i \quad (i = 1,2)}{\vdash \mathcal{N}, P_1 \oplus P_2}\,(plus) \qquad \frac{\vdash \Gamma, N \qquad \vdash \Gamma, N'}{\vdash \Gamma, N \,\&\, N'}\,(with)$$

$$\frac{\vdash N, N', \Gamma}{\vdash N \parr N', \Gamma}\,(par) \qquad \frac{\vdash \mathcal{N}, P \qquad \vdash \mathcal{N}', P'}{\vdash \mathcal{N}, \mathcal{N}', P \otimes P'}\,(tens.)$$

$$\frac{\vdash ?P_1, \ldots, ?P_n, N}{\vdash ?P_1, \ldots, ?P_n, !N}\,(prom.) \qquad \frac{\vdash \mathcal{N}, P}{\vdash \mathcal{N}, ?P}\,(der.) \qquad \frac{\vdash \mathcal{N}, N^\perp \qquad \vdash N, \Gamma}{\vdash \mathcal{N}, \Gamma}\,(cut)$$

$$\frac{\vdash \Gamma}{\vdash ?P, \Gamma}\,(weak.) \qquad \frac{\vdash ?P, ?P, \Gamma}{\vdash ?P, \Gamma}\,(cont.)$$

The linear subsystem of LLpol, denoted by MALLpol, is LLpol without the structural rules (weakening and contraction). In MALLpol we denote ! by $\downarrow$ and ? by $\uparrow$ and these modalities are called shifts, since they are not real exponentials but shifts of polarities.

We use the notation [ ] for multisets while the notation { } is, as usual, for sets.

2 Polarized Bordered Games

If A is an alphabet (a set) then A^* denotes the set of words on A (the finite sequences, up to reindexing). We denote by ε the empty word and by $w \cdot m$ the concatenation of the words w and m. The longest common prefix of two words w and w' is denoted by $w \wedge w'$.

If A and B are disjoint sets and if $C \subseteq A \cup B \cup (A \times B) \cup (B \times A)$ then *restriction to A* is the function from C^* to A^* defined by $\varepsilon \restriction A = \varepsilon$, $w \cdot l \restriction A = (w \restriction A) \cdot a$ if $l = a \in A$ or if $l = (a, b) \in A \times B$ or if $l = (b, a) \in A \times B$ and $w \cdot l \restriction A = w \restriction A$ if $l \in B$.

A set of words E is seen as a *forest:* vertices are the non empty prefixes of the elements of E, roots are the one letter words and an edge relates two words when one of them is the immediate prefix of the other. Conversely, a forest can always be described as a set of words. This set is usually taken prefix-closed for ensuring some unicity of the representation, but this restriction is not convenient in the present setting.

Here, a *tree isomorphism* between $E \subseteq A^*$ and $F \subseteq B^*$ is a bijection $f : E \to F$ such that for each pair $s, s' \in E$, the length of $f(s) \wedge f(s')$ is equal to the length of $s \wedge s'$. The usual notion of tree of forest isomophism would normally corresponds to a standard representation of trees as prefix-closed sets of words. Here, E and F may have isomorphic prefix closures without being isomorphic.

In what follows, logical polarities and game polarities can be identified. So, negative is opponent and positive is player.

Definition 1 (polarized bordered game). *A PBG A is a tuple* (ϵ, A^-, A^+, S) *where* $\epsilon \in \{-, +\}$ *is the polarity of* A *(*$\bar{\epsilon}$ *denotes the opposite polarity),* A^- *and* A^+ *are two countable and disjoint sets, and where S, the* border *of A, is a subset of* $(A^\epsilon \cdot A^{\bar{\epsilon}})^*$. *The elements of the prefix closure of S, denoted as* P_A, *are the* plays *of A, the elements of S are the* terminated plays *of A, and* $P_A^{\bar{\epsilon}}$ *(resp.* P_A^ϵ*) denotes the even prefix (resp. odd prefix) closure of S.*

If S is a set of words then a subset x of S is *even-deterministic* (resp. *odd-deterministic*) if for each two elements s and s' of x which are incomparable for the prefix order, the length of $s \wedge s'$ is even (resp. odd).

Definition 2 (strategies). *In a PBG A, a* strategy *is a deterministic subset of* S_A*: even-deterministic if* $\epsilon_A = -$*, or odd-deterministic if* $\epsilon_A = +$.

Let $E \subseteq C^*$ and $E' \subseteq C'^*$ be two sets of words on two disjoint alphabets. Let $m \in E$ and $m' \in E'$. The set $m \bullet_{C,C'} m'$ is the subset of words w on the alphabet $C \cup C'$ such that $w \restriction C = m$ and $w \restriction C' = m'$. And the set $E \bullet_{C,C'} E'$ is equal to

$$E \bullet_{C,C'} E' = \bigcup_{\substack{m \in E \\ m' \in E}} m \bullet_{C,C'} m'. \tag{1}$$

If $C = A \cdot (B \cdot B)^* \cdot A$ and $C' = A' \cdot (B' \cdot B')^* \cdot A'$, where A, A', B, B' are pairwise disjoint, then we also denote by $\odot_{A,A'}$ the operation $\bullet_{C,C'}$.

On words of even length, the operation $\otimes_{A,A'}$ is defined as follows:

$$d_1 \cdot m \cdot d_2 \otimes_{A,A'} d'_1 \cdot m' \cdot d'_2 = \{(d_1, d'_1) \cdot w \cdot (d_2, d'_2) \mid w \in m \odot_{A,A'} m'\} \quad (2)$$

where $d_1, d_2 \in A \cup B$ and $d'_1, d'_2 \in A' \cup B'$, and if one of the two words t or t' is empty then $t \otimes t' = \emptyset$.

We define the logical connectives on PBGs respecting the LLpol polarity restrictions.

The *orthogonal* of $A = (\epsilon_A, A^-, A^+, S_A)$ is the PBG $A^\perp = (\overline{\epsilon_A}, A^-, A^+, S_A)$. *Top* is the PBG $\top = (-, \emptyset, \emptyset, \emptyset)$. *Bot* is the PBG $\bot = (-, \{*\}, \{*'\}, \{**'\})$. Let $A = (-, A^-, A^+, S_A)$ and $B = (-, B^-, B^+, S_B)$ be two negative disjoint PBGs (when A and B are not disjoint we separate them by using subscripts). The *positive shift* of A is the PBG $\downarrow A = (+, A^- \cup \{*'\}, A^+ \cup \{*\}, \{*\} \cdot S_A \cdot \{*'\})$ (again we use subscripts when needed for avoiding confusion between the moves of A and $\{*, *'\}$). The PBG A *with* B is $A \,\&\, B = (-, A^- \cup B^-, A^+ \cup B^+, S_A \cup S_B)$. The PBG A *par* B is $A \parr B = (-, A^- \cup B^- \cup (A^- \times B^-), A^+ \cup B^+ \cup (A^+ \times B^+), S_A \otimes_{A,B} S_B)$. The interpretation of positives is defined using duality. Linear implication is defined as usual by setting $A \multimap B = A^\perp \parr B$ (according to logical polarities, A must be positive and B negative).

We also introduce the following non logical constructions. The *negative tensor* of A and B, $A \odot B$ is the PBG $(-, A^- \cup B^-, A^+ \cup B^+, S_A \odot S_B)$. The *negative linear map* from A *to* B, $A \rightarrowtriangle B$ is the PBG $(-, A^+ \cup B^-, A^- \cup B^+, S)$ where $S = \{w \in ((A^+ \cup B^-) \cdot (A^- \cup B^+))^* , w \restriction A \in S_A \text{ and } w \restriction B \in S_B\}$. This S is tree-isomorphic to the border of $\downarrow A \multimap B = \uparrow(A^\perp) \parr B$. The *negative tensor unit*, $\int$, is the PBG $(-, \emptyset, \emptyset, \{\varepsilon\})$.

There are many (tree) isomorphisms between the borders associated to these constructions. (Polarized tree isomorphisms are in fact isomorphisms of games in the category to be defined later). Some of them express standard associativity, commutativity, neutrality and distributivity properties.

The other important isomorphisms are[1]:

$$\downarrow \top = 0 \quad (3)$$

$$\downarrow(A \odot B) \cong \downarrow A \otimes \downarrow B \quad (\text{and } \downarrow \int = 1) \quad (4)$$

$$A \rightarrowtriangle \bot = \uparrow(A^\perp) \quad (5)$$

$$\int \rightarrowtriangle A = A \quad (6)$$

$$A \parr \int \cong A \parr \top \cong \top \quad (7)$$

$$\text{If } \varepsilon \notin S_A \text{ and } \varepsilon \notin S_B \text{ then } A \rightarrowtriangle B \cong (\downarrow A)^\perp \parr B = (\downarrow A) \multimap B. \quad (8)$$

Definition 3 (linear/affine, full, terminated). *A PBG A is* linear *(resp.* affine*) if $\varepsilon \notin S_A$ (resp. $\varepsilon \in S_A$). A PBG is* full *if for each play p in P_A there exist $s, s' \in S$ such that $s \wedge s' = p$. A PBG is* terminated *if no terminated play is the prefix of another terminated play.*

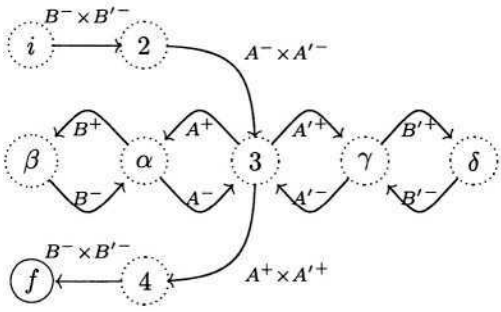
[1] In [15], $\int = \top$ and the same isomorphisms hold.

The interpretation of MALLpol proofs is inductively defined by cases on the last rule, following the pattern given in [15]. As in polarized games we use central strategies to interpret sequents containing one positive formula[2].

Definition 4 (central strategy). *A strategy x in $A \multimap B$ is* central *if, for each element s of x, the first positive move of s is in A^- and the last negative move of s is in A^+.*

A category NG of negative PBGs is defined. The morphisms from A to B are just the strategies of $A \multimap B$. Identity morphisms are, as usual, copycat strategies. In NG, a copycat strategy is central. Every isomorphism is a copycat and defines a unique tree isomorphism between the borders. Conversely, a tree isomorphism between borders of PBGs defines a unique isomorphism, in NG.

The composition $x \fatsemi y$ of two strategies, $x \subseteq S_{A \multimap B}$ and $y \subseteq S_{B \multimap C}$, is defined *pointwise:* $x \fatsemi y = \{s \fatsemi s' \mid s \in x, s' \in y, s \restriction B = s' \restriction B\}$, where $s \fatsemi s'$ is the projection on $A \cup C$ of the *unique* word t in $(A \cup B \cup C)^*$ such that $t \restriction A \cup B = s$ and $t \restriction B \cup C = s'$. The word t is called a *witness* of the composition of x and y. If $s'' \in x \fatsemi y$ then there is a unique witness t such that $t \restriction A \cup C = s''$. Remark that one has a similar property in coherence spaces or in hypercoherences.



An important point about composition is that it cannot be defined by, first, taking the usual game composition of the even prefix closure of x and y, and then, restrict the result to the border of $A \multimap C$. For non terminated PBGs, this would in general lead to a non associative operation.

We also define a sub-category CNG of NG where objects are linear negative terminated PBGs and morphisms are central strategies.

The *par* of two central strategies, $x \subseteq S_{A \multimap B}$ and $y \subseteq S_{A' \multimap B'}$, is a central strategy $x \parr y$, equal to the set of words w accepted by the automaton above and such that $w \restriction (A \multimap B) \in x$ and $w \restriction (A' \multimap B') \in y$.

Proposition 5. *The operation $\parr$ is a bifunctor of CNG and this category is symmetric monoidal for the structure $(\parr, \bot)$. The categories NG and CNG are also Cartesian for the with and have $\top$ as terminal object.*

We interpret proofs of $\vdash N_1, \ldots, N_n$ as strategies of $N_1 \parr \ldots \parr N_n$ and proofs of $\vdash P, N_1, \ldots, N_n$ as central strategies of $P^\bot \multimap (N_1 \parr \ldots \parr N_n)$.

The interpretation of additive and *par* rules are as usual. Axioms are interpreted as identity morphisms and cuts are interpreted as composition. If x and y are the central strategies interpreting, respectively, a proof π of $\vdash P, N$ and a proof π' of $\vdash P', N'$, then the application of a tensor rule between π and π' is interpreted as $x \parr y$. The negative shift rule (*i.e.* dereliction)

[2] This terminology has been introduced by Laurent in [15] and refers to Selinger's control categories [19].

is interpreted as a composition with the isomorphism $P^{\perp} \multimap N \cong \uparrow P \parr N$. If x is the interpretation of a proof π of $\vdash \uparrow P_1, \ldots, \uparrow P_k, N$ then π followed by a positive shift rule (*i.e.* promotion) is interpreted as the central strategy $\downarrow x \subseteq (\downarrow N)^{\perp} \multimap (\uparrow P_1 \parr \ldots \parr \uparrow P_k)$, equal to:

$$\{(*_1, \ldots, *_k) * n \cdot m \cdot n' * ' (*'_1, \ldots, *'_k) \mid (n, *_1, \ldots, *_k) \cdot m \cdot (n', *'_1, \ldots, *'_k) \in x\}. \quad (9)$$

2.1 Reversibility

All the constructions involved in the PBG semantics of MALLpol present a symmetry between the beginning and the end of terminated plays. This symmetry can be exploited to show a reversibility property of the semantics. We do not know if this reversibility property has a deep meaning.

Let $\overleftarrow{\ }$ be an operation reversing letters in words ($\overleftarrow{\varepsilon} = \varepsilon$ and $(s \cdot a)\overleftarrow{\ } = a \cdot \overleftarrow{s}$).

The *reverse* game of a PBG A is the PBG $\overleftarrow{A} = (\overline{\epsilon_A}, A^-, A^+, (S_A)\overleftarrow{\ })$.

The reverse of a strategy in A has no reason to be a strategy in $\overleftarrow{A}$ or in $\overleftarrow{A}^{\perp}$.

Proposition 6 (Reversibility). *If A is the PBG interpreting a MALLpol formula F without atoms then $A^{\perp} \cong \overleftarrow{A}$. Let ϕ be the associated tree isomorphism from $S_{A^{\perp}} = S_A$ to $S_{\overleftarrow{A}}$ (it just consists in exchanging the elements $*$ and $*'$ in each letter of each word). If x is the interpretation of a proof π of F in MALLpol then the two sets $\overleftarrow{x}$ and $\phi(x)$ are equal. Provided atoms enjoy the same property (on formulæ), this extends to MALLpol with atoms.*

2.2 Projection on Hypercoherences

We adapt the hypercoherence semantics of MALL to MALLpol as follows. If X is a hypercoherence then $\downarrow X$ (resp. $\uparrow X$) is the same hypercoherence, but where each element a of the web of X is renamed as the singleton multiset $[a]$. Hence $|\downarrow X|$ (resp. $|\uparrow X|$) equals $\{[a] \mid a \in |X|\}$. Up to this renaming of elements of the webs, shift rules leave unchanged interpretation of proofs in hypercoherences.

We define a projection of terminated plays in the interpretation of a MALLpol formula on hypercoherences, inductively as follows. For a formula A, p_A denotes the projection. We set $p_{A^{\perp}}(s) = p_A(s)$, $p_{\perp}(**') = *$, $p_{\alpha}(s \cdot a) = a$ and: if $A = B_1 \,\&\, B_2$ and $s \in S_{B_i}$ then $p_A(s) = p_{B_i}(s)$; if $A = B \parr C$ then $p_A(s) = (p_B(s\restriction B), p_C(s\restriction C))$; if $A = \uparrow B$ then $p_A(* \cdot s \cdot *') = [p_B(s\restriction B)]$.

Proposition 7. *Let F be a formula of MALLpol without atoms and π be a proof of F. Then p_F maps the PBG interpretation of π (resp. F) to the hypercoherence interpretation of π (resp. to the web of F). Provided atoms enjoy the same property (on formulæ), this extends to MALLpol with atoms.*

The projection of a strategy in the PBG interpretation of a formula is not, in general, a clique in the hypercoherence interpretation of this formula. For instance, let F be the formula $\downarrow \perp \otimes \downarrow \uparrow (1 \oplus 1)$. Let $A = \downarrow_4 \uparrow_3 (1_1 \oplus 1_2) \otimes \downarrow_6 \perp_5$

be the PBG interpretating F, where the indices separate the various copies of moves, and X be the hypercoherence interpretation of F. Let $s = (*_4, *_6) \cdot w_1 \cdot m \cdot (*'_4, *'_6)$ and $t = (*_4, *_6) \cdot m \cdot w_2 \cdot (*'_4, *'_6)$, where $m = *_2 *'_2$ and $w_i = *_3 *_i *'_i *'_3$ be two terminated plays in A. Then $x = \{s, t\}$ is a strategy but $p(x) = \{([[*'_1]], [*']), ([[*'_2]], [*'])\}$ is not a clique in X since $*'_1$ and $*'_2$ are strictly incoherent in the hypercoherence $1_1 \oplus 1_2$.

3 PBG Semantics of ILL and LLpol

A sub-category of NG, ANG, turns out to be a *new Seely* (see [4]) categorical model of ILL. Objects of ANG are affine negative PBGs, and morphisms are strategies containing the empty word. The tensor is interpreted by the negative tensor product. The terminal object of the category is then the unit of the negative tensor product. So, in this semantics, $\top = \mathbf{1}$. The PBGs interpreting formulæ are full but not terminated. ANG admits a comonad structure where the *"of course"*, $\sharp_1$, stands in-between the one of sequential algorithms and the Abrambsky-Jagadeesan-Malacaria ([1]), AJM, for short) or Hyland-Ong ([11], HO, for short) constructions. We describe its action on objects below.

By commutativity and associativity, the binary operation $\odot$ on words can be generalized to an n-ary operation on words on disjoint alphabets. We write $\bigodot\{p_1, \ldots, p_n\}$ for $p_1 \odot \cdots \odot p_n$. We adopt the convention that if $n = 0$ then the resulting set of words is $\{\varepsilon\}$ (which is neutral for $\odot$). We also generalize this operation in the case where alphabets are not disjoint. We set

$$\bigodot_A [p_1, \ldots, p_n] = \left(\bigodot_{A_1, \ldots, A_n} \{p_1 \times \{1\}, \ldots, p_n \times \{n\}\} \right) \restriction A \qquad (10)$$

where $[p_1, \ldots, p_n] \in \mathcal{M}_{\text{fin}}((A \cdot A)^*)$, $A_i = A \times \{i\}$ and where $w \times \{i\}$ is defined inductively by setting $(w' \cdot a) \times \{i\} = (w' \times \{i\}) \cdot (a, i)$ and $\varepsilon \times \{i\} = \varepsilon$. Projecting on A removes these indices. This generalization is well defined, since it does not depend on the enumeration of the elements of the multiset.

We define an embedding operation plg_A, from A^* to $(A^*)^*$, by $\text{plg}_A(\varepsilon) = \varepsilon$ and $\text{plg}_A(m' \cdot a) = \text{plg}_A(m') \cdot (m' \cdot a)$. This operation preserve the prefix ordering.

The PBG $\sharp_1 A$ is equal to $(-, P_A^-, P_A^+, S)$ where:

$$S = \bigcup \left\{ \bigodot_A [\text{plg}_A(p_i) \mid i \in I] \mid I \text{ finite and } \{p_i \mid i \in I\} \text{ strategy in } A \right\}. \qquad (11)$$

Observe that the following two plays in a AJM (resp. HO) *"of course"* game:

$$p = (a_1, 1)(a_2, 1)(a_1, 2)(a_2, 2)(a_3, 1) \quad (\text{resp. } a_1 \; a_2 \; a_1 \; a_2 \; a_3) \qquad (12)$$

$$q = (a_1, 1)(a_2, 1)(a_1, 2)(a_2, 2)(a_3, 2) \quad (\text{resp. } a_1 \; a_2 \; a_1 \; a_2 \; a_3) \qquad (13)$$

correspond to a unique play in the $\sharp_1$ construction:

$$p = q = (a_1)(a_1 a_2)(a_1)(a_1 a_2)(a_1 a_2 a_3). \qquad (14)$$

An operation eff erasing repetitions in words is defined by $\mathrm{eff}(s \cdot a) = \mathrm{eff}(s) \cdot a$ if $a \notin s$, $\mathrm{eff}(s \cdot a) = \mathrm{eff}(s)$ if $a \in s$ and $\mathrm{eff}(\varepsilon) = \varepsilon$.

The construction corresponding to the sequential algorithms' *"of course"* in bordered games would have been to take $\mathrm{eff}(S_{\sharp_1})$ instead of $S_{\sharp_1}$. We denote this construction by $\sharp_a$. Strangely enough $\sharp_a$ is not functorial for non terminated PBGs : in ILL the composition $\sharp_a f \fatsemi \sharp_a g$ may produce plays which are not terminated plays in $\sharp_a(f \fatsemi g)$.

PROPOSITION 8 below shows that the PBG semantics of MALLpol is also a semantics for LLpol (just take the comonad structure associated with $\sharp_1$).

Proposition 8. *If ANG is a new Seely category for a comonad* $(\sharp, \mathrm{dig}, \mathrm{der})$, *then our PBG model of MALLpol extends into a model of LLpol where* $!A = \downarrow\sharp A$.

The arrow of simple types $A \to B$ is interpreted by $(\downarrow\sharp A) \multimap B$ in the LLpol model and by $\sharp A \rightarrowtriangle B$ in the ILL model (these two games are isomorphic).

For the purpose of extending the PBG model of MALLpol to LLpol, one can also use the $\sharp_a$ construction. In fact, we do not really need $\sharp_a$ to be functorial for all objects of ANG but only for negative terminated PBGs as an hypothesis when proving the last proposition.

For $\sharp_1$, as for $\sharp_a$, the reversibility result (PROPOSITION 6) and projection result (PROPOSITION 7) do not extend to LLpol.

The natural way of extending the projection to exponentials would be to associate to a play of $!N$ the projection of its underlying finite strategy in N which is a finite set of points of N.

There are two reasons for not being able to extend PROPOSITION 7 to exponentials. First, with hypercoherences as target, the projection is not defined for all terminated plays. Second, if using the relational model as target, then, with $\sharp_1$ or $\sharp_a$ exponentials at the source, there are points in interpretation of formulæ and proofs which are not in the image of the projection. This is due to the fact that the hypercoherence model and our PBG models are *uniform*: they refer to the notion of agent in the interpretation of exponential formulæ.

For hypercoherences, the points of a $!N$ formula are the finite cliques of N and the projection of a strategy has no reason to be a clique. Hence there are plays in the PBG interpretation of $!N$ to which we cannot naturally associate a point in the hypercoherence $!N$. This prevents the projection from being defined on exponentials when using (standard) hypercoherences as target.

In fact, extending the projection to LLpol was our main motivation in introducing non uniform hypercoherences (see [5]).

With the $\sharp_1$ or $\sharp_a$ based exponentials interpretations in PBGs and with non uniform hypercoherences as target, the projection is well-defined. But the exponentials $\sharp_1$ and $\sharp_a$ require that a play in $!N$ is built from a finite strategy in N. Hence, there are points in $!N$ which are not the image of a play of $!N$ by the projection and such points can occur in the interpretation of proofs. For instance, the point $a = ([([*'], []), ([], [*'])])$ of $P = !(?1 \parr ?1)$ has no counterpart in the PBG interpretation of this formula. As a consequence, the projection of

the PBG interpretation of the identity proof of $\vdash P, P^{\perp}$ misses the point $([a], a)$ of its non uniform hypercoherence interpretation.

It is possible to use a non uniform Hyland-Ong style exponential for PBGs in which the reversibility and full projection properties extends to LLpol. This is a work in progress which supposes the introduction of pointers in PBGs such that, in a play, a move can point to several previous moves.

4 Hypercoherences Game Structures

Infinite coherence. Let X be a hypercoherence. An *infinitely coherent* (resp. infinitely incoherent) subset of X is a non empty directed union on Γ (resp. $\Gamma^{\perp}$) and this subset is strictly infinitely coherent if non reduced to a singleton. Observe that a coherent (resp. incoherent) subset of X is infinitely coherent (resp. infinitely incoherent). In absence of second order, all hypercoherences we use when doing semantics satisfy a convenient property of *local finiteness* we do not recall (see [9]). If X is locally finite then the set of strictly infinitely coherent subets of X, $\Delta^*(X)$, and the set of infinitely incoherent subsets of X, $\Delta(X)$ form a partition of the set of subsets of $|X|$ of cardinality greater than 2. From now on, hypercoherences are always supposed to be locally finite.

Towers. We define a binary relation, $\vdash_X$, on $\mathcal{P}^*(|X|)$ by setting $x \vdash_X y$ when $y \subsetneq x$ and either x is strictly infinitely incoherent in X and y is a maximal infinitely coherent subset of x or x is strictly infinitely coherent in X and y is a maximal infinitely incoherent subset of x.

A *move* on X is a vertex y of the directed acyclic graph $(\mathcal{P}^*(X), \vdash_X)$ such that there exists a directed path $|X| = x_1 \vdash_X x_2 \dots \vdash_X x_n = y$ from $|X|$ to y in this graph. Local finiteness of X implies that every oriented path in $(\mathcal{P}^*(|X|), \vdash_X)$ is finite. A *tower* on X is a (finite) directed path from $|X|$ to a singleton. A hypercoherence X is *serial parallel* if for each $a \in |X|$ there exists a unique tower on X ending on $\{a\}$.

The *tower graph* on X, denoted by $G(X)$, is the complete subgraph of $(\mathcal{P}^*(X), \vdash_X)$ whose set of vertices is the set $M(X)$ of moves on X. In a tower, moves alternate between $M(X) \cap \Delta^*$, the positive moves, and $M(X) \cap \Delta^{*\perp}$, called negative moves, except the last move, a singleton, which we equip with a relative polarity in the tower with respect to the alternation. The set of towers on X, denoted by $T(X)$, defines a tree, the *tower tree* of X.

A *negative hypercoherence* is a hypercoherence whose web contains at least two points and such that each tower ends on a positive move (*i.e.* singletons are always positive). *Positive* hypercoherences are defined dualy.

Definition 9. *If X is a polarized hypercoherence its associated PBG,* PBG(X), *is* $(\epsilon_X, M^-(X), M^+(X), S(X))$ *where* ϵ_X *is the polarity of* X, $M^-(X)$ *and* $M^+(X)$ *are respectively the set of negative and positive moves on* X *and* $S(X)$ *is the* game structure *of* X*: either* $T(X)$ *if* $|X| \in M^{\epsilon_X}(X)$ *or the forest obtained by erasing the root* $|X|$ *of* $T(X)$*, otherwise.*

Proposition 10. *If X and Y are disjoints polarized hypercoherences then:*

1. $\mathrm{PBG}(X^{\perp}) = \mathrm{PBG}(X)^{\perp}$,
2. *if* $X < 0$ *and* $Y < 0$ *then* $\mathrm{PBG}(X \,\&\, Y) = \mathrm{PBG}(X) \,\&\, \mathrm{PBG}(Y)$,
3. *if* $X > 0$ *and* $Y > 0$ *then* $\mathrm{PBG}(X \otimes Y) \cong \mathrm{PBG}(X) \otimes \mathrm{PBG}(Y)$,
4. *if* $X < 0$ *and* $Y < 0$ *then* $\mathrm{PBG}(X \otimes Y) \cong \mathrm{PBG}(X) \odot \mathrm{PBG}(Y)$ *et*
5. *if* $X < 0$ *and* $Y < 0$ *then* $\mathrm{PBG}(X \multimap Y) \cong \mathrm{PBG}(X) \multimap \mathrm{PBG}(Y)$.

By duality, this last case amounts to say that if $X < 0$ and $Y > 0$ then $\mathrm{PBG}(X \otimes Y) \cong {\downarrow}\mathrm{PBG}(X) \otimes \mathrm{PBG}(Y)$.

There is no construction in hypercoherences which correspond to the interpretation of polarity shifts $\downarrow$ and $\uparrow$ in PBGs through an equality like $\mathrm{PBG}({\downarrow} X) = {\downarrow}\mathrm{PBG}(X)$ because the PBGs built from hypercoherences are full and terminated but the polarity shift of a full and terminated non empty PBG is never full.

For the exponentials results are limited. First, to be able to describe simply the towers on $!X$ by means of operations on the set of words $T(X)$, we need to assume that X is serial parallel and this property is not preserved by logical connectives. Second $!X$ is not, in general, a polarized hypercoherence (even if X is negative and serial parallel). But, if we only consider exponentials inside intuitionistic implications (given by the equality $A \to B = !A \multimap B$) then this second limitation is circumvented.

Proposition 11. *If X and Y are two disjoint negative hypercoherences, and if X is serial parallel then* $\mathrm{PBG}(!X \multimap Y) \cong ({\downarrow}\sharp_a \mathrm{PBG}(X)) \multimap \mathrm{PBG}(Y)$.

5 Conclusion

In this paper, we present the polarized bordered game model together with a projection of games onto hypercoherences. This projection commutes to the interpretation of proofs in MALLpol. Extending this commutation result to exponentials requires non uniform game and hypercoherence models.

Our projection relates the sets of plays of games with the webs of hypercoherences, thanks to the introduction of a set of terminated plays. The coherence structures over these webs are still to be related with the dynamical structure of games. Our work on tower unfolding is a first attempt in that direction.

In [17], Melliès presents games as directed acyclic graphs from which he extracts hypercoherences. On simple types, the hypercoherence extracted from a game interpreting a type is the hypercoherence interpreting this type. His theory involves a partial projection from plays to points in hypercoherences which associates a web to each game, and an operation on graphs which allows to define coherences on these webs and relates strategies on graph games with cliques in associated hypercoherences. Our intuition about this last operation is that it is a reverse for the operation which, to each hypercoherence X, associates its tower graph $G(X)$ but forgets everything about moves except their polarities. We think that polarized bordered games and hypercoherences unfolding might help in extending Melliès' results to LLpol.

We would like to thank the referees and Pierre-Louis Curien for their suggestions on the presentation of this paper.

References

[1] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. In *Theoretical Aspects of Computer Software,* pages 1–15, 1994.

[2] Patrick Baillot, Vincent Danos, Thomas Ehrhard, and Laurent Regnier. Timeless games. In Mogens Nielsen and Wolfgang Thomas, editors, *Computer Science Logic,* volume 1414 of *Lecture Notes in Computer Science,* pages 56–77, Aahrus, Denmark, August 1997. European Association for Computer Science Logic, Springer-Verlag.

[3] Gérard Berry and Pierre-Louis Curien. Sequential algorithms on concrete data structures. *Theoretical Computer Science,* 20:265–321, 1982.

[4] G. M. Bierman. What is a categorical model of intuitionistic linear logic? In M. Dezani, editor, *Proceedings of Conference on Typed lambda calculus and Applications.* Springer-Verlag LNCS 902, 1995.

[5] Pierre Boudes. Non uniform hypercoherences. In Rick Blute and Peter Selinger, editors, *Electronic Notes in Theoretical Computer Science,* volume 69. Elsevier, 2003.

[6] Antonio Bucciarelli and Thomas Ehrhard. Sequentiality in an extensional framework. *Information and Computation,* 110(2), 1994.

[7] Thomas Ehrhard. Hypercoherences: a strongly stable model of linear logic. *Mathematical Structures in Computer Science,* 3, 1993.

[8] Thomas Ehrhard. A relative definability result for strongly stable functions and some corollaries. *Information and Computation,* 152, 1999.

[9] Thomas Ehrhard. Parallel and serial hypercoherences. *Theoretical computer science,* 247:39–81, 2000.

[10] Jean-Yves Girard. Linear logic. *Theoretical Computer Science,* 50:1–102, 1987.

[11] Martin Hyland and Luke Ong. On full abstraction for PCF: I, II and III. *Information and Computation,* 163(2):285–408, 2000.

[12] J. Laird. Games and sequential algorithms. Available by http, 2001.

[13] François Lamarche. Sequentiality, games and linear logic (announcement). In *Workshop on Categorical Logic in Computer Science.* Publications of the Computer Science Department of Aarhus University, DAIMI PB-397-II, 1992.

[14] Olivier Laurent. *Étude de la polarisation en logique.* Thèse de doctorat, Université Aix-Marseille II, March 2002.

[15] Olivier Laurent. Polarized games (extended abstract). In *Proceedings of the seventeenth annual IEEE symposium on Logic In Computer Science,* pages 265–274. IEEE Computer Society Press, July 2002.

[16] J.R. Longley. The sequentially realizable functionals. *Annals of Pure and Applied Logic,* 117(1-3):1–93, 2002.

[17] Paul-André Melliès. Sequential algorithms and strongly stable functions. To appear in the special issue of TCS: Game Theory Meets Theoretical Computer Science, 2003.

[18] Paul-André Melliès. Comparing hierarchies of types in models of linear logic. *Information and Computation,* 189(2):202–234, March 2004.

[19] Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science,* 11:207–260, 2001.

[20] Jaap van Oosten. A combinatory algebra for sequential functionals of finite type. Technical Report 996, University of Utrecht, 1997.

An Analog Characterization of Elementarily Computable Functions over the Real Numbers

Olivier Bournez and Emmanuel Hainry

LORIA/INRIA, 615 Rue du Jardin Botanique, BP101
54602 Villers lès Nancy, France
{Olivier.Bournez,Emmanuel.Hainry}@loria.fr

Abstract. We present an *analog* and *machine-independent* algebraic characterization of elementarily computable functions over the real numbers in the sense of recursive analysis: we prove that they correspond to the smallest class of functions that contains some basic functions, and closed by composition, linear integration, and a simple limit schema.
We generalize this result to all higher levels of the Grzegorczyk Hierarchy. Concerning *recursive analysis,* our results provide machine-independent characterizations of natural classes of computable functions over the real numbers, allowing to define these classes without usual considerations on higher-order (type 2) Turing machines. Concerning *analog models,* our results provide a characterization of the power of a natural class of analog models over the real numbers.

1 Introduction

Several approaches have been proposed to model computations over real numbers. *Recursive analysis* or *computable analysis,* was introduced by Turing [28], Grzegorczyk [12], Lacombe [15]. Alternative discrete-time computational models have also been investigated: see e.g. [4].

These models concern *discrete time* computability. Models of machines where the time is *continuous* can also be considered. The first ever built computers were continuous time machines: e.g. *Blaise Pascal's pascaline* or Lord Kelvin's model of *Differential Analyzer* [27], that gave birth to a real machine, built in 1931 at the MIT to solve differential equations [7], and which motivated Shannon's *General Purpose Analog Computer (GPAC) model* [25], whose computational power was characterized algebraically in terms of solutions of polynomial differential equations [25,23,16,11]. Continuous time machines also include analog neural networks [26], hybrid systems [3,5], or theoretical physical models [21,14, 10]: see also survey [22].

The relations between all the models are not fully understood. One can say, that the theory of analog computations has not yet experienced the unification that digital discrete time computations have experienced through Turing work and the so-called *Church thesis* [9,22].

This however becomes a crucial matter since the progress of electronics makes the construction of some of the machines realistic, whereas some models were

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 269–280, 2004.

recently proved very (far too?) powerful: using the so-called *Zeno's paradox,* some models make it possible to compute non-Turing computable functions in a constant time: see e.g. [17,6,3,14,10].

In [17], Moore introduced a class of functions over the reals inspired from the classical characterization of computable functions over integers: observing that the continuous analog of a primitive recursion is a differential equation, Moore proposes to consider the class of $\mathbb{R}$-*recursive functions,* defined as the the smallest class of functions containing some basic functions, and closed by composition, differential equation solving (called *integration*), and minimization.

This class of functions, also investigated in [18,19], can be related to GPAC computable functions: see [17], corrected by [11]. The original definitions of this class in [17] suffer from several technical problems, as well as also from some physical realizability problems providing the possibility of using super-Turing *"compression tricks".*

In his PhD dissertation, Campagnolo [9] proposes to restrict to the better-defined subclass $\mathcal{L}$ of $\mathbb{R}$-recursive functions corresponding to the smallest class of functions containing some basic functions and closed by composition and *linear* integration. Class $\mathcal{L}$ is related to functions elementarily computable over integers in classical recursion theory and functions elementarily computable over the real numbers in recursive analysis (discussed in [30]): any function of class $\mathcal{L}$ is elementarily computable in the sense of recursive analysis, and conversely, any function over the integers computable in the sense of classical recursion theory is the restriction to integers of a function that belongs to $\mathcal{L}$ [9,8].

However, the previous results do not provide a characterization of *all* functions over the reals that are computable in the sense of recursive analysis. This paper provides one: *for functions over the reals of class* $\mathcal{C}^2$ *defined on a product of compact intervals with rational endpoints,* f *is elementarily computable in the sense of recursive analysis iff it belongs to the smallest class of functions containing some basic functions and closed by composition, linear integration and a simple limit schema.* This can be extended to characterize all higher levels of the Grzegorczyk hierarchy: *for functions over the reals of class* $\mathcal{C}^2$ *defined on a product of compact intervals with rational endpoints,* f *is computable in the sense of recursive analysis in level* $n \geq 3$ *of the Grzegorczyk hierarchy iff* f *belongs to the smallest class of functions containing some (other) basic functions and closed by composition, linear integration and a simple limit schema.*

Concerning *analog models,* these results have several impacts: first, they contribute to understand analog models, in particular the relations between GPAC computable functions, $\mathbb{R}$-recursive functions, and computable functions in the sense of recursive analysis. Furthermore, they prove that no *Super-Turing* phenomena can occur for these classes of functions. In particular we have a "robust" class of functions in the sense of [13,2].

Concerning *recursive analysis,* our theorems provide *a purely algebraic* and *machine independent* characterization of elementarily computable functions over the reals. Observe the potential benefits offered by these characterizations com-

pared to classical definitions of these classes in recursive analysis, involving discussions about higher-order (type 2) Turing machines: see e.g. [29].

In Section 2, we start by some mathematical preliminaries. In Section 3, we recall some notions from classical recursion theory. We present basic definitions of recursive analysis in Section 4. Previous known results are recalled in Section 5. Our characterizations are presented in Section 6. The proofs are given in remaining sections.

2 Mathematical Preliminaries

Let $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{R}^{>0}$ denote the set of natural integers, the set of rational numbers, the set of real numbers, and the set of positive real numbers respectively. Given $x \in \mathbb{R}^n$, we write $\vec{x}$ to emphasize that x is a vector.

We will use the following simple mathematical result.

Lemma 1. *Let $F : \mathbb{R}\times\mathcal{V} \subset \mathbb{R}^2 \to \mathbb{R}^l$ be a function of class[1] $\mathcal{C}^1$, and $\beta(x) : \mathcal{V} \to \mathbb{R}$ be some continuous function. Assume that for all t and x, $\frac{\partial^2 F}{\partial t \partial x}(t,x)$ exists and $\|\frac{\partial F}{\partial t}(t,x)\| \leq K\exp(-t\beta(x))$, and $\|\frac{\partial^2 F}{\partial t \partial x}(t,x)\| \leq K\exp(-t\beta(x))$ for some constant $K > 0$.*

For all $x \in \mathcal{D}$, where $\mathcal{D}$ is the subset of the $x \in \mathcal{V}$ with $\beta(x) > 0$, $F(t,x)$ has a limit $L(x)$ in $t = +\infty$. Function $L(x)$ is of class $\mathcal{C}^1$, and its derivative L' is the limit of $\frac{\partial F(t,x)}{\partial x}(t,x)$ in $t = +\infty$. Furthermore

$$\|F(t,x) - L(x)\| \leq \frac{K\exp(-t\beta(x))}{\beta(x)} \quad \text{and} \quad \|\frac{\partial F}{\partial x}(t,x) - L'(x)\| \leq \frac{K\exp(-t\beta(x))}{\beta(x)}.$$

The following result[2], with previous lemma, is a key to provide upper bounds on the growth of functions of our classes (c.f. Lemma 4).

Lemma 2 (Bounding Lemma for Linear Differential Equations [1]). *For linear differential equation $\vec{x}' = A(t)\vec{x}$, if A is defined and continuous on interval $I = [a,b]$, where $a \leq 0 \leq b$, then, for all $\vec{x}_0$, the solution of $\vec{x}' = A(t)\vec{x}$ with initial condition $\vec{x}(0) = \vec{x}_0$ is defined and unique on I. Furthermore, the solution satisfies*

$$\|\vec{x}(t)\| \leq \|\vec{x}_0\|\exp(\sup_{\tau\in[0,t]} \|A(\tau)\|t).$$

3 Classical Recursion Theory

Classical recursion theory deals with functions over integers. Most classes of classical recursion theory can be characterized as closures of a set of basic functions by a finite number of basic rules to build new functions [24,20]: given a set $\mathcal{F}$ of

[1] Recall that function $f : \mathcal{D} \subset \mathbb{R}^k \to \mathbb{R}^l$, $k,l \in \mathbb{N}$, is said to be of class $\mathcal{C}^r$ if it is r-times continuously differentiable on $\mathcal{D}$.

[2] As it was already the case in Campagnolo's Dissertation.

functions and a set $\mathcal{O}$ of operators on functions (an operator is an operation that maps one or more functions to a new function), $[\mathcal{F};\mathcal{O}]$ will denote the closure of $\mathcal{F}$ by $\mathcal{O}$.

Proposition 1 (Classical settings: see e.g. [24,20]). *Let f be a function from $\mathbb{N}^k$ to $\mathbb{N}$ for $k \in \mathbb{N}$. Function f is*

- *elementary iff it belongs to $\mathcal{E} = [0, S, U, +, \ominus; \mathrm{COMP}, \mathrm{BSUM}, \mathrm{BPROD}]$;*
- *in class $\mathcal{E}_n$ of the Grzegorczyk Hierarchy ($n \geq 3$) iff it belongs to $\mathcal{E}_n = [0, S, U, +, \ominus, E_{n-1}; \mathrm{COMP}, \mathrm{BSUM}, \mathrm{BPROD}]$;*
- *primitive recursive iff it belongs to $\mathcal{PR} = [0, U, S; \mathrm{COMP}, \mathrm{REC}]$;*
- *recursive iff it belongs to $\mathcal{Rec} = [0, U, S; \mathrm{COMP}, \mathrm{REC}, \mathrm{MU}]$.*

A function $f : \mathbb{N}^k \to \mathbb{N}^l$ is elementary (resp: primitive recursive, recursive) iff its projections are elementary (resp: primitive recursive, recursive).

The base functions $0, (U_i^m)_{i,m\in\mathbb{N}}, S, +, \ominus$ *and the operators* COMP, BSUM, BPROD, REC, MU *are given by*

1. $0 : \mathbb{N} \to \mathbb{N}$, $0 : n \mapsto 0$; $U_i^m : \mathbb{N}^m \to \mathbb{N}$, $U_i^m : (n_1, \ldots, n_m) \mapsto n_i$; $S : \mathbb{N} \to \mathbb{N}, S : n \mapsto n+1$; $+ : \mathbb{N}^2 \to \mathbb{N}$, $+ : (n_1, n_2) \mapsto n_1 + n_2$; $\ominus : \mathbb{N}^2 \to \mathbb{N}$, $\ominus : (n_1, n_2) \mapsto max(0, n_1 - n_2)$;
2. BSUM : *bounded sum. Given* f, $h = \mathrm{BSUM}(f)$ *is defined by* $h : (\vec{x}, y) \mapsto \sum_{z<y} f(\vec{x}, z)$; BPROD : *bounded product. Given* f, $h = \mathrm{BPROD}(f)$ *is defined by* $h : (\vec{x}, y) \mapsto \prod_{z<y} f(\vec{x}, z)$;
3. COMP : *composition. Given* f *and* g, $h = \mathrm{COMP}(f, g)$ *is defined as the function verifying* $h(\vec{x}) = g(f(\vec{x}))$;
4. REC : *primitive recursion . Given* f *and* g, $h = \mathrm{REC}(f, g)$ *is defined as the function verifying* $h(\vec{x}, 0) = f(\vec{x})$ *and* $h(\vec{x}, n+1) = g(\vec{x}, n, h(\vec{x}, n))$.
5. MU : *minimization. The minimization of* f *is* $h : \vec{x} \mapsto \inf\{y : f(\vec{x}, y) = 0\}$.

Functions E_n, involved in the definition of the classes $\mathcal{E}_n$ of the Grzegorczyk Hierarchy, are defined by induction as follows (when f is a function, $f^{[d]}$ denotes its d-th iterate: $f^{[0]}(\vec{x}) = x$, $f^{[d+1]}(\vec{x}) = f(f^{[d]}(\vec{x}))$):

1. $E_0(x, y) = x + y$, $E_1(x, y) = (x+1)\times(y+1)$, $E_2(x) = 2^x$;
2. $E_{n+1}(x) = E_n^{[x]}(1)$ *for* $n \geq 2$.

We have $\mathcal{E} \subseteq \mathcal{PR} \subseteq \mathcal{Rec}$, and the inclusions are known to be strict [24,20]. It is also known that $\mathcal{E}_3 = \mathcal{E}$ and $\mathcal{PR} = \cup_i \mathcal{E}_i$ [24,20].If $\mathrm{TIME}(t)$ and $\mathrm{SPACE}(t)$ denote the classes of functions that are computable with time and space t, then, for all $n \geq 3$, $\mathcal{E}_n = \mathrm{TIME}(\mathcal{E}_n) = \mathrm{SPACE}(\mathcal{E}_n)$, and $\mathcal{PR} = \mathrm{TIME}(\mathcal{PR}) = \mathrm{SPACE}(\mathcal{PR})$ [24,20]. $\mathcal{PR}$ corresponds to functions computable using *loop programs*. $\mathcal{E}$ corresponds to computable functions bounded by some iterate of the exponential function [24,20].

In classical computability, more general objects than functions over the integers can be considered, in particular functionals, i.e. functions $\Phi : (\mathbb{N}^{\mathbb{N}})^m \times \mathbb{N}^k \to \mathbb{N}^l$. A functional will be said to be *elementary* (respectively. $\mathcal{E}_n$, *primitive recursive, recursive*) when it belongs to the corresponding[3] class.

[3] Formally, a function f over the integers can be considered as functional $\overline{f}$: $(V_1, \ldots, V_m, \vec{n}) \mapsto f(\vec{n})$. Similarly, an operator Op on functions $f_1, \ldots, f_m$

4 Computable Analysis

The idea sustaining *Computable analysis,* also called *recursive analysis,* is to define computable functions over real numbers by considering functionals over fast-converging sequences of rationals [28,15,12,29].

Formally, assume that a representation of rational numbers by integers is fixed[4]: let $\nu_{\mathbb{Q}}(r)$ be the rational represented by integer r. A product $\mathcal{C} = [a_1, b_1] \times \ldots \times [a_k, b_k]$ of compact intervals with rational endpoints can be encoded by an integer $\nu(\mathcal{C})$ encoding the list $\langle \nu_{\mathbb{Q}}(a_1), \nu_{\mathbb{Q}}(b_1), \ldots, \nu_{\mathbb{Q}}(a_k), \nu_{\mathbb{Q}}(b_k) \rangle$.

A sequence of integers $(x_i) \in \mathbb{N}^{\mathbb{N}}$ *represents a real number* x if it converges quickly toward x (denoted by $(x_i) \rightsquigarrow x$) in the following sense: $\forall i, |\nu_{\mathbb{Q}}(x_i) - x| < \exp(-i)$. For $X = ((x_1), \ldots, (x_k)) \in (\mathbb{N}^{\mathbb{N}})^k$, $\vec{x} \in \mathbb{R}^k$, we write $X \rightsquigarrow \vec{x}$ for $(x_i) \rightsquigarrow U_i^k(\vec{x})$ for $i = 1, \ldots, k$.

Definition 1 (Recursive analysis). *A function $f : \mathcal{D} \to \mathbb{R}$, where $\mathcal{D} \subset \mathbb{R}^k$ is a product of compact intervals with rationals endpoints, is said to be computable (in the sense of recursive analysis) if there exists a recursive functional $\phi : (\mathbb{N}^{\mathbb{N}})^k \times \mathbb{N} \to \mathbb{N}$ such that for all $\vec{x} \in \mathcal{D}$, for all $X \in (\mathbb{N}^{\mathbb{N}})^k$, we have $(\phi(X, j))_j \rightsquigarrow f(\vec{x})$ whenever $X \rightsquigarrow \vec{x}$.*

A function $f : \mathcal{D} \to \mathbb{R}$, where $\mathcal{D} \subset \mathbb{R}^k$ is not necessarily compact, is said to be computable if there exists a recursive functional $\phi : (\mathbb{N}^{\mathbb{N}})^k \times \mathbb{N}^2 \to \mathbb{N}$ such that for all product $\mathcal{C}$ of compact intervals with rational endpoints included in $\mathcal{D}$, $\forall \vec{x} \in \mathcal{C}$, for all $X \in (\mathbb{N}^{\mathbb{N}})^k$, we have $(\phi(X, \nu(\mathcal{C}), j))_j \rightsquigarrow f(\vec{x})$ whenever $X \rightsquigarrow \vec{x}$.

A function $f : \mathcal{D} \to \mathbb{R}^l$, with $l > 1$, is said to be computable if all its projections are.

A function f will be said to be *elementarily* (respectively $\mathcal{E}_n$) computable whenever the corresponding functional Φ is. The class of elementarily (respectively $\mathcal{E}_n$) computable functions over the reals will be denoted by $\mathcal{E}(\mathbb{R})$ (resp. $\mathcal{E}_n(\mathbb{R})$). Observe that elementarily computable functions were discussed in [30].

5 Real-Recursive and Recursive Functions

Following the original ideas from [17], but avoiding the minimization schema, Campagnolo proposed in [9] to consider the following class, built in analogy with elementarily computable functions over the integers (a real extension of a function $f : \mathbb{N}^k \to \mathbb{N}^l$ is a function $\tilde{f}$ from $\mathbb{R}^k$ to $\mathbb{R}^l$ whose restriction to $\mathbb{N}^k$ is f).

over the integers can be extended to $\overline{Op}(F_1, \ldots, F_m) : (V_1, \ldots, V_m, \vec{n}) \mapsto Op(F_1(V_1, \ldots, V_m, .), \ldots, F_m(V_1, \ldots, V_m, .))(\vec{n})$. We will still (abusively) denote by $[f_1, \ldots, f_p; O_1, \ldots, O_q]$ for the smallest class of functionals that contains basic functions $\overline{f_1}, \ldots, \overline{f_p}$, plus the functionals $Map_i : (V_1, \ldots, V_m, n) \to (V_i)_n$, the nth element of sequence V_i, and which is closed by the operators $\overline{O_1}, \ldots, \overline{O_q}$. For example, a functional will be said to be elementary iff it belongs to $\mathcal{E} = [Map, \overline{0}, \overline{S}, \overline{U}, \overline{+}, \overline{\ominus}; \overline{\mathrm{COMP}}, \overline{\mathrm{BSUM}}, \overline{\mathrm{BPROD}}]$.

[4] We will assume that in this representation, the basic functions on rationals +, –, ×, / are elementarily computable.

Definition 2 ([9,8]). *Let $\mathcal{L}$ and $\mathcal{L}_n$ be the classes of functions $f : \mathbb{R}^k \to \mathbb{R}^l$, for some $k, l \in \mathbb{N}$, defined by $\mathcal{L} = [0, 1, -1, \pi, U, \theta_3; \mathrm{COMP}, \mathrm{LI}]$ and $\mathcal{L}_n = [0, 1, -1, \pi, U, \theta_3, \overline{E_{n-1}}; \mathrm{COMP}, \mathrm{LI}]$ where the base functions 0, 1, –1, π, $(U_i^m)_{i,m\in\mathbb{N}}$, θ_3, $\overline{E_n}$ and the schemata* COMP *and* LI *are defined as follows:*

1. *0,1,–1, π are the corresponding constant functions; $U_i^m : \mathbb{R}^m \to \mathbb{R}$ are, as in the classical settings, projections: $U_i^m : (x_1, \ldots, x_m) \mapsto x_i$;*
2. *$\theta_3 : \mathbb{R} \to \mathbb{R}$ is defined as $\theta_3 : x \mapsto x^3$ if $x \geq 0, 0$ otherwise.*
3. *$\overline{E_n}$: for $n \geq 3$, let $\overline{E_n}$ denote a monotone real extension of the function $\exp_n$ over the integers defined inductively by $\exp_2(x) = 2^x$, $\exp_{i+1}(x) = \exp_i^{[x]}(1)$.*
4. COMP: *composition is defined as in the classical settings: Given f and g, $h = \mathrm{COMP}(f, g)$ is the function verifying $h(\vec{x}) = g(f(\vec{x}))$;*
5. LI: *linear integration. From g and h, $\mathrm{LI}(g, h)$ is the maximal solution of the linear differential equation $\frac{\partial f}{\partial y}(\vec{x}, y) = h(\vec{x}, y) f(\vec{x}, y)$ with $f(\vec{x}, 0) = g(\vec{x})$.*
   In this schema, if g goes to $\mathbb{R}^n$, $f = \mathrm{LI}(g, h)$ also goes to $\mathbb{R}^n$ and $h(\vec{x}, y)$ is a $n \times n$ matrix with elements in $\mathcal{L}$.

These classes contain functions $id : x \mapsto x$, sin, cos, exp, $+, \times$, $x \mapsto r$ for all rational r, as well as for all $f \in \mathcal{L}$, or $f \in \mathcal{L}^*$, its primitive function F equal to $\vec{0}$ at $\vec{0}$, denoted by $\int(f)$. Indeed, function *id* is given by LI(1, 0). Function $\Theta : t \mapsto (\sin(t), \cos(t))$ can be defined by $\mathrm{LI}\left(\begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0 & 1\\-1 & 0\end{bmatrix}\right)$. Project this function on each of its two variables to get sinus and cosinus function. Function exp is given by $\mathrm{LI}(1, id)$. Addition is given by $x + 0 = x$, $\frac{\partial x+y}{\partial y} = 1$. Multiplication is given by $x \times 0 = 0$, $\frac{\partial x \times y}{\partial y} = x$. $\int(f)$ can be defined by $\begin{pmatrix}F\\1\end{pmatrix} = \mathrm{LI}\left(\begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0 & f\\0 & 0\end{bmatrix}\right)$. Given $p, q \in \mathbb{N}$ with $q > 0$, Function $x \mapsto p$, is $1 + 1 + \ldots + 1$, function $x \mapsto x^{q-1}$ is $x \times \ldots \times x$, and $p \times \int(x \mapsto x^{q-1})$ is $x \mapsto px^q/q$ whose value in 1 is p/q.

Proposition 2 ([8]). *All functions from $\mathcal{L}$ are continuous, defined everywhere, and of class $\mathcal{C}^2$.*

The previous classes can be partially related to classes $\mathcal{E}(\mathbb{R})$ and $\mathcal{E}_n(\mathbb{R})$:

Proposition 3 ([9,8]).

- 1. *$\mathcal{L} \subset \mathcal{E}(\mathbb{R})$: any function from $\mathcal{L}$ is elementarily computable over real numbers.*
  2. *"$\mathcal{E} \subset \mathcal{L}$": any elementarily computable function over the integers, has a real extension that belongs to $\mathcal{L}$.*
- 1. *$\mathcal{L}_n \subset \mathcal{E}_n(\mathbb{R})$: any function from $\mathcal{L}_n$ is $\mathcal{E}_n$-computable.*
  2. *"$\mathcal{E}_n \subset \mathcal{L}_n$": any $\mathcal{E}_n$-computable function over the integers, has a real extension that belongs to $\mathcal{L}_n$.*

Although Proposition 3 gives the inclusions $\mathcal{L} \subset \mathcal{E}(\mathbb{R})$ and $\mathcal{L}_n \subset \mathcal{E}_n(\mathbb{R})$, it fails to characterize completely $\mathcal{E}(\mathbb{R})$ and $\mathcal{E}_n(\mathbb{R})$: these inclusions are strict. Indeed, $x \mapsto 1/x$ is elementarily computable while Proposition 2 says that all functions from $\mathcal{L}$ are defined everywhere. A similar argument works for $\mathcal{E}_n(\mathbb{R})$. We conjecture the inclusions to be strict even when restricting to total functions.

6 Real-Recursive and Recursive Functions Revisited

We now propose to consider new classes of functions that we will prove to correspond precisely to $\mathcal{E}(\mathbb{R})$ and $\mathcal{E}_n(\mathbb{R})$.

First, we modify a little bit the composition schema, since (non-total) elementarily computable functions are not stable by composition.

Definition 3 (COMP schema). *Given f,g, if there is a product of closed intervals[5] C with rational or infinite endpoints with $Range(f) \subset C \subset Domain(g)$, then function $\mathrm{COMP}(f,g)$ is defined. It is defined by $\mathrm{COMP}(f,g) : \vec{x} \mapsto g(f(\vec{x}))$ on all $\vec{x}$ where $f(\vec{x})$ and $g(f(\vec{x}))$ exist.*

Now, we suggest to add a limit operator denoted by LIM, inspired by Lemma 1: a polynomial β over $\mathbb{R}$ is a function of the form $\beta : \mathbb{R} \to \mathbb{R}$, $\beta : x \mapsto \sum_{i=0}^{n} a_i x^i$ for some $a_0, \ldots, a_n \in \mathbb{R}$.

Definition 4 (LIM schema). *Let $f : \mathbb{R}\times\mathcal{D} \subset \mathbb{R}^2 \to \mathbb{R}$, and $\beta : \mathcal{D} \to \mathbb{R}$ a polynomial with the following hypothesis: there exists a constant K such that for all t,x, $\|\frac{\partial f}{\partial t}(t,x)\| \leq K\exp(-t\beta(x))$, $\frac{\partial^2 f}{\partial t \partial x}(t,x)$ exists, and $\|\frac{\partial^2 f}{\partial t \partial x}(t,x)\| \leq K\exp(-t\beta(x))$.*

Then, for every interval $I \subset \mathbb{R}$ on which $\beta(x) > 0$, $F = \mathrm{LIM}(f,\beta)$ is defined as the function $F : I \to \mathbb{R}$, with $F(x) = \lim_{t\to+\infty} f(t,x)$, under the condition that it is of class[6] $\mathcal{C}^2$.

We are ready to define our classes:

Definition 5 (Classes $\mathcal{L}^*$, $\mathcal{L}_n^*$). *The class $\mathcal{L}^*$, and $\mathcal{L}_n^*$, for $n \geq 3$, of functions from $\mathbb{R}^k$ to $\mathbb{R}^l$, for $k,l \in \mathbb{N}$, are following classes:*

- $\mathcal{L}^* = [0, 1, -1, U, \theta_3; \mathrm{COMP}, \mathrm{LI}, \mathrm{LIM}]$.
- $\mathcal{L}_n^* = [0, 1, -1, U, \theta_3, \overline{E_{n-1}}; \mathrm{COMP}, \mathrm{LI}, \mathrm{LIM}]$.

Example 1. Previous classes can easily be shown stable by the primitive operator that sends a function f to its primitive $\int(f)$ equal to $\vec{0}$ at $\vec{0}$.

Class $\mathcal{L}^*$ also includes some non-total functions, in particular the function $\frac{1}{x} : \mathbb{R}^{>0} \to \mathbb{R}$, $\frac{1}{x} : x \mapsto \frac{1}{x}$: indeed, $\int(\exp(-tx))$ is function $E : (t,x) \mapsto \frac{(1-\exp(-tx))}{x}$ for $x \neq 0$, t for $x = 0$ (of class $\mathcal{C}^k$ for all k). Now $\frac{1}{x} = \mathrm{LIM}(E, id)$.

Proposition 4. *$\mathcal{L} \subsetneq \mathcal{L}^*$, $\mathcal{L}_n \subsetneq \mathcal{L}_n^*$ for all $n \geq 3$.*

Proof. The function $x \mapsto \pi$ is actually in $\mathcal{L}^*$. Indeed, from $x \mapsto \frac{1}{1+x^2}$ in the class, we have $\arctan x = \int(\frac{1}{1+x^2})$, and $\pi = 4\arctan(1)$. Observing that our composition schema for total functions subsumes the composition schema of class $\mathcal{L}$, the result follows.

[5] That can be $\mathbb{R}^k$ when g is total.

[6] If f is of class $\mathcal{C}^1$, function F exists and is at least of class $\mathcal{C}^1$ by Lemma 1.

The main results of this paper are the following (proved in following two sections):

Theorem 1 (Characterization of $\mathcal{E}(\mathbb{R})$). *Let $f : \mathcal{D} \subset \mathbb{R}^k \to \mathbb{R}^l$ be some function over the reals of class $\mathcal{C}^2$, with $\mathcal{D}$ product of compact intervals with rational endpoints. f is in $\mathcal{E}(\mathbb{R})$ iff it belongs to $\mathcal{L}^*$.*

Theorem 2 (Characterization of $\mathcal{E}_n(\mathbb{R})$). *Let $f : \mathcal{D} \subset \mathbb{R}^k \to \mathbb{R}^l$ be some function over the reals of class $\mathcal{C}^2$, with $\mathcal{D}$ product of compact intervals with rational endpoints. Let $n \geq 3$. f is in $\mathcal{E}_n(\mathbb{R})$ iff it belongs to $\mathcal{L}^*_n$.*

7 Upper Bounds

We now prove the upper bound $\mathcal{L}^* \subset \mathcal{E}(\mathbb{R})$. As one may expect, this direction of the proof has many similarities with the proof $\mathcal{L} \subset \mathcal{E}$ in [9,8]: main differences lie in the presence of non-total functions and of schema LIM.

A structural induction shows:

Lemma 3. *All functions from $\mathcal{L}^*$ are of class $\mathcal{C}^2$ and defined on a domain of the form $I_1 \times I_2 \ldots \times I_k$ where each I_i is an interval.*

We propose to introduce the following notation: given $a \in \mathbb{R}$, let ρ_a be the function $x \mapsto \frac{1}{x-a}$. Let $\rho_{+\infty}$ and $\rho_{-\infty}$ be the function identity $x \mapsto x$.

Given I real interval with bounds $a, b \in \mathbb{R} \cup \{-\infty, +\infty\}$, $\rho_I(x) = |\rho_a(x)| + |\rho_b(x)|$. For $\mathcal{D} = I_1 \times I_2 \ldots \times I_k$, let $\rho_{\mathcal{D}}(x) = \rho_{I_1}(U_1^k(x)) + \ldots + \rho_{I_k}(U_k^k(x))$. In any case, $\rho_{\mathcal{D}}(x)$ is elementarily computable and grows to $+\infty$ when x gets close to a bound of domain $\mathcal{D}$.

The following Lemma is an extension of a Lemma of [9,8] (it is proved by structural induction using Lemma 1 for schema LIM, Lemma 2 for schema LI, plus the fact that it is always possible to assume that the degree of a product or a sum of two functions f and g is less than the maximum of their degrees).

Lemma 4. *Let $f : \mathcal{D} \subset \mathbb{R}^k \to \mathbb{R}^l$ be a function of $\mathcal{L}^*$. There exist some integer d, and some constants A and B such that for all $\vec{x} \in \mathcal{D}$, $\|f(\vec{x})\| \leq A \exp^{[d]}(B\rho_{\mathcal{D}}(\vec{x}))$. Call the smallest such integer d the degree of f. All the partial derivatives of f also have a finite degree.*

We are ready to prove the upper bound.

Proposition 5. $\mathcal{L}^* \subseteq \mathcal{E}(\mathbb{R})$.

Proof. — The basic functions $0, 1, -1, U, \theta_3$ are easily shown elementarily computable.

- When $h = \mathrm{COMP}(f, g)$, f and g elementarily computable, then h is also elementarily computable: indeed, there exists some closed set F with $Range(f) \subset F \subset Domain(g)$. Adapting the constructions in [29], given a product of compact intervals $\mathcal{C}$ with rational endpoints included in $Domain(f)$, we can compute elementarily a product of compact intervals $\mathcal{C}'$ with rational endpoints with $f(\mathcal{C}) \subset \mathcal{C}'$. Now, for $x \in \mathcal{C}$, compose the functional that computes g on $\mathcal{C}' \cap F$ with the one that computes f on $\mathcal{C}$.

- Let $g = \mathrm{LIM}(f, \beta)$, with f computed by elementary functional ϕ. We give the proof for f defined on $\mathbb{R}\times\mathcal{C}$ where $\mathcal{C}$ is a compact interval of $\mathbb{R}$. The general case is easy to obtain.
  Let $x \in \mathbb{R}$, with $\beta(x) > 0$. Since $\beta(x)$ is a polynomial, $1/\beta(x)$ can be bounded elementarily by some computable integer N in some computable neighborhood of x.
  Let $(x_n) \rightsquigarrow x$. For all $i, j \in \mathbb{N}$, if we write (i) for the constant sequence $k \mapsto i$, we have $|\nu_{\mathbb{Q}}(\phi((i),(x_n),j)) - f(i,x)| < \exp(-j)$.
  By Lemma 1, we have $\|f(i,x) - g(x)\| \leq \frac{K\exp(-\beta(x)i)}{\beta(x)} \leq KN\exp(-\beta(x)i)$.
  Hence, $|\nu_{\mathbb{Q}}(\phi((i),(x_n),j)) - g(x)| < \exp(-j) + KN\exp(-\beta(x)i)$.
  If we take $j' = j+1$, $i' = N(j+1+\lceil\ln(KN)\rceil)$, we have $exp(-j') \leq \exp(-j)/2$, and $KN\exp(-\beta(x)i') \leq \exp(-j)/2$. Hence g is computed by the functional $\psi : ((x_n),j) \mapsto \phi((N(j+1+\lceil\ln(KN)\rceil)),(x_n),j+1)$.
- Let $f = \mathrm{LI}(g,h)$. The proof for this case is very similar to [9,8].
  This ends the proof.

Replacing in previous proofs the bounds of Lemma 4 by bounds of type $\|f(\vec{x})\| \leq A\overline{E}^{[d]}_{n-1}(B\rho_{\mathcal{D}}(\vec{x}))$, one can also obtain.

Proposition 6. $\forall n \geq 3$, $\mathcal{L}^*_n \subseteq \mathcal{E}_n(\mathbb{R})$.

8 Lower Bounds

We will now consider the opposite inclusion: $\mathcal{E}(\mathbb{R}) \subseteq \mathcal{L}^*$, proved for functions of class $\mathcal{C}^2$ on compact domains with rational endpoints.

Let $\epsilon > 0$ be some real. We write $\mathbb{N}\epsilon$ for the set of reals of the form $i\epsilon$ for some integer i. Given $y \in \mathbb{R}$, write $\lfloor y \rfloor_\epsilon$ for the unique $j\epsilon$ with j integer and $y \in [j\epsilon, j\epsilon+\epsilon)$.

Lemma 5. *Let $\epsilon : \mathbb{R} \to \mathbb{R}$ be some decreasing elementarily computable function, with $\epsilon(x) > 0$ for all x and going to 0 when x goes to $+\infty$. Write ϵ_i for $\epsilon(\lfloor i \rfloor)$.*

Given $f : \mathbb{R}^2 \to \mathbb{R}^l$ in $\mathcal{L}^*$, there exists $F : \mathbb{R}^2 \to \mathbb{R}^l$ in $\mathcal{L}^*$ with the following properties:

- *For all $i \in \mathbb{N}$, $x \in \mathbb{N}\epsilon_i$, $F(i,x) = f(i,x)$*
- *For all $i \in \mathbb{N}$, $x \in \mathbb{R}$, $\|F(i,x) - f(i,\lfloor x \rfloor_{\epsilon_i})\| \leq \|f(i,\lfloor x \rfloor_{\epsilon_i} + \epsilon_i) - f(i,\lfloor x \rfloor_{\epsilon_i})\|$*
- *For all $i \in \mathbb{R}$, $x \in \mathbb{R}$, $\|\frac{\partial F}{\partial i}(i,x)\| \leq 5\|f(\lfloor i+1 \rfloor, \lfloor x \rfloor_{\epsilon_i}) - f(\lfloor i \rfloor, \lfloor x \rfloor_{\epsilon_i})\| + 25\|f(\lfloor i \rfloor, \lfloor x \rfloor_{\epsilon_i} + \epsilon_i) - f(\lfloor i \rfloor, \lfloor x \rfloor_{\epsilon_i})\| + 25\|f(\lfloor i+1 \rfloor, \lfloor x \rfloor_{\epsilon_{i+1}} + \epsilon_{i+1}) - f(\lfloor i+1 \rfloor, \lfloor x \rfloor_{\epsilon_{i+1}})\|$.*

Proof (Sketch). Let $\zeta = \frac{3\pi}{2}$. Let $\omega : x \mapsto \zeta\theta_3(\sin(2\pi x))$. $\forall i, \int_i^{i+1}\omega = 1$ and ω is equal to 0 on $[i+\frac{1}{2}, i+1]$ for every $i \in \mathbb{N}$. Let $\Omega = \int(\omega)$ its primitive, and $int : x \mapsto \Omega(x - \frac{1}{2})$. The function *ink* is similar to the integer part: $\forall i$, $\forall x \in [i, i+\frac{1}{2}]$, $int(x) = i = \lfloor x \rfloor$. Let $\Delta(i,x) = f(i, x+\epsilon(i)) - f(i,x)$.

Let G be the solution of the linear differential equation $G(i,0) = f(i,0)$, $\frac{\partial G}{\partial x}(i,x) = \frac{\omega(x/\epsilon(i))}{\epsilon(i)}\Delta(i, \epsilon(i)int(x/\epsilon(i)))$. An easy induction on j then shows that $G(i, j\epsilon(i)) = f(i, j\epsilon(i))$ for all integer j.

Then, let $\Delta'(i,x) = G(i+1,x) - G(i,x)$.

Let F be the solution of the linear differential equation $F(0,x) = G(0,x)$, $\frac{\partial F}{\partial i} = \omega(i)\Delta'(int(i),x)$. By induction, we have $F(i,x) = f(i,x)$ for all $i \in \mathbb{N}$, $x \in \mathbb{N}\epsilon_i$.

Some technical computations allow to conclude that function F satisfies all the claims.

We are now ready to prove the missing inclusion of Theorem 1.

Proposition 7. *Let $f : \mathcal{D} \subset \mathbb{R}^k \to \mathbb{R}^l$ be some function over the reals of class $\mathcal{C}^2$, with $\mathcal{D}$ product of compact intervals with rational endpoints. If f is $\mathcal{E}(\mathbb{R})$, then it belongs to $\mathcal{L}^*$.*

Proof. We give the proof for a function f defined on interval [0,1] to $\mathbb{R}$. The general case is easy to obtain.

Since f'' is continuous on a compact set, f'' is bounded by some constant M. By mean value theorem, we have $|f'(x) - f'(y)| \leq M|x-y|$ for all x, y.

Given i, consider n with $\exp(n)\exp(-i) \geq 4M$ and $\exp(-n) \leq 1/4$. For all j, consider $x_j = j\exp(-n)$, so that for all $x, y \in [x_j, x_{j+1}]$ we have $|f'(x) - f'(y)| \leq \exp(-i)/4$.

For all j, let y_j be some rational number at most $\exp(-i)/2$ far from $f(x_j)$ and $z_j = (y_{j+1} - y_j)\exp(-n)$. By mean value theorem, there exists $\chi_j \in [x_j, x_{j+1}]$ such that $f'(\chi_j) = (f(x_{j+1}) - f(x_j))/\exp(n)$. So, $|z_j - f'(\chi_j)| \leq \exp(-i)/4$ for some $\chi_j \in [x_j, x_{j+1}]$, which implies $|f'(\chi_j) - f'(x_j)| < \exp(-i)/4$, and so, z_j is at most $\exp(-i)/2$ far from $f'(x_j)$. Let $p_j, q_j \in \mathbb{N}$ such that $p_j \times \exp(-q_j)$ is at most $\exp(-i)/2$ far from z_j, hence, at most $\exp(-i)$ far from $f'(x_j)$. Observing that the y_j, and so the z_j can be elementarily obtained from i and j, the functions $p_{\mathbb{N}} : \mathbb{N}^2 \to \mathbb{N}$, and $q_{\mathbb{N}} : \mathbb{N}^2 \to \mathbb{N}$ that map (i,j) to corresponding p_j and q_j are elementarily computable. By Proposition 3, they can be extended to function $p : \mathbb{R}^2 \to \mathbb{R}$ and $q : \mathbb{R}^2 \to \mathbb{R}$ in $\mathcal{L}$. Consider function $g : \mathbb{R}\times[0,1] \to \mathbb{R}$ defined on all $(i,x) \in \mathbb{R}\times[0,1]$ by $g(i,x) = p(i,\exp(n)x)\exp(-q(i,\exp(n)x))$. By construction, for i, j integer, we have $g(i,x_j) = p_j\exp(-q_j)$.

Consider the function F given by Lemma 5 for function g and $\epsilon : n \mapsto \exp(-n)$. We have $F(i,x_j) = z_j$ for all $i, j \in \mathbb{N}$.

For all integer i, and all $x \in \mathbb{R}$, we have

$$\begin{aligned}
\|F(i,x) - f'(x)\| &\leq \|F(i,x) - F(i,\lfloor x\rfloor_\epsilon)\| + \|F(i,\lfloor x\rfloor_\epsilon - g(i,\lfloor x\rfloor_\epsilon)\| \\
&\quad + \|g(i,\lfloor x\rfloor_\epsilon) - f'(\lfloor x\rfloor_\epsilon)\| + \|f'(\lfloor x\rfloor_\epsilon) - f'(x)\| \\
&\leq \|F(i,\lfloor x\rfloor_\epsilon + \epsilon_i) - F(i,\lfloor x\rfloor_\epsilon)\| + 0 + \exp(-i) + M\epsilon_i \\
&\leq \|f'(x_{j+1}) - g(i,x_{j+1})\| + \|f'(x_j) - g(i,x_j)\| \\
&\quad + \|f'(x_{j+1}) - f'(x_j)\| + \exp(-i) + \exp(-i)/4 \\
&\leq 3\times\exp(-i) + \exp(-i) + \exp(-i)/4 \\
&\leq 5\times\exp(-i)
\end{aligned}$$

Consider the function $G : \mathbb{R}^2 \to \mathbb{R}$ defined for all $i, x \in \mathbb{R}$ by the linear differential equation $G(i,0) = f(0)$[7] and $\frac{\partial G}{\partial x}(i,x) = F(i,x)$. For all integer i, we

[7] A technique similar to the one we use here to get function f', can be used to show that $f(0)$ is always in $\mathcal{L}^*$.

have $G(i,0)-f(0)=0$ and $\|\frac{\partial G}{\partial x}(i,x)-f'(x)\| = \|F(i,x)-f'(x)\| \leq 5\times\exp(-i)$. By mean value theorem on function $G(i,x)-f(x)$, we get $\|G(i,x)-f(x)\| \leq 5\times\exp(-i)$ on [0,1]. Hence, $f(x)$ is the limit of $G(i,x)$ when i goes to $+\infty$ with integer values. We just need to check that schema LIM can be applied to function G of $\mathcal{L}^*$ to conclude: indeed, the limit of $G(i,x)$ when i goes to $+\infty$ will exist and coincide with this value, i.e. $f(x)$.

Since $\frac{\partial G}{\partial x} = F$, and hence $\|\frac{\partial^2 G}{\partial i \partial x}\| = \|\frac{\partial F}{\partial i}\|$ and since $\frac{\partial G}{\partial i} = \int_0^x \frac{\partial F}{\partial i}(i,x)dx$ implies $\|\frac{\partial G}{\partial i}\| \leq \int_0^1 \|\frac{\partial F}{\partial i}\|dx \leq \|\frac{\partial F}{\partial i}\|$ we only need to prove that we can bound $\|\frac{\partial F}{\partial i}\|$ by $K\times\exp(-i)$ for a constant K. But from Lemma 5, we know that for all i,x, $\|\frac{\partial F}{\partial i}(i,x)\| \leq 5\|g(\lfloor i+1\rfloor, \lfloor x\rfloor_{\epsilon_i}) - g(\lfloor i\rfloor, \lfloor x\rfloor_{\epsilon_i})\| + 25\|g(\lfloor i\rfloor, \lfloor x\rfloor_{\epsilon_i}+\epsilon_i) - g(\lfloor i\rfloor, \lfloor x\rfloor_{\epsilon_i})\| + 25\|g(\lfloor i+1\rfloor, \lfloor x\rfloor_{\epsilon_{i+1}}+\epsilon_{i+1}) - g(\lfloor i+1\rfloor, \lfloor x\rfloor_{\epsilon_{i+1}})\|$. First term can be bounded by $5\times\exp(-i)+5\times\exp(-i) = 10\times\exp(-i)$. Second term can be bounded by $25(\|g(\lfloor i\rfloor, \lfloor x\rfloor_{\epsilon_i}+\epsilon_i) - f'(\lfloor x\rfloor_{\epsilon_i}+\epsilon_i)\| + \|f'(\lfloor x\rfloor_{\epsilon_i}+\epsilon_i) - f'(\lfloor x\rfloor_{\epsilon_i})\| + \|g(\lfloor i\rfloor, \lfloor x\rfloor_{\epsilon_i}) - f'(\lfloor x\rfloor_{\epsilon_i})\|) \leq 25\times\exp(-i) + 25\times\exp(-i) + 25\times\exp(-i) = 75\times\exp(-i)$. Similarly for third term, replacing i by $i+1$.

Hence $\|\frac{\partial F}{\partial i}(i,x)\| \leq 160\times\exp(-i)$, and so schema LIM can be applied on function G of $\mathcal{L}^*$ to get function f. This ends the proof.

The missing inclusion of Theorem 2 can be proved similarly for all levels $n \geq 3$ of the Grzegorczyk hierarchy.

Proposition 8. *Let $f : \mathcal{D} \subset \mathbb{R}^k \to \mathbb{R}^l$ be some function over the reals of class $\mathcal{C}^2$, with $\mathcal{D}$ product of compact intervals with rational endpoints. If f is $\mathcal{E}_n(\mathbb{R})$, for $n \geq 3$, then it belongs to $\mathcal{L}_n^*$.*

Remark 1.
- We have actually a *normal form theorem:* previous proof shows that every function of $\mathcal{L}^*$ and $\mathcal{L}_n^*$ can be defined using only 1 schema LIM.
- A corollary of this remark is that composing several LIM schemata is always equivalent to at most two for functions of our classes.

References

1. V. I. Arnold. *Ordinary Differential Equations.* MIT Press, 1978.
2. E. Asarin and A. Bouajjani. Perturbed Turing machines and hybrid systems. In *Logic in computer science,* pages 269–278, 2001.
3. E. Asarin and O. Maler. Achilles and the tortoise climbing up the arithmetical hierarchy. *Journal of Computer and System Sciences,* 57(3):389–398, dec 1998.
4. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer-Verlag, 1998.
5. O. Bournez. Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy. *Theoretical Computer Science,* 210(1):21–71, 6 1999.
6. O. Bournez. *Complexité algorithmique des systèmes dynamiques continus et hybrides.* PhD thesis, École Normale Supérieure de Lyon, janvier 1999.
7. M. Bowles. United States technological enthusiasm and the british technological skepticism in the age of the analog brain. In *IEEE Annals of the History of Computing,* volume 4, pages 5–15, 1996.

8. M. Campagnolo, C. Moore, and J. F. Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity,* 18(4):977–1000, 2002.
9. M. L. Campagnolo. *Computational complexity of real valued recursive functions and analog circuits.* PhD thesis, Universidade Técnica de Lisboa, 2001.
10. G. Etesi and I. Németi. Non-Turing computations via Malament-Hogarth space-times. *International Journal Theoretical Physics,* 41:341–370, 2002.
11. D. Graça and J. F. Costa. Analog computers and recursive functions over the reals. *Journal of Complexity,* 19:644–664, 2003.
12. A. Grzegorczyk. Computable functionals. *Fundamenta Mathematicae,* 42:168–202, 1955.
13. T. Henzinger and J.-F. Raskin. Robust undecidability of timed and hybrid systems. *Hybrid systems: computation and control; second international workshop, hscc '99, berg en dal, the netherlands, march 29–31, 1999; proceedings,* 1569, 1999.
14. M. L. Hogarth. Does general relativity allow an observer to view an eternity in a finite time? *Foundations of physics letters,* 5:173–181, 1992.
15. D. Lacombe. Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles III. *Comptes rendus de l'Académie des Sciences Paris,* 241:151–153, 1955.
16. L. Lipshitz and L. A. rubel. A differentially algebraic replacement theorem, and analog computability. *Proceedings of the American Mathematical Society,* 99(2):367–372, February 1987.
17. C. Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science,* 162(1):23–44, 5 1996.
18. J. Mycka. Infinite limits and R-recursive functions. *Acta Cybernetica,* 16:83–91, 2003.
19. J. Mycka. μ-recursion and infinite limits. *Theoretical Computer Science,* 302:123–133, 2003.
20. P. Odifreddi. *Classical recursion theory II.* North-Holland, 1999.
21. T. Ord. Hypercomputation: computing more than the Turing machine. Technical report, University of Melbourne, September 2002. available at `http://www.arxiv.org/abs/math.lo/0209332`.
22. P. Orponen. *Algorithms, languages and complexity,* chapter A survey of continuous-time computational theory, pages 209–224. Kluwer Academic Publishers, 1997.
23. M. B. Pour-El. Abstract computability and its relation to the general purpose analog computer (some connections between logic, differential equations and analog computers). *Transactions of the American Mathematical Society,* 199:1–28, 1974.
24. H. Rose. *Subrecursion: Functions and Hierarchies.* Clarendon Press, 1984.
25. C. E. Shannon. Mathematical theory of the differential analyser. *Journal of Mathematics and Physics MIT,* 20:337–354, 1941.
26. H. Siegelmann. *Neural networks and analog computation - beyond the Turing limit.* Birkauser, 1998.
27. W. Thomson. On an instrument for calculating the integral of the product of two given functions. In *Proceedings of the royal society of London,* number 24, pages 266–276, 1876.
28. A. Turing. On computable numbers, with an application to the "Entschei–dungs–problem". In *Proceedings of the london mathematical society,* volume 2, pages 230–265, 1936.
29. K. Weihrauch. *Computable Analysis.* Springer, 2000.
30. Q. Zhou. Subclasses of computable real valued functions. *Lecture Notes in Computer Science,* 1276:156–165, 1997.

Model Checking with Multi-valued Logics

Glenn Bruns and Patrice Godefroid

Bell Laboratories, Lucent Technologies,
{grb,god}@bell-labs.com

Abstract. In multi-valued model checking, a temporal logic formula is interpreted relative to a structure not as a truth value but as a lattice element. In this paper we present new algorithms for multi-valued model checking. We first show how to reduce multi-valued model checking with any distributive DeMorgan lattice to standard, two-valued model checking. We then present a direct, automata-theoretic algorithm for multi-valued model checking with logics as expressive as the modal mu-calculus. As part of showing correctness of the algorithm, we present a new fundamental result about extended alternating automata, a generalization of standard alternating automata.

1 Introduction

In multi-valued model checking, one interprets a temporal logic formula on a multi-valued Kripke structure, which is like a Kripke structure except that an atomic proposition is interpreted at a state as a lattice element, not a truth value. The meaning of a temporal logic formula at a state in such a structure is then also given as a lattice element.

Multi-valued model checking is proving valuable as the basis for a variety of new verification methods. For example, the abstraction method of [4] involves model checking with the lattice L_3 of Figure 1, where 1 represent truth, 0 represents falsity, and 1/2 represents "unknown whether true or false". Model checking with the lattice $L_{2,2}$ can be used to analyze whether conflict will arise when multiple requirements are combined [8,18]. Temporal logic query checking [6,3,9] can be regarded as model checking over lattices in which each element is a set of propositional formulas.

One approach to multi-valued model checking is the *reduction method,* in which a multi-valued model checking problem is reduced to a set of standard, two-valued model checking problems [2,19,18]. For example, in the case of lattice L_3, a model checking problem for a Kripke structure over L_3 can be reduced to two model checking problems for Kripke structures over L_2. Another approach is the *direct method,* in which multi-valued model checking is performed directly using special-purpose algorithms. An advantage of the reduction method is that it can use existing tools, and benefits as these tools are improved. The advantage of the direct approach is that it works in a more "on-demand" manner than the reduction approach (more comparisons are made in Section 6).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 281–293, 2004.

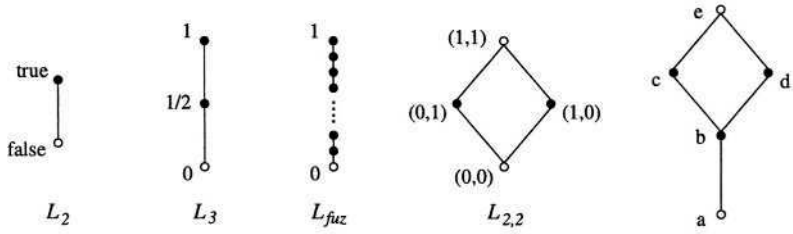


Fig. 1. Some distributive lattices

This paper describes improved reduction and direct methods for multi-valued model checking. A problem with existing reduction methods [2,19] is their limitation to selected sub-classes of DeMorgan lattices. A recent method [17] is more general but also more complicated, involving a step that uses an additional logic. Our method is simple and general. We show that, for a finite distributive lattice, the number of standard model checks required is equal to the number of join-irreducible elements of the lattice in the worst case. From a multi-valued Kripke structure over a finite distributive lattice, we show how a standard Kripke structure can be derived for each join-irreducible element of the lattice, and how the results of model checking on each of these Kripke structures can be combined to give a result for the multi-valued model check. The method yields complexity bounds for the multi-valued model-checking problem for various temporal logics.

Existing work on direct methods is limited in the class of lattices that are handled, or the logic that is supported. In [4] an algorithm is defined for CTL over L_3. In [10] an automata-theoretic algorithm is defined for LTL over finite linear orders. In [7] a BDD-based algorithm is defined for CTL over DeMorgan lattices. Our method is automata-theoretic and handles all DeMorgan lattices and the full modal mu-calculus. To adapt the automata-theoretic method to multi-valued model checking, we use extended alternating automata (EAA) [3], which extend alternating automata (AA). In model checking applications of AA (e.g., [21]), an input tree of the automaton has nodes that are labelled with sets of atomic propositions, and a run of the automaton has no value associated with it. With EAA, the nodes of the input tree are labelled with functions mapping atomic propositions to elements of a lattice, and a run has an associated value. We show how to use EAA for multi-valued model checking, but also prove a fundamental result about EAA that is interesting independently of this application: that the set of values of all the accepting runs of an EAA has a maximal element.

The following section briefly covers some background material. In Section 3, we define our reduction method. In Section 4 we define extended alternating automata, and in Section 5 we show how to directly model check with them. We conclude in Section 6 by comparing the reduction and direct approaches to multi-valued model checking.

2 Background

Lattices and Negation. We take for granted the notion of lattice and complete lattice. We write $x \vee y$ or $\bigvee P$ for join and $x \wedge y$ or $\bigwedge P$ for meet (where P is a set). Every complete lattice has a greatest element, called *top,* and a least element, called *bottom* (and written $\perp$). Every finite lattice is complete. A lattice is *distributive* if $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ for all lattice elements x, y, z.

A *join-irreducible element* x of a distributive lattice L is an element that is not bottom and for which $x = y \vee z$ implies $x = y$ or $x = z$. If L is finite, the join-irreducible elements are easily spotted in the Hasse diagram for L as elements having exactly one lower cover (i.e. one line connected to the element from below). The darkened elements in Figure 1 are the join-irreducible ones. We write $\mathcal{J}(L)$ for the set of all join-irreducible elements of L.

If one orders truth and falsity as shown in lattice L_2 of Figure 1, then conjunction can be interpreted as meet and disjunction as join. In this way conjunction and disjunction can be interpreted over an arbitrary lattice. To interpret negation on lattices, a restricted class of lattices must be used if one hopes to obtain expected properties of negation. *Boolean* lattices support a strong sense of complement. Every element x in such a lattice has a unique complement $\neg x$ such that $x \vee \neg x$ equals the top element of the lattice and $x \wedge \neg x$ equals the bottom element of the lattice. Lattice L_2 of Fig. 1 is boolean. However, there are "few" boolean lattices.

In a *DeMorgan* (or *quasi-boolean*) lattice [1], every element x has a unique complement $\neg x$ such that $\neg\neg x = x$, DeMorgan's laws hold, and $x \leq y$ implies $\neg y \leq \neg x$. DeMorgan lattices can be characterized as lattices with horizontal symmetry [7]. Lattice L_3 of Fig. 1 is DeMorgan, but not boolean. Using DeMorgan complement we get that $\neg 0 = 1$, $\neg 1/2 = 1/2$, and $\neg 1 = 0$

A *Heyting algebra* is a lattice with a bottom element in which every element x has a unique *relative pseudo-complement* $\neg x$ defined as the greatest element y such that $x \wedge y$ equals the lattice's bottom element. In the case of finite lattices, Heyting algebras and distributive lattices are the same thing [13]. The rightmost lattice in Fig. 1 is a Heyting algebra but is not DeMorgan. In this lattice, using relative pseudo-complement as complement, we get $\neg a = e$ and $\neg b = a$. In lattice L_3 we get $\neg 0 = 1$, $\neg 1 / 2 = 0$, and $\neg 1 = 0$. Some DeMorgan lattices are not Heyting algebras.

Reasoning about partial information with three-valued logic based on L_3 is an important application of multi-valued model checking, and since in this application we want to interpret negation in the DeMorgan sense, we adopt DeMorgan lattices for multi-valued model checking.

The Modal Mu-Calculus. The modal mu-calculus [20] is an expressive modal logic that includes as fragments linear-time temporal logic (LTL) and computation-tree logic (CTL) [12]. Without loss of generality, we use a positive form of the modal mu-calculus in which negation applies only to atomic propositions. Formulas have the following abstract syntax, where p ranges over a set P of atomic propositions and X ranges over a set *Var* of fixed-point variables:

$$\phi ::= p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \Box\phi \mid \Diamond\phi \mid X \mid \nu X.\phi \mid \mu X.\phi$$

In fixed-point formulas $\nu X.\phi$ and $\mu X.\phi$ the operators ν and μ bind free occurrences of X in ϕ. We call this logic μL.

A *Kripke structure* $M = (S, s_0, \Theta, \mathcal{R})$ consists of a set S of states, an initial state s_0 in S, a mapping Θ from states to subsets of P, and a transition relation $\mathcal{R} \subseteq S \times S$, assumed to be total. We say M is *finite* if it has finitely many states. We write $s \to s'$ if $(s, s') \in \mathcal{R}$ and write $succ_{\mathcal{R}}(s)$ for the set $\{s' \in S \mid s \to s'\}$. For a finite subset D of $\mathbb{N}$, we say M *has degrees in* D if $|succ_{\mathcal{R}}(s)| \in D$ for all states s of S.

A Kripke structure $M = (S, s_0, \Theta, \mathcal{R})$ over a lattice L differs from a standard Kripke structure in that now Θ maps a state to a mapping from propositions to elements of L. We write $P \to L$ for the set of all mappings from P to L.

A *valuation* $\mathcal{V}$ over a lattice L maps a variable to a mapping from states to elements of L. We write $()$ for the valuation such that $()(X)(s) = \bot$ for all X and s (it is required here that L has a bottom element), and write $\mathcal{V}[X := f]$ for the valuation that is like $\mathcal{V}$ except that it maps X to f.

We define the meaning $\|M, \phi\|_{\mathcal{V}}$ of a μL formula relative to a Kripke structure $M = (S, s_0, \Theta, \mathcal{R})$ over lattice L as a mapping from S to L. In the following definition the function $f : (S \to L) \to (S \to L)$ is defined by $f(g) = \|M, \phi\|_{\mathcal{V}[X:=g]}$, and νf and μf stand for the greatest and least fixed-points of f. We know f has greatest and least fixed-points by the Knaster-Tarski fixpoint theorem [23] because the functions in $S \to L$, under pointwise ordering, form a complete lattice, and function f preserves this ordering.

Definition 1. *The interpretation* $\|M, \phi\|_{\mathcal{V}}$ *of a* μL *formula relative to Kripke structure* $M = (S, s_0, \Theta, \mathcal{R})$ *and valuation* $\mathcal{V}$ *over complete DeMorgan lattice L is defined as follows:*

$$\begin{aligned}
\|M, p\|_{\mathcal{V}} &= \lambda s.\Theta(s)(p) & \|M, \phi_1 \wedge \phi_2\|_{\mathcal{V}} &= \lambda s.\|M, \phi_1\|_{\mathcal{V}}(s) \wedge \|M, \phi_2\|_{\mathcal{V}}(s) \\
\|M, \neg p\|_{\mathcal{V}} &= \lambda s.\neg\Theta(s)(p) & \|M, \phi_1 \vee \phi_2\|_{\mathcal{V}} &= \lambda s.\|M, \phi_1\|_{\mathcal{V}}(s) \vee \|M, \phi_2\|_{\mathcal{V}}(s) \\
\|M, \nu X.\phi\|_{\mathcal{V}} &= \nu f & \|M, \Box\phi\|_{\mathcal{V}} &= \lambda s.\bigwedge\{\|M, \phi\|_{\mathcal{V}}(s') \mid s \to s'\} \\
\|M, \mu X.\phi\|_{\mathcal{V}} &= \mu f & \|M, \Diamond\phi\|_{\mathcal{V}} &= \lambda s.\bigvee\{\|M, \phi\|_{\mathcal{V}}(s') \mid s \to s'\} \\
\|M, X\|_{\mathcal{V}} &= \mathcal{V}(X)
\end{aligned}$$

If ϕ is a closed formula then we write $[(M, s), \phi]$ for the value $\|M, \phi\|_{()}(s)$ of formula ϕ at state s of Kripke structure M. Given ϕ, (M, s), and L, computing $[(M, s), \phi]$ is called the *multi-valued model-checking problem.* If M is a Kripke structure over lattice L_2, then we write $(M, s) \models \phi$ if $[(M, s), \phi] = true$.

Proposition 1. *The* μL *semantics of Def. 1 collapses to the standard two-valued semantics of* μL *when lattice L is* L_2 *of Fig. 1.*

3 Reduction to 2-Valued Model Checking

In this section we show how multi-valued model checking of a μL formula ϕ relative to a Kripke structure M over a finite distributive lattice L can be performed by model checking ϕ relative to a set of standard Kripke structures.

A key part of our approach is the treatment of negation. We transform ϕ to a formula ϕ' containing no negation symbols. Each negated proposition $\neg p$ in ϕ is replaced by $\tilde{p}$, where $\tilde{p}$ is a fresh proposition not already appearing in ϕ. Correspondingly, M is transformed to M' by extending the proposition valuation Θ of M to Θ', where $\Theta'(s)(\tilde{p}) = \neg\Theta(s)(p)$. Then $[(M, s), \phi] = [(M', s), \phi']$ for all states s of M. In the rest of this section we consider only formulas of μL not containing the negation symbol. Note that our step of eliminating negation symbols requires a negation operation on the underlying lattice.

3.1 Reduction Method

We now describe how to derive a standard Kripke structure M_x from a Kripke structure M over lattice L. If M is defined to be $(S, s_0, \Theta, \mathcal{R})$, and x is an element of L, then M_x is defined to be $(S, s_0, \Theta_x, \mathcal{R})$, where

$$\Theta_x(s)(p) = \Theta(s)(p) \geq x$$

M_x differs from M only in its treatment of atomic propositions. In M_x, propositions with value x or greater are regarded as true, and all others as false. Thus, if $x \geq x'$, we expect a formula that holds in M_x to also hold in $M_{x'}$.

Proposition 2. *Let M be a Kripke structure over a finite distributive lattice L, with s in M and x, x' in L. Then $((M_x, s) \models \phi$ and $x \geq x') \Rightarrow (M_{x'}, s) \models \phi$*

The value of a formula relative a Kripke structure over a lattice L can be determined by checking the standard Kripke structures derived from the join-irreducible elements of L.

Lemma 1. *Let M be a Kripke structure over a finite distributive lattice L, with s in M and x in $\mathcal{J}(L)$. Then $(M_x, s) \models \phi \Leftrightarrow x \leq [(M, s), \phi]$.*

From this lemma our main theorem follows using Birkhoff's representation theorem for finite distributive lattices, which states that every element a of such a lattice can be represented as the join of all the join-irreducible elements less than or equal to a in the lattice.

Theorem 1. *Let M be a Kripke structure over a finite distributive lattice L, with s in M. Then $[(M, s), \phi] = \bigvee\{x \in \mathcal{J}(L) \mid (M_x, s) \models \phi\}$.*

For example, consider the model checking of a formula ϕ relative to a structure M over lattice L_3 of Fig. 1. The join-irreducible elements of L_3 are 1/2 and 1. Intuitively, the model M_1 represents a pessimistic view in which 1/2 is taken as false, while $M_{1/2}$ represents an optimistic view in which 1/2 is taken as true. The algorithm first checks whether ϕ holds in M_1. If so, the result is $\bigvee\{1/2, 1\}$, or 1. If not, it checks whether ϕ holds of model $M_{1/2}$. If so, the result is $\bigvee\{1/2\}$, or 1/2. Otherwise the result is $\bigvee\emptyset$, or 0.

Since two-valued model checking is a special case of multi-valued model checking, our reduction immediately gives the following complexity bounds for the multi-valued model-checking problem.

Theorem 2. *Let L be a finite distributive DeMorgan lattice with n join-irreducible elements, and let TL denote μL or any of its fragments. Then the multi-valued model-checking problem for TL with respect to L can be solved in time linear in n. Moreover, the complexity of multi-valued model checking for TL has the same time and space complexity, both in the size of the Kripke structure and of the formula, as traditional two-valued model checking for TL.*

The linear complexity in the number of join-irreducible elements can be improved for some classes of lattices. For example, when the join-irreducible elements of a lattice L are linearly ordered, a binary search (i.e., checking first the join-irreducible element in the middle of the lattice, then the join-irreducible element in the middle of the upper or lower half, etc.) can be performed instead of a linear search, providing a decision procedure for the multi-valued model-checking problem for L with a worst-case time complexity of $O(log(n))$ instead of $O(n)$.

3.2 Multi-valued Transitions

In Kripke structures with multi-valued transitions, transitions are represented by a function R that maps pairs of states to lattice values. The μL semantics (see Section 2) changes only for the modal operators, as follows:

$$\|M, \Box\, \phi\|_{\mathcal{V}} = \lambda s. \bigwedge \{\neg R(s, s') \vee \|M, \phi\|_{\mathcal{V}}(s') \mid \text{all } s'\}$$
$$\|M, \Diamond\, \phi\|_{\mathcal{V}} = \lambda s. \bigvee \{R(s, s') \wedge \|M, \phi\|_{\mathcal{V}}(s') \mid \text{all } s'\}$$

A Kripke structure with multi-valued transitions can be transformed to a structure without multi-valued transitions using the idea described in Definitions 16 and 17 of [16]. However, this transformation may in the worst case involve a blow-up of size $|L|$. Therefore we extend our reduction method to handle multi-valued transitions directly, with no blow-up in $|L|$. The extended method works in two steps. First, as before, from the original Kripke structure M over a lattice L, we obtain a set $\{M_x \mid x \in \mathcal{J}(L)\}$ of structures. However, each structure M_x now has two transition relations: $\mathcal{R}^+$ and $\mathcal{R}^-$. In the second step, each M_x is translated to a standard Kripke structure M'_x having only a single transition relation.

We now briefly cover the details. Suppose $M = (S, s_0, \Theta, R)$ is a Kripke structure over a finite distributive lattice L, where $R : S \times S \to L$ is the multi-valued transition function. Given a join-irreducible element x of L, we define M_x as before, except that now M_x has the form $(S, s_0, \Theta_x, \mathcal{R}_x^+, \mathcal{R}_x^-)$, where we define $\mathcal{R}_x^+(s, s') = R(s, s') \geq x$ and define $\mathcal{R}_x^-(s, s') = \neg(\neg(R(s, s')) \geq x)$. In interpreting a formula over such a structure, we modify the μL semantics as follows:

$$\|M_x, \Box\, \phi\|_{\mathcal{V}} = \lambda s. \bigwedge \{\neg \mathcal{R}^-(s, s') \vee \|M, \phi\|_{\mathcal{V}}(s') \mid \text{all } s'\}$$
$$\|M_x, \Diamond\, \phi\|_{\mathcal{V}} = \lambda s. \bigvee \{\mathcal{R}^+(s, s') \wedge \|M, \phi\|_{\mathcal{V}}(s') \mid \text{all } s'\}$$

Our reduction lemma (Lemma 1) also holds for this extended reduction.

Lemma 2. *Let M be a Kripke structure with multi-valued transitions over a finite distributive lattice L, with s in S, and x in $\mathcal{J}(L)$. Then, letting M_x be the result of the extended reduction, $(M_x, s) \models \phi \Leftrightarrow x \leq [(M, s), \phi]$.*

In the second step, we translate the structure $M_x = (S, s_0, \Theta, \mathcal{R}^+, \mathcal{R}^-)$ to a standard Kripke structure $M'_x = (S', s'_0, \Theta', \mathcal{R}')$. The set of propositions over which Θ' is defined is $P \cup \{p^+\}$, and

$$S' = \{(s, sign) \mid s \in S, sign \in \{+, -\}\}$$
$$s'_0 = (s_0, +)$$
$$\Theta'(s, sign)(p) = \text{if } p \equiv p^+ \text{ then } (sign = +) \text{ else } \Theta(s, p)$$
$$\mathcal{R}'((s, sign), (s', sign')) = (s, s') \in \mathcal{R}^{sign'}(s, s')$$

For every state s in M_x there are states $(s, +)$ and $(s, -)$ in M'_x. Moreover, every pair $(s, +)$, $(s, -)$ of states in M'_x is strongly bisimilar. Since strong bisimulation preserves μL formulas [22], we have that $(s, +)$ satisfies ϕ iff $(s, -)$ does.

We also define a translation T that maps formulas of μL to formulas of μL. The translation maps all operators $\oplus$ homomorphically (i.e., $T(\phi_1 \oplus \phi_2) = T(\phi_1) \oplus T(\phi_2)$), except the modal operators. In these cases we have $T(\Box \phi) = \Box(p^+ \vee T(\phi))$ and $T(\Diamond \phi) = \Diamond(p^+ \wedge T(\phi))$. The correctness condition for the second step is that a formula holds of M_x iff the translated formula holds of M'_x.

Proposition 3. *Let M_x be a Kripke structure with two transition relations, M'_x be the standard Kripke structure obtained by translation from M_x, s be a state of M_x, and ϕ be a formula of μL. Then $(M_x, s) \models \phi \Leftrightarrow (M'_x, (s, +)) \models T(\phi)$.*

3.3 Related Work

In [2] a reduction is given for three-valued model checking. In [19], reductions are given for total orders, binary products of total orders, and the lattice $2 \times 2 + 2$, which can be obtained from the right-most lattice of Fig. 1 by adding a new top element f above element a.

A method [17] with the same generality as ours was discovered independently (see [5]). In the method of [17] each μL formula is translated first to a set of formulas in a logic designed specifically for the reduction, then each formula in this set is translated to a μL formula. Our approach uses fewer steps, no additional logic, and has simpler proofs (due to the use of Birkhoff's theorem).

In [14], Fitting shows how a many-valued Kripke structure can be transformed to a "multiple-expert" structure, that includes a set of *experts* and a binary *dominates* relation over experts. Although the core idea of our method comes from a construction in the proof of Prop. 5.1 of [14], our work differs in several ways. We reduce to standard Kripke structures rather than multi-expert models, we use μL rather than propositional modal logic, we use join-irreducible elements rather than proper prime filters, and most importantly, we treat negation parametrically rather than as relative pseudo-complement. The advantage

of our approach to negation is generality; the disadvantage is that it increases the size of the model's propositional valuation.

[18] concerns AC-lattices, which are pairs of graph-isomorphic lattices in which the order relation of one is the inverse of the other. Negation in an AC-lattice is captured as two maps, each mapping an element of one lattice to the isomorphic image in the other. AC-lattices can be used for the analysis of conflict between multiple requirements. A notion of expert similar to Fitting's is used. It is shown, for finite models, that for each of the two "modes" captured by the two lattices in an AC-lattice, the set of views for which a modal mu-calculus formula holds is equal to the set obtained by an interpretation of the formula as a view set. The result differs from ours in that it is based on AC-lattices, in its treatment of negation, and in that it relates view sets rather than lattice elements directly.

4 Extended Alternating Automata

The idea behind alternating automata is to describe successor states through boolean expressions built up from states and truth values using conjunction and disjunction. EAA generalize this idea by allowing expressions built up from states and lattice elements using meet and join. A run of an EAA on an input tree is itself a tree, as in alternating automata. However, each node of the run is now labelled with a lattice element.

With alternating automata, one is interested in whether an accepting run exists on an input tree. With EAA, each accepting run has a value (the value at its root), and one is interested in the set of values of all accepting runs. A fundamental question for EAA, and one that is key for the use of EAA in model checking, is whether this set of values has a maximum element. We show below that this is indeed the case.

Definitions. Formally, a *tree* τ is a subset of $\mathbb{N}^*$ such that if $x \cdot c \in \tau$ then $x \in \tau$ and $x \cdot c' \in \tau$ for all $1 \leq c' < c$. The elements of τ are called its *nodes,* with ϵ called the *root.* Given a node x of τ, values of the form $x \cdot i$ in τ are called the *children* or *successors* of x. The number of successors of x is called the *degree* of x. A node with no successors is called a *leaf.* Given a set $D \subset \mathbb{N}$, a *D-tree* is a tree in which the degree of every node is in D. A *Σ-labeled* tree is a pair (τ, T) in which τ is a tree and $T : \mathbb{N}^* \to \Sigma$ is a labeling function.

Let $L = (B, \wedge, \vee)$ be a lattice, and let $\mathcal{B}^+(X)$ stand for the set of terms built from elements in a set X using $\wedge$ and $\vee$. A *tree EAA over L* is a tuple $A = (\Sigma, D, S, s_0, \rho, F)$, where Σ is a nonempty finite alphabet, S is a nonempty finite set of states, $s_0 \in S$ is the initial state, F is an acceptance condition, $D \subset \mathbb{N}$ is a finite set of arities, and $\rho : S \times \Sigma \times D \to \mathcal{B}^+((\mathbb{N} \times S) \cup B)$ is a transition function, where $\rho(s, a, k) \in \mathcal{B}^+((\{1, \ldots, k\} \times S) \cup B)$ is defined for each s in S, a in Σ, and k in D. Various types of acceptance conditions F can be used with EAA, just as in alternating automata, and are discussed below.

A *v-run* of a tree EAA A on a Σ-labeled leafless D-tree (τ, T) is an $\mathbb{N}^* \times S \times B$-labeled tree (τ_σ, T_σ). A node in τ_σ labeled by (x, s, v) describes a copy

of automaton A that reads the node x of τ in the state s of A and has value $v \in B$ associated with it. Formally, a v-run (τ_σ, T_σ) is an $\mathbb{N}^* \times S \times B$-labeled tree, defined as follows.

- $T_\sigma(\epsilon) = (\epsilon, s_0, v)$
- Let $y \in \tau_\sigma$, $T_\sigma(y) = (x, s, v')$, $arity(x) = k$, and $\rho(s, T(x), k) = \theta$. Then there is a (possibly empty) set $Q = \{(c_1, s_1, v_1), \ldots, (c_n, s_n, v_n)\} \subseteq \{1, \ldots, k\} \times S \times B$ such that
  - for all $1 \leq i, j \leq n$, $c_i = c_j$ and $s_i = s_j$ implies $v_i = v_j$,
  - $Eval(Q, \theta) = v'$, and
  - for all $1 \leq i \leq n$, we have $y \cdot i \in \tau_\sigma$ and $T_\sigma(y \cdot i) = (x \cdot c_i, s_i, v_i)$

$Eval(Q, \theta)$ denotes the value of the expression θ obtained by replacing each term (c_i, s_i) in θ by v_i if $(c_i, s_i, v_i) \in Q$ or by $\bot$ otherwise.

A v-run σ is *accepting* if (1) the value associated with each node of the run is not $\bot$ and (2) all infinite branches of the run satisfy the acceptance condition F. As with traditional alternating automata, various types of acceptance conditions can be used. For instance, a path w satisfies a *parity acceptance condition* $F = \{F_1, F_2, \ldots, F_n\}$ with $F_1 \subseteq F_2 \subseteq \ldots \subseteq F_n$ if the minimal index i for which some state s in F_i appears infinitely often along w is even. Note that an accepting run can have finite branches: if, for some $y \in \tau_\sigma$, $T_\sigma(y) = (x, s, v)$ and $\rho(s, T(x), arity(x)) = v$ with v in B and $v \neq \bot$, then y does not need to have any successor.

A tree EAA A accepts a Σ-labeled leafless D-tree (τ, T) with value v if there exists an accepting v-run of A on that tree. We define the language $\mathcal{L}_v(A)$ as follows (for $v \neq \bot$): $\mathcal{L}_v(A) = \{(\tau, T) \mid A \text{ accepts } (\tau, T) \text{ with value } v\}$. For convenience, we define $\mathcal{L}_\bot(A)$ as $\{(\tau, T) \mid A \text{ has no accepting run on } (\tau, T)\}$. When D is a singleton, A runs over trees with a fixed branching degree. In particular, a *word EAA* is simply a tree EAA in which $D = \{1\}$.

Existence of Maximum Value. We now establish a new, fundamental property of EAA: for any EAA and any input tree, there always exists a maximum value v of L for which the EAA has an accepting v-run on the input tree. Note that this property is non-trivial since it is not generally true that, if an EAA has an accepting v_1-run and an accepting v_2-run on an input tree, then the EAA has an accepting $(v_1 \vee v_2)$-run on this input tree.

Theorem 3 (Maximum-value theorem). *Let A be a (finite) tree EAA over a lattice L, and let (τ, T) be a Σ-labeled leafless D-tree. Then the subset $\{v \mid (\tau, T) \in \mathcal{L}_v(A)\}$ of L has a maximum value, which we denote by $Max(A, (\tau, T))$.*

We will write simply $Max(A)$ when A is a word EAA on a 1-letter alphabet.

5 Model Checking with EAA

Our model-checking procedure for multi-valued logics using EAA generalizes the automata-theoretic approach to 2-valued model checking with AAs [21]. Our procedure computes the value $[(M, s), \phi]$ defined by a μL formula ϕ evaluated

in state s of a Kripke structure M over a DeMorgan lattice L. (Multi-valued transitions in M can be transformed first as discussed in Section 3.2.) In the first step of the procedure we translate ϕ to an EAA A_ϕ. Then we build a product automaton from A_ϕ and M in such a way that the maximum value that labels an accepting run of the product automaton is $[(M, s), \phi]$. We now present these steps in detail.

We begin with a translation of μL formulas to EAA. The translation is similar to the translation from μL to parity alternating automata given in [21] except for the case of atomic propositions, which are mapped to lattice elements in our context. The property we want of the translation is that the value of the maximum accepting run of the EAA for formula ϕ and an input tree (τ, T) agrees with the value $[(\tau, T), \phi]$ defined by the semantics of μL (with (τ, T) viewed as a Kripke structure over L).

Theorem 4. *Let ϕ be a closed μL formula and L be a DeMorgan lattice. Then a parity EAA $A_{D,\phi}$ for ϕ can be constructed in linear time such that $[((\tau, T), \epsilon), \phi)] = Max(A_{D,\phi}, (\tau, T))$ for every leafless D-tree (τ, T) on L.*

In the next step of the procedure, we compute the product of a Kripke structure and an EAA representing a μL formula. The product construction defined here is again nearly identical to that given for alternating automata in [21].

Definition 2. *Let ϕ be a closed μL formula, L be a DeMorgan lattice, $M = (S, s_0, \Theta, \mathcal{R})$ be a finite Kripke structure over L, with degrees in D, and $A_{D,\phi} = (P \to L, D, Q_\phi, q_0, \rho_\phi, F)$ be a parity EAA representing ϕ. Then the* product automaton *$A_{M,\phi} = (\{a\}, S \times Q_\phi, (s_0, q_0), \rho, F)$ of M and $A_{D,\phi}$ is a parity word EAA over a 1-letter alphabet with at most $O(|S| \cdot |Q_\phi|)$ states, where ρ and F are defined as follows:*

- *For all $q \in Q_\phi$, $s \in S$, if $succ_\mathcal{R}(s) = (s_1, \ldots, s_n)$ and $\rho_\phi(q, \Theta(s), n) = \theta$, then $\rho((s, q), a) = \theta'$ where θ' is obtained from θ by replacing each atom (c, q') in θ by (s_c, q').*
- *If $F_\phi = \{F_1, F_2, \ldots, F_m\}$ is a parity acceptance condition, then so is $F = \{(S \times F_1), (S \times F_2), \ldots, (S \times F_m)\}$.*

The product automaton $A_{M,\phi}$ is used to prove the following.

Theorem 5. *Let ϕ be a closed μL formula, M be a finite Kripke structure over a DeMorgan lattice L, and s be a state of M. Then there exists a parity word EAA $A_{M,\phi}$ over a 1-letter alphabet such that $[(M, s), \phi)] = Max(A_{M,\phi})$.*

In the final step of the procedure, we compute the value $Max(A_{M,\phi})$ of the product EAA.

Theorem 6. *Given a parity word EAA $A_{M,\phi}$ over L with a 1-letter alphabet, computing $Max(A_{M,\phi})$ has the same complexity as checking whether the language accepted by a parity word AA with a 1-letter alphabet is nonempty, i.e., can be done in nondeterministic polynomial time.*

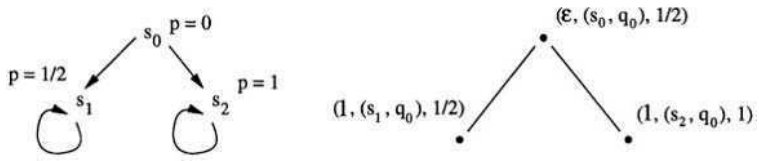


Fig. 2. Example Kripke structure M and accepting run

Algorithms for computing $Max(A)$ of a word EAA A over a 1-letter alphabet are similar to algorithms for checking emptiness of AAs over a 1-letter alphabet except that the algorithms dealing with EAA propagates values in L instead of values in $\{true, false\}$. The number of iterations for each state can be bounded by $O(|h(L)|)$ where $h(L)$ is the height of L (e.g., [15]). The traditional μL model-checking problem is in NP∩co-NP, and this upper bound carries over to the multi-valued case. However, computing $Max(A_{M,\phi})$ can be done more efficiently for some subclasses of μL. For instance, the EAA for a CTL formula ϕ is *weak* [21], and computing the value $Max(A_{M,\phi})$ of the product of a weak EAA with a Kripke structure M can be done in time linear in $|M|$ and $|\phi|$ [3].

Example 1. Consider the μL formula $\mu X.p \vee \Box X$, which is equivalent to the CTL formula AFp. By translating this formula into an EAA satisfying Theorem 4, we obtain a tree EAA with a single state q_0, an acceptance condition $F = \emptyset$ and the following transition function: $\rho(q_0, \sigma, k) = \sigma(p) \vee \bigwedge_{c=1}^{k}(c, q_0)$. We next take the product of this automaton with the Kripke structure M over L_3 shown on the left of Figure 2. The figure shows the value of the atomic proposition p at each state. Using the product construction of Definition 2, we obtain a (weak) word EAA over a 1-letter alphabet with no accepting states and the following transition function: $\rho((s_0, q_0), a, 1) = 0 \vee ((s_1, q_0) \wedge (s_2, q_0))$, $\rho((s_1, q_0), a, 1) = 1/2 \vee (s_1, q_0)$, and $\rho((s_2, q_0), a, 1) = 1 \vee (s_2, q_0)$. This EAA has the accepting 1/2-run shown on the right in Figure 2. The value 1/2 is the greatest value v for which there is an accepting v-run, so by Theorem 5, we have $[(M, s_0), \mu X.p \vee \Box X] = 1/2$.

6 Discussion

As mentioned in the introduction, an advantage of the reduction approach to multi-valued model checking is that it can be implemented using existing model checkers. On the other hand, the direct approach can work in a more "on-the-fly" fashion, computing whatever information is necessary to solve the problem at hand on a demand-driven basis. Indeed, in the reduction approach, only the lattice and Kripke structure are used in building the two-valued Kripke structures, each of which can then be model checked possibly on-the-fly, thus using the formula to guide the verification needs. In contrast, the direct approach can make use of all three inputs together to further limit computational resources. For instance, consider a lattice of n incomparable elements plus a top and bottom element, and suppose the formula we wish to model check is simply the atomic

proposition p. In the reduction approach we must then perform n model checks. In the direct approach we will perform a single model check that examines only the initial state of the multi-valued Kripke structure and reads only the value of p, which requires reading only $log(n)$ bits.

Note that, in a finite-state Kripke structure with finitely-many atomic propositions, at most finitely-many lattice elements will appear. From these, by closing under meet and join, one obtains a finite sublattice of the original lattice. This finite sublattice can be used in place of the original one for multi-valued model checking, with either approach, and thus the size of the original lattice does not matter (and could even be infinite). Finally note that, unlike the reduction approach, the direct approach does not require the lattice to be distributive.

Acknowledgements. We thank the anonymous reviewers for their helpful comments. This work was funded in part by NSF CCR-0341658.

References

1. L. Bolc and P. Borowik. *Many-Valued Logics.* Springer Verlag, 1992.
2. G. Bruns and P. Godefroid. Generalized Model Checking: Reasoning about Partial State Spaces. In *Proc. of CONCUR 2000, LNCS 1877.* Springer-Verlag, 2000.
3. G. Bruns and P. Godefroid. Temporal Logic Query Checking. In *Proc. of LICS '01,* pages 409–417. IEEE, 2001.
4. G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Proc. of CAV '99, LNCS 1633.* Springer-Verlag, 1999.
5. G. Bruns and P. Godefroid. Model checking with multi-valued logics. Technical Report BL03.00018, Bell Labs, Lucent Technologies, May 2003.
6. W. Chan. Temporal-logic queries. In *Proc. of CAV 2000, LNCS 1855,* pages 450–463. Springer-Verlag, 2000.
7. M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. Multi-valued symbolic model checking. Tech. Report 448, Comp. Sys. Res. Group, Univ. of Toronto, 2001.
8. M. Chechik and W. Easterbrook. A framework for multi-valued reasoning over inconsistent viewpoints. In *Proc. of ICSE '01,* 2001.
9. M. Chechik, W. Easterbrook, and A. Gurfinkel. Model exploration with temporal logic query checking. In *Proc. of FSE '02,* ACM, 2002.
10. M. Chechik, B. Devereux, and A. Gurfinkel. Model-checking infinite state-space systems with fine-grained abstractions using SPIN. In *Proc. of SPIN Workshop on Model-Checking Software,* 2001.
11. B.A. Davey and H.A. Priestly. *Introduction to Lattices and Order.* Cambridge University Press, 1990.
12. E. A. Emerson. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science,* pages 995–1072. Elsevier, 1990.
13. M. Fitting. Many-valued modal logics I. *Fund. Informaticae,* 15:235–254, 1992.
14. M. Fitting. Many-valued modal logics II. *Fund. Informaticae,* 17:55–73, 1992.
15. Ch. Fecht and H. Seidl. A Faster Solver for General Systems of Equations. *Sci. Comp. Programming,* 35(2): 137–161, 1999.
16. P. Godefroid and R. Jagadeesan. On the Expressiveness of 3-Valued Models. In *Proc. of VMCAI 2003, LNCS 2575,* pages 206–222. Springer-Verlag, 2003.

17. A. Gurfinkel and M. Chechik. Multi-valued model checking via classical model checking. In *Proc. of CONCUR 2003, LNCS 2761*. Springer-Verlag, 2003.
18. M. Huth and S. Pradhan. Lifting assertion and consistency checkers from single to multiple viewpoints. Technical report 2002/11, Dept. of Computing, Imperial College, London, 2002.
19. B. Konikowska and W. Penczek. Reducing model checking from multi-valued CTL* to CTL*. In *Proc. of CONCUR '02, LNCS 2421*. Springer-Verlag, 2002.
20. D. Kozen. Results on the Propositional Mu-Calculus. *Theoretical Computer Science,* 27:333–354, 1983.
21. O. Kupferman, M. Y. Vardi, and P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. *JACM,* 47(2):312–360, March 2000.
22. C. Stirling. Modal and temporal logics for processes. Notes for Summer School in Logic Methods in Concurrency, C.S. Dept., Åarhus University, Denmark, 1993.
23. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. of Maths,* 5:285–309, 1955.

The Complexity of Partition Functions

Andrei Bulatov[1] and Martin Grohe[2]

[1] Computing Laboratory, University of Oxford, Oxford, UK
Andrei.Bulatov@comlab.ox.ac.uk
[2] Institut für Informatik, Humboldt-Universität, Berlin, Germany
grohe@informatik.hu-berlin.de

Abstract. We give a complexity theoretic classification of the counting versions of so-called H-colouring problems for graphs H that may have multiple edges between the same pair of vertices. More generally, we study the problem of computing a weighted sum of homomorphisms to a weighted graph H.
The problem has two interesting alternative formulations: First, it is equivalent to computing the partition function of a spin system as studied in statistical physics. And second, it is equivalent to counting the solutions to a constraint satisfaction problem whose constraint language consists of two equivalence relations.
In a nutshell, our result says that the problem is in polynomial time if the adjacency matrix of H has row rank 1, and #P-complete otherwise.

1 Introduction

This paper has two different motivations: The first is concerned with constraint satisfaction problems, the second with "spin-systems" as studied in statistical physics. A known link between the two are so-called H-colouring problems. Our main result is a complete complexity theoretic classification of the problem of counting the number of solutions of an H-colouring problem for an undirected graph H which may have multiple edges, and actually of a natural generalisation of this problem to weighted graphs H. Translated to the world of constraint satisfaction problems, this yields a classification of the problem of counting the solutions to constraint satisfaction problems for two equivalence relations. Translated to the world of statistical physics, it gives a classification of the problem of computing the partition function of a spin system.

Let us describe our result from each of the different perspectives: Let H be a graph, possibly with multiple edges between the same pair of vertices. An *H-colouring* of a graph G is a homomorphism from G to H. Both the decision problem, asking whether a given graph has an H-colouring, and the problem of counting the H-colourings of a given graph, have received considerable attention [6,7,10,12,13]. Here we are interested in the counting problem. Dyer and Greenhill [6] gave a complete complexity theoretic classification of the counting problem for undirected graphs H without multiple edges; they showed that the problem is in polynomial time if each connected component of H is complete bipartite without any loops or is complete with all loops present, and #P-complete

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 294–306, 2004.

otherwise. Here we are interested in counting H-colourings for graphs H that may have multiple edges. Note that, as opposed to the decision problem, multiple edges do make a difference for the counting problem. Let H be a graph with vertex set $\{1, \ldots, k\}$. H is best described in terms of its adjacency matrix $A = (A_{ij})$, where A_{ij} is the number of edges between vertices i and j. Given a graph $G = (V, E)$, we want to compute the number of homomorphisms from G to H. Observe that this number is

$$Z_A(G) = \sum_{\sigma: V \to \{1,\ldots,k\}} \prod_{e=\{u,v\} \in E} A_{\sigma(u)\sigma(v)}. \tag{1}$$

Borrowing from the physics terminology, we call Z_A the *partition function* of A (or H). We denote the problem of computing $Z_A(G)$ for a given graph G by EVAL(A). Of course if we define Z_A as in (1), the problem is not only meaningful for matrices A that are adjacency matrices of graphs, but for arbitrary square matrices A. We may view such matrices as adjacency matrices of weighted graphs. We call a symmetric matrix A *connected (bipartite)* if the corresponding graph is connected (*bipartite,* respectively).

We prove the following classification result:

Theorem 1. *Let A be a symmetric matrix with non-negative real entries.*

1. *If A is connected and not bipartite, then* EVAL(A) *is in polynomial time if the row rank of A is at most* 1; *otherwise* EVAL(A) *is #P-complete.*
2. *If A is connected and bipartite, then* EVAL(A) *is in polynomial time if the row rank of A is at most* 2; *otherwise* EVAL(A) *is #P-complete.*
3. *If A is not connected, then* EVAL(A) *is in polynomial time if each of its connected components satisfies the corresponding condition stated in (1) or (2); otherwise* EVAL(A) *is #P-complete.*

Note that this generalises Dyer and Greenhill's [6] classification result for graphs without multiple edges, whose adjacency matrices are symmetric 0-1 matrices.

Our proof builds on interpolation techniques similar to those used by Dyer and Greenhill, recent results on counting the number of solutions to constraint satisfaction problems due to Dalmau and the first author [1], and a considerable amount of polynomial arithmetic. Even though we present the proof in the language of constraint satisfaction problems here, in finding the proof it has been very useful to jump back and forth between the H-colouring and constraint satisfaction perspective. The complete proof can be found in [4].

Let us now explain the result for constraint satisfaction problems. A *constraint language* Γ on a finite *domain* D is a set of relations on D. An instance of the problem CSP(Γ) is a triple $(V, D, \mathcal{C})$ consisting of a set V of *variables,* the domain D, and a set $\mathcal{C}$ of *constraints* (s, ρ), where, for some $r \geq 1$, $s \in V^r$ and ρ is an r-ary relation in Γ. A *solution* is a mapping $\sigma : V \to D$ such that for each constraint $((v_1, \ldots, v_r), \rho) \in \mathcal{C}$ we have $(\sigma(v_1), \ldots, \sigma(v_r)) \in \rho$. There has been considerable interest in the complexity of constraint satisfaction problems [17,

15,8,2,3], which has mainly been driven by Feder and Vardi's [8] *dichotomy question,* asking whether for all languages Γ the problem $\mathrm{CSP}(\Gamma)$ is either solvable in polynomial time or NP-complete. A similar dichotomy question can be asked for the problem $\#\mathrm{CSP}(\Gamma)$ of counting the solutions for a given instance [5,1].

We consider constraint languages Γ consisting of two equivalence relations α, β. Suppose that α has k equivalence classes and β has ℓ equivalence classes. Then Γ can be described by a $(k \times \ell)$-matrix $B = (B_{ij})$, where B_{ij} is the number of elements in the intersection of the ith class of α and the jth class of β. We show that, provided that the matrix is "indecomposable" (in a sense made precise in Section 2.1), the problem $\#\mathrm{CSP}(\Gamma)$ is in polynomial time if the row rank of B is 1 and #*P*-complete otherwise. In [1], it has been shown that if $\#\mathrm{CSP}(\Gamma)$ is in polynomial time, then Γ has a so-called *Mal'tsev polymorphism.* The result of this paper provides a further necessary condition for Γ to give to a counting problem solvable in polynomial time.

There is also a straightforward extension to "decomposable" matrices. We can generalise the result to *weighted CSP,* where each domain element d carries a non-negative real weight $\omega(d)$. The weight of a solution $\sigma : V \to D$ is defined to be the product $\prod_{v \in V} \omega(\sigma(v))$, and the goal is to compute the weighted sum over all solutions. As an important intermediate step, we even prove our classification result for weights that are polynomials with integer coefficients.

Let us finally explain the connection with statistical physics. Statistical physics explaines properties of substances, such as gases, liquids or crystals, using probability distributions on certain states of the substance. In one of the standard models, a substance is considered as a conglomeration of particles (atoms) viewed as a graph $G = (V,E)$, called also a *lattice,* in which adjacent vertices represent particles interacting in a non-negligible way. Every particle may have one of k *spins;* the interaction between neighbouring particles can be described by a *spin system,* which is just a $k \times k$-matrix $K = (K_{ij})$. The entry K_{ij} of K corresponds, in a certain way, the energy that a pair of interacting particles, one of which has spin i, the other one has spin j, contributes into the overall energy of G. We always assume K to be symmetric. A *configuration* of the system on a graph $G = (V, E)$ is a mapping $\sigma : V \to \{1, \ldots, k\}$. The *energy* of σ is the sum $H(\sigma) = \sum_{e=\{u,v\} \in E} K_{\sigma(u)\sigma(v)}$. Then the probability that G has configuration σ is $\frac{1}{Z} \exp(-H(\sigma)/cT)$, where $Z = \sum_{\sigma} \exp(-H(\sigma)/cT)$ is the *partition function* and T is a parameter of the system (the *temperature*) and c is a constant. As is easily seen, this probability distribution obeys the law "the lower energy a configuration has, the more likely it is". Observe that $Z = Z_A(G)$ for the matrix A with

$$A_{ij} = \exp(-K_{ij}/cT).$$

Thus $\mathrm{EVAL}(A)$ is just the problem of computing the partition function for the system described by A. Dyer and Greenhill in [6] dealt with spin systems in which certain configuration are prohibited and the others are uniformly distributed, while our results are applicable to arbitrary spin systems.

Preliminaries

$\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{Z}$ denote the real numbers, rational numbers and integers, respectively, and $\mathbb{Q}[X]$ and $\mathbb{Z}[X]$ denote the polynomial rings over $\mathbb{Z}$ and $\mathbb{Q}$ in an indeterminate X. Throughout this paper, we let $\mathbb{S}$ denote one of these five rings.

For every set S, $S^{m\times n}$ denotes the set of all $m \times n$-**matrices** with entries from S. For a matrix A, A_{ij} denotes the entry in row i and column j. The row rank of a matrix $A \in \mathbb{S}^{m\times n}$ is denoted by $\mathsf{rank}(A)$. A matrix $A \in \mathbb{S}^{m\times n}$ is *non-negative (positive),* if, for $1 \le i \le m, 1 \le j \le n$, the leading coefficient of A_{ij} is non-negative (positive, respectively).

Graphs are always undirected, unless we explicitly call them *directed graphs.* Graphs and directed graphs may have loops and multiple edges. The *degree, in-degree,* and *out-degree* of a vertex in a (directed) graph are defined in the obvious way and denoted by $\mathsf{deg}(v)$, $\mathsf{indeg}(v)$, $\mathsf{outdeg}(v)$, respectively.

Our model of real number computation is a standard model, as it is, for example, underlying the complexity theoretic work on linear programming (cf. [11]). We can either assume that the numbers involved in our computations are polynomial time computable or that they are given by an oracle (see [16] for a detailed description of the model). However, our results do not seem to be very model dependent. All we really need is that the basic arithmetic operations are polynomial time computable. Our situation is fairly simple because all real numbers we encounter are the entries of some matrix A, which is always considered fixed, and numbers computed from the entries of A using a polynomial number of arithmetic operations. Instances of the problem EVAL(A) are just graphs, and we do not have to worry about real numbers as inputs of our computations.

2 The Tractable Cases

2.1 Block Decompositions

Let $B \in \mathbb{S}^{k\times\ell}$. A *submatrix* of B is a matrix obtained from B by deleting some rows and columns. For non-empty sets $I \subseteq \{1,\ldots,k\}$, $J \subseteq \{1,\ldots,\ell\}$, where $I = \{i_1,\ldots,i_p\}$ with $i_1 < \ldots < i_p$ and $J = \{j_1,\ldots,j_q\}$ with $j_1 < \ldots < j_q$, B_{IJ} denotes the $(p \times q)$-**submatrix** with $(B_{IJ})_{rs} = B_{i_r j_s}$ for $1 \le r \le p, 1 \le s \le q$. A *proper submatrix* of B is a submatrix $B' \neq B$.

Definition 2. *Let* $B \in \mathbb{S}^{k\times\ell}$.

1. *A* decomposition *of* B *consists of two proper submatrices* B_{IJ}, $B_{I'J'}$ *such that*
   a) $I = \{1,\ldots,k\} \setminus I'$,
   b) $J = \{1,\ldots,\ell\} \setminus J'$,
   c) $B_{ij} = 0$ *for all* $(i,j) \in (I \times J') \cup (I' \times J)$.
   B *is* indecomposable *if it has no decomposition.*
2. *A* block *of* B *is an indecomposable submatrix* B_{IJ} *with at least one non-zero entry such that* $B_{IJ}, B_{I^{\mathbb{C}}J^{\mathbb{C}}}$ *with* $I^{\mathbb{C}} = \{1,\ldots,k\} \setminus I$ *and* $J^{\mathbb{C}} = \{1,\ldots,\ell\} \setminus J$ *is a decomposition of* B.

Indecomposability may be viewed as a form of "connectedness" for arbitrary matrices. For square matrices there is also a natural graph based notion of connectedness.

Let $A \in \mathbb{S}^{k\times k}$ be a square matrix. A *principal submatrix* of A is a submatrix of the form A_{II} for some $I \subseteq \{1,\dots,k\}$. Instead of A_{II} we just write A_I. The *underlying graph* of A is the (undirected) graph $G(A)$ with vertex set $\{1,\dots,k\}$ and edge set $\{\{i,j\} \mid 1 \le i,j \le n \text{ such that } A_{ij} \neq 0\}$. Note that we define $G(A)$ to be an undirected graph even if A is not symmetric.

Definition 3. *Let $A \in \mathbb{S}^{k\times k}$.*

1. *A is* connected *if the graph $G(A)$ is connected.*
2. *A* connected component *of the matrix A is a principal submatrix A_C, where C is the vertex set of a connected component of $G(A)$.*

Lemma 4. *A connected symmetric matrix is either indecomposable or bipartite. In the latter case, the matrix has two blocks corresponding to the two parts of the bipartition.*

There is another useful connection between indecomposability and connectedness. For a matrix $B \in \mathbb{S}^{k\times\ell}$, let

$$\mathsf{bip}(B) = \begin{pmatrix} 0 & B \\ 0 & 0 \end{pmatrix} \in \mathbb{S}^{(k+\ell)\times(k+\ell)}.$$

Note that $\mathsf{bip}(B)$ is the adjacency matrix of a weighted bipartite directed graph. The following lemma is straightforward.

Lemma 5. *Let $B \in \mathbb{S}^{k\times\ell}$ and $A = \mathsf{bip}(B)$. Then for every block B_{IJ} of B there is a connected component A_C of A such that $A_C = \mathsf{bip}(B_{IJ})$, and conversely for every connected component A_C of A there is a block B_{IJ} of B such that $A_C = \mathsf{bip}(B_{IJ})$.*

In particular, B is indecomposable if, and only if, A is connected.

2.2 Partition Functions of Graphs

Even though our main result is about symmetric matrices and (undirected) graphs, it is useful to generalise partition functions to directed graphs, which we do in the most straightforward way. Let $A \in \mathbb{S}^{k\times k}$ be a square matrix that is not necessarily symmetric and $G = (V, E)$ a directed graph. For every $\sigma : V \to \{1,\dots,k\}$ we let

$$\omega_A(\sigma) = \prod_{(u,v)\in E} A_{\sigma(u)\sigma(v)},$$

and we let

$$Z_A(G) = \sum_{\sigma:V\to\{1,\dots,k\}} \omega_A(\sigma).$$

Note that if A is symmetric, $G = (V, E)$ a directed graph, and G_U the underlying undirected graph, then $Z_A(G_U) = Z_A(G)$. Thus by EVAL(A) we may denote the problem of computing $Z_A(G)$ for a given directed graph, with the understanding that for symmetric A we can always consider the input graph as undirected.

Theorem 6. *Let $A \in \mathbb{R}^{k\times k}$ be a matrix such that each connected component of A has row rank* 1. *Then* EVAL(A) *is in polynomial time.*

Proof. Let $A_1, \ldots, A_\ell$ be the connected components of A. Then for every graph G with connected components $G_1, \ldots, G_m$ we have

$$Z_A(G) = \prod_{i=1}^{m} \sum_{j=1}^{\ell} Z_{A_j}(G_i).$$

Thus without loss of generality we may assume that A is connected.

Then $\mathsf{rank}(A) \leq 1$, and thus there are numbers $a_1, \ldots, a_k, b_1, \ldots, b_k \in \mathbb{R}$ such that for $1 \leq i, j \leq k$ we have:

$$A_{ij} = a_i \cdot b_j$$

(the b_j can be chosen to be the A_{1j} and $a_i = A_{i1}/A_{11}$). Let $G = (V, E)$ be a directed graph and $\sigma : V \to \{1, \ldots, k\}$. Then

$$\omega_A(\sigma) = \prod_{(v,w)\in E} A_{\sigma(v)\sigma(w)} = \prod_{(v,w)\in E} a_{\sigma(v)} b_{\sigma(w)} = \prod_{v\in V} a_{\sigma(v)}^{\mathsf{outdeg}(v)} b_{\sigma(v)}^{\mathsf{indeg}(v)}.$$

Thus

$$Z_A(G) = \sum_{\sigma: V\to\{1,\ldots,k\}} \omega_A(\sigma) = \sum_{\sigma} \prod_{v\in V} a_{\sigma(v)}^{\mathsf{outdeg}(v)} b_{\sigma(v)}^{\mathsf{indeg}(v)} = \prod_{v\in V} \sum_{i=1}^{k} a_i^{\mathsf{outdeg}(v)} b_i^{\mathsf{indeg}(v)}.$$

The last term can easily be evaluated in polynomial time. □

Corollary 7. *Let $A \in \mathbb{R}^{k\times k}$ be a symmetric matrix such that each connected component of A either has row rank at most* 1 *or is bipartite and has row rank at most* 2. *Then* EVAL(A) *is in polynomial time.*

Proof. We may assume that A is connected and bipartite with $\mathsf{rank}(A) = 2$. Then there are $k_1, k_2 \geq 1$ such that $k_1 + k_2 = k$ and a matrix $B \in \mathbb{R}^{k_1\times k_2}$ with $\mathsf{rank}(B) = 1$ and

$$A = \begin{pmatrix} 0 & B \\ B^{\mathsf{T}} & 0 \end{pmatrix}.$$

Let $G = (V, E)$ be a graph. If G is not bipartitite then $Z_A(G) = 0$, therefore, we may assume that G is connected and bipartite, say, with bipartition V_1, V_2. Let G_{12} be the directed graph obtained from G by directing all edges from V_1 to V_2, and let G_{21} be the directed graph obtained from G by directing all edges from V_2 to V_1. Recall that

$$\mathsf{bip}(B) = \begin{pmatrix} 0 & B \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{k\times k}.$$

We have

$$Z_A(G) = Z_{\mathsf{bip}(B)}(G_{12}) + Z_{\mathsf{bip}(B)}(G_{21}).$$

Since EVAL(bip(B)) is in polynomial time by Theorem 6, this shows that $Z_A(G)$ can be computed in polynomial time. □

3 Weighted Constraint Satisfaction Problems

It will be convenient for us to view constraint satisfaction problems as homomorphism problems (as first suggested by Feder and Vardi [8]). Recall that a *(relational)vocabulary* τ is a set of *relation symbols,* each with a prescribed arity. A *(relational) structure* $\mathcal{A}$ of vocabulary τ consists of a universe A and, for each r-ary relation symbol $\rho \in \tau$, a relation $\rho^{\mathcal{A}} \subseteq A^r$. Observe that a constraint language Γ on a domain D may be viewed as a relational structure $\mathcal{D}$ with universe D. Feder and Vardi call this structure the *template* of the problem $\mathrm{CSP}(\mathcal{D}) = \mathrm{CSP}(\Gamma)$. An instance $(V, D, \mathcal{C})$ of $\mathrm{CSP}(\mathcal{D})$ may be viewed as a structure $\mathcal{P}$ of the same vocabulary as $\mathcal{D}$. The universe of $\mathcal{P}$ is V, and for each relation symbol ρ in the vocabulary we let $\rho^{\mathcal{P}} = \{s \mid (s, \rho^{\mathcal{D}}) \in \mathcal{C}\}$. Then a *solution* is a homomorphism σ from $\mathcal{P}$ to $\mathcal{D}$. Note that with this notation the H-colouring problem simply becomes *CSP*(H).

The objective of the counting problem $\#\mathrm{CSP}(\mathcal{D})$ is to count the solutions for a given instance $\mathcal{P}$. We shall now define a weighted version of this problem. Let $\mathcal{D}$ be a template and $\omega : D \to \mathbb{S}$ be a *weight function.* Slightly abusing notation, we also use ω to denote the weight of a solution $\sigma : V \to D$ of $\mathrm{CSP}(\mathcal{D})$ for $\mathcal{V}$; we let

$$\omega(\sigma) = \prod_{v \in V} \omega(\sigma(v)).$$

As usually, V denotes the set of variables of the CSP, that is, the universe of the structure $\mathcal{P}$. We let

$$\mathcal{Z}_{\mathcal{D},\omega}(\mathcal{P}) := \sum_{\sigma} \omega(\sigma)$$

where the sum ranges over all solutions σ of $\mathrm{CSP}(\mathcal{D})$ for $\mathcal{P}$. We denote the problem of computing $\mathcal{Z}_{\mathcal{D},\omega}$ by $\mathrm{WCSP}(\mathcal{D}, \omega)$.

3.1 CSPs with Two Equivalence Relations

For the rest of this paper, we let α *and* β *be binary relation symbols.* Let $\mathcal{D} = (D, \alpha^{\mathcal{D}}, \beta^{\mathcal{D}})$ be a structure in which $\alpha^{\mathcal{D}}$ and $\beta^{\mathcal{D}}$ are equivalence relations on D, and let $\omega : D \to \mathbb{S}$ be a weight function. Suppose that the equivalence classes of $\alpha^{\mathcal{D}}$ are $C_1, \ldots, C_k$ and those of $\beta^{\mathcal{D}}$ are $D_1, \ldots, D_\ell$. Let $B = B(\mathcal{D}, \omega) \in \mathbb{S}^{k \times \ell}$ be defined by

$$B_{ij} = \sum_{d \in C_i \cap D_j} \omega(d).$$

The next lemma shows that the function $\mathcal{Z}_{\mathcal{D},\omega}$ only depends on the matrix B.

Lemma 8. *Let $\mathcal{D}, \mathcal{D}'$ be templates with two equivalence relations and $\omega : D \to \mathbb{S}$, $\omega' : D' \to \mathbb{S}$ weight functions. Suppose that $B(\mathcal{D}, \omega) = B(\mathcal{D}', \omega')$. Then $Z_{\mathcal{D},\omega} = Z_{\mathcal{D}',\omega'}$.*

The proof is straightforward. □

Conversely, for every matrix $B \in \mathbb{S}^{k\times\ell}$ we define a *canonical template* $\mathcal{D}_B$ and a *canonical weight function* ω_B as follows: The universe of $\mathcal{D}_B$ is $D_B = \{1,\ldots,k\} \times \{1,\ldots,\ell\}$, the equivalence relation $\alpha^{\mathcal{D}(B)}$ is equality on the first component, and $\beta^{\mathcal{D}(B)}$ is equality on the second component. The weight function $\omega_B : D_B \to \mathbb{S}$ is defined by $\omega_B((i,j)) = B_{ij}$. Then clearly $B = B(\mathcal{D}, \omega)$.

In the following, we write Z_B instead of $Z_{\mathcal{D}_B,\omega_B}$ and WCSP(B) instead of WCSP($\mathcal{D}_B, \omega_B$).

The following useful lemma is an immediate consequence of the definitions.

Lemma 9. *Let $B, B' \in \mathbb{S}^{k\times\ell}$ be such that B' is obtained from B by permuting rows and/or columns. Then $Z_B = Z_{B'}$.*

3.2 Back and Forth Between CSP and H-Colouring

The next lemma shows that weighted CSP for two equivalence relations are equivalent to evaluation problems for weighted bipartite graphs.

Lemma 10. *Let $B \in \mathbb{S}^{k\times\ell}$. Then the problems* WCSP(B) *and* EVAL(bip(B)) *are polynomial time equivalent.*

Proof. Let

$$A = \mathsf{bip}(B) = \begin{pmatrix} 0 & B \\ 0 & 0 \end{pmatrix} \in \mathbb{S}^{(k+\ell)\times(k+\ell)}.$$

Observe that for every directed graph $G = (V,E)$ we have $Z_A(G) = 0$ unless there is a bipartition V_1, V_2 of V such that $E \subseteq V_1 \times V_2$ (that is, all edges are directed from V_1 to V_2). Assuming that there is such a bipartition V_1, V_2, we let $\mathcal{P} = \mathcal{P}(G)$ be the $\{\alpha,\beta\}$-structure with universe E in which $\alpha^{\mathcal{P}}$ is the relation

$$\{(e,e') \in E^2 \mid e \text{ and } e' \text{ have the same endpoint in } V_1\}$$

and $\beta^{\mathcal{P}}$ is the relation

$$\{(e,e') \in E^2 \mid e \text{ and } e' \text{ have the same endpoint in } V_2\}.$$

Note that for every $\sigma : V \to \{1,\ldots,k+\ell\}$ we have $\omega_A(\sigma) = 0$ unless $\sigma(V_1) \subseteq \{1,\ldots,k\}$ and $\sigma(V_2) \subseteq \{k+1,\ldots,\ell\}$.

Recall the definition of the canonical template $\mathcal{D}_B$ and the canonical weight function ω_B. For a mapping $\sigma : V \to \{1,\ldots,k+\ell\}$ with $\omega_A(\sigma) \neq 0$, let $\sigma^* : E \to D_B$ be the mapping that maps $e = (u,v)$ with $u \in V_1, v \in V_2$ to $(\sigma(u), \sigma(v) - k)$. Observe that σ^* is a solution of CSP($\mathcal{D}_B$) for the instance $\mathcal{P}$. Conversely, every

solution of $\mathrm{CSP}(\mathcal{D}_B)$ for the instance $\mathcal{P}$ is of the form σ^* for some σ with $\omega_A(\sigma) \neq 0$. Furthermore, we have

$$\omega_A(\sigma) = \prod_{(v,w)\in E} B_{\sigma(v)\sigma(w)} = \omega_B(\sigma^*).$$

Thus

$$Z_A(G) = \sum_{\sigma} \omega_A(\sigma) = \sum_{\sigma^*} \omega_B(\sigma^*) = \mathcal{Z}_B(\mathcal{P}).$$

This yields a reduction from EVAL(A) to WCSP(B).

Let $\mathcal{P} = (P, \alpha^{\mathcal{P}}, \beta^{\mathcal{P}})$ be an instance of WCSP(B). Without loss of generality we may assume that $\alpha^{\mathcal{P}}$ and $\beta^{\mathcal{P}}$ are equivalence relations. To see this, just note that every solution of WCSP(B) for $\mathcal{P}$ is also a solution for the instance $(P, \overline{\alpha}^{\mathcal{P}}, \overline{\beta}^{\mathcal{P}})$, where $\overline{\alpha}^{\mathcal{P}}$ and $\overline{\beta}^{\mathcal{P}}$ are the reflexive symmetric transitive closures of $\alpha^{\mathcal{P}}$ and $\beta^{\mathcal{P}}$, respectively. Let $C_1, \ldots, C_k$ be the equivalence classes of $\alpha^{\mathcal{P}}$ and $D_1, \ldots, D_\ell$ the equivalence classes of $\beta^{\mathcal{P}}$. Let $G = (V, E)$ be the directed graph defined as follows: The vertex set is $V = \{1, \ldots, k+\ell\}$, and for $1 \leq i \leq k$, $1 \leq j \leq \ell$ there are $|C_i \cap D_j|$ edges from i to j. It is easy to see that $\mathcal{Z}_B(\mathcal{P}) = Z_A(G)$. This yields a reduction from WCSP(B) to EVAL(A). □

The following corollary is an immediate consequence of the preceding lemma and Lemma 5:

Corollary 11. *Let $B \in \mathbb{R}^{k\times\ell}$ such that every block of B has row rank at most 1. Then* WCSP(B) *is in polynomial time.*

The following lemma is needed to derive the hardness part of Theorem 1 from the hardness results on weighted CSP.

Lemma 12. *Let $A \in \mathbb{S}^{k\times k}$. Then* WCSP($A$) *is polynomial time reducible to* EVAL(A).

Proof. Let $A' = \mathsf{bip}(A)$. By Lemma 10, it suffices to prove that $\mathrm{EVAL}(A')$ is reducible to EVAL(A).

Let $G = (V, E)$ be a directed graph. If G is not bipartite with all edges directed from one part to the other, then $Z_{A'}(G) = 0$. Therefore, we assume that there is a partition V_1, V_2 of V such that $E \subseteq V_1 \times V_2$. We claim that

$$Z_{A'}(G) = Z_A(G). \tag{2}$$

Note that for every $\sigma' : V \to \{1, \ldots, 2k\}$ with $\omega_{A'}(\sigma) \neq 0$ we have $\sigma(V_1) \subseteq \{1, \ldots, k\}$ and $\sigma(V_2) \subseteq \{k+1, \ldots, 2k\}$.

For $\sigma : V \to \{1, \ldots, k\}$, let $f(\sigma) : V \to \{1, \ldots, 2k\}$ be defined by $f(\sigma)(v_1) = \sigma(v_1)$ and $f(\sigma)(v_2) = \sigma(v_2) + k$ for all $v_1 \in V_1$, $v_2 \in V_2$. Then $\omega_A(\sigma) = \omega_{A'}(\sigma')$. Moreover, f is one-to-one, and for every $\sigma' : V \to \{1, \ldots, 2k\}$ with $\omega_{A'}(\sigma') \neq 0$ there exists $\sigma : V \to \{1, \ldots, k\}$ such that $\sigma' = f(\sigma)$. This proves (2). □

4 The Main Hardness Theorem

Theorem 13. *Let $B \in \mathbb{S}^{k\times\ell}$ be non-negative such that at least one block of B has row rank at least 2. Then* WCSP(B) *is #P-complete.*

The full proof of Theorem 13 can be found in [4]. A brief outline of the proof will be given in the next subsection.

Note that, combined with Corollary 11, Theorem 13 yields a complete complexity theoretic classification of problems WCSP(B) for non-negative matrices $B \in \mathbb{R}^{k\times\ell}$.

Furthermore Theorem 1 follows easily from Theorem 6, Corollary 7 (for the tractability results) and Lemma 12, Theorem 13 (for the hardness results). Note that there is no contradiction between Theorem 1(2) and Theorem 13, because if the graph $G(A)$ of a symmetric matrix A is bipartite then A is not indecomposable.

4.1 Outline of the Proof

In this subsection we sketch the proof of Theorem 13. Let $B \in \mathbb{S}^{k\times\ell}$ be a non-negative matrix such that at least one block of B has row rank at least 2.

Step 1: From numbers to polynomials. In this first step of the proof we show that we can assume that all positive (i.e. non-zero) entries of B are powers of some indeterminate X. More precisely, we prove that there is a matrix B^* whose positive entries are powers of X such that B^* also has a block of row rank at least 2 and WCSP(B^*) is polynomial time reducible to WCSP(B). The construction is based on a lemma, which essentially goes back to [6], stating that the problem WCSP(B) is equivalent to the problem counting all solutions of a given weight. For simplicity, let us assume here that all entries of B are non-negative integers; additional tricks are required for real matrices. We can use the lemma to filter out powers of a particular prime p from all entries of B. This way we obtain a matrix B' whose positive entries are powers of a prime p. Using a technique which corresponds to "thickening" in the graph context (cf. [14,6]), we can replace the entries of this matrix by arbitrary powers, and by interpolation we can then replace p by the indeterminate X. This gives us the desired matrix B^*.

From now on, we assume that all positive entries of B are powers of X.

Step 2: Further preparations. Now we employ two results due to [1]. A consequence of the first is a lemma stating that if there are rows i, i' and columns j, j' such that the entries $B_{ij}, B_{ij'}, B_{i'j}$ are non-zero and $B_{i'j'}$ is zero, then WCSP(B) is #P-complete. This implies that we may assume that in every block of B all entries are positive.

The second result due to [1] is that the more general problem where we only count solutions in which the values of some variables are fixed (i.e., solutions extending some fixed partial solutions) is reducible to WCSP(B). This implies that we may assume that B is indecomposable.

Together, the two assumptions imply that all entries of B are positive.

Another simple reduction shows that either (a) each row of B and each column of B contains a 1, or (b) B has principal submatrix in which all entries are 1, and no 1s appear outside of this principal submatrix. From here we branch into cases (a) and (b).

Step 3(a): Separate 1*s.* We assume that all entries of B are positive and each row of B and each column of B contains a 1. Since we may permute rows and columns of B, we may assume that all diagonal entries of B are 1. It is not hard to see that then we can reduce the problem EVAL(A) for a symmetric non-singular 2×2-matrices to WCSP(B). For such matrices A the problem EVAL(A) is #P-hard. We believe that this is known (implicitly it is underlying [9]), but in absence of a clear reference we give a proof which boils down to a reduction from the problem of counting MAXCUTs of a graph.

Step 3(b): All 1s together. This part of the proof is the hardest, and it is difficult to describe on a high level. We assume that all entries of B are positive and that a principal submatrix in the upper left corner of B contains all 1s. We define a sequence $B^{[k]}$, for $k \geq 1$, of matrices that are obtained from B by some construction on the instances that is remotely similar to "stretching" and "thickening" (cf. [14,6]), but more complicated. We show that $\mathrm{WCSP}(B^{[k]})$ is reducible to WCSP(B) for all k.

The entries of the $B^{[k]}$ are polynomials with integer coefficients (no longer just powers of X as the entries of B). Employing a little bit of complex analysis, we prove that for some k, $B^{[k]}_{11}$ has an irreducible factor $p(X)$ such that the multiplicity of $p(X)$ in $B^{[k]}_{11}$ is higher than in all other entries in the first row and column, and the multiplicity in the corresponding diagonal entries is also sufficiently high. Using similar tricks as in Step 1, we can filter out the powers of this irreducible polynomial $p(X)$. We obtain a matrix whose weighted CSP is #P-complete by the results of Steps 2 and 3(a).

5 Conclusions

We give a complete complexity theoretic classification for the problem of evaluating the partition function of a symmetric non-negative matrix A, which may be viewed as the adjacency matrix of an undirected weighted graph H. Our proofs explore a correspondence between this evaluation problem and weighted constraint satisfaction problems for constraint languages with two equivalence relations.

Peculiarly, our proof does not go through for matrices with negative entries. Indeed, we do not know whether the evaluation problem for the matrix

$$\begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}$$

is #P-complete. (Observe that the evaluation problem for this matrix is equivalent to the problem of counting induced subgraphs with an even number of edges.)

The more important open problem is to obtain a classification result for the evaluation problem for non-symmetric matrices, corresponding to directed graphs. We believe that with our results such a classification may now be within reach, in particular because our main hardness result goes through for directed graphs. The ultimate goal of this line of research is a classification of counting and weighted CSP for arbitrary constraint languages. Towards a solution of this problem, one may try to reduce the weighted CSP to evaluation problems for directed graphs. It is interesting to note that the known reduction between the corresponding decision problems does not give a reduction between the counting problems we are interested in here.

Acknowledgement. We wish to thank Mark Jerrum for many useful discussions.

References

1. A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science, FOCS'03,* pages 562–571, 2003.
2. A.A. Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, FOCS'02,* pages 649–658, 2002.
3. A.A. Bulatov. Tractable conservative constraint satisfaction problems. In *Proceedings of the 18th Annual IEEE Simposium on Logic in Computer Science,* pages 321–330, 2003.
4. A.A. Bulatov and M. Grohe. The complexity of partition functions. Technical Report PRG-RR-04-04, Computing Laboratory, University of Oxford, Oxford, UK, 2004.
5. N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation,* 125(1):1–12, 1996.
6. M. Dyer and C. Greenhill. The complexity of counting graph homomorphisms. *Random Structures and Algorithms,* 17:260–289, 2000.
7. M.E. Dyer, L.A. Goldberg, and M. Jerrum. Counting and sampling H-colourings. In J.D.P. Rolim and S.P. Vadhan, editors, *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques,* volume 2483 of *Lecture Notes in Computer Science,* pages 51–67. Springer-Verlag, 2002.
8. T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal of Computing,* 28:57–104, 1998.
9. L.A. Goldberg, M. Jerrum, and M. Paterson. The computational complexity of two-state spin systems. *Random Structures and Algorithms,* 23:133–154, 2003.
10. L.A. Goldberg, S. Kelk, and M. Paterson. The complexity of choosing an H-colouring (nearly) uniformly at random. In *Proceedings of the 34rd ACM Simposium on Theory of Computing,* pages 53–62, 2002.

11. M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimazation.* Springer-Verlag, 1993. 2nd edition.
12. P. Hell and J. Nešetřil. On the complexity of *H*-coloring. *Journal of Combinatorial Theory, Ser.B,* 48:92–110, 1990.
13. P. Hell, J. Nešetřil, and X. Zhu. Duality and polynomial testing of tree homomorphisms. *Trans. of the AMS,* 348(4):1281–1297, 1996.
14. F. Jaeger, D.L. Vertigan, and D.J.A. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Mathematical Proceedings of the Cambridge Philosophical Society,* 108:35–53, 1990.
15. P.G. Jeavons, D.A. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM,* 44(4):527–548, 1997.
16. K. Ko. *Complexity Theory of Real Functions.* Birkhäuser, 1991.
17. T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th ACM Symposium on Theory of Computing,* pages 216–226, 1978.

Comparing Recursion, Replication, and Iteration in Process Calculi

Nadia Busi, Maurizio Gabbrielli, and Gianluigi Zavattaro

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura A.Zamboni 7, I-40127 Bologna, Italy.
busi,gabbri,zavattar@cs.unibo.it

Abstract. In [BGZ03] we provided a discrimination result between recursive definitions and replication in a fragment of CCS by showing that termination (i.e., all computations terminate) is undecidable in the calculus with recursion, whereas it turns out to be decidable in the calculus with replication. Here we extend the results in [BGZ03] by considering iteration, a third mechanism for expressing infinite behaviours. We show that convergence (i.e., the existence of a terminating computation) is undecidable in the calculus with replication, whereas it is decidable in the calculus with iteration. We also show that recursion, replication and iteration constitute a strict expressiveness hierarchy w.r.t. weak bisimulation: namely, there exist weak bisimulation preserving encodings of iteration in replication (and of replication in recursion), whereas there exist no weak bisimulation preserving encoding in the other direction.

1 Introduction

In this paper we continue the investigation we have started in [BGZ03], devoted to the comparison of different mechanisms used in the context of channel-based process calculi for extending finite processes with infinite behaviours. More precisely, we focus on three classical mechanisms, namely, *recursion, replication,* and *iteration.*

We adopt process constants to express recursion: we assume that each process constant D has an associated (possibly recursive) definition $D \stackrel{def}{=} P$. By using recursively defined process constants one can obtain an "in depth" infinite behaviour, since process copies can be nested at an arbitrary depth by using constant application. On the other hand, the replication operator $!P$ allows to create an unbounded number of parallel copies of a process P, thus providing an "in width" infinite behaviour, since the copies are placed at the same level. Finally, the iteration operator P^* permits to iterate the execution of a process P, i.e. at the end of the execution of one copy of P another copy can be activated. In this case, a "repetitive" infinite behaviour is supported, since the copies are executed one after the other.

In [BGZ03] we proved a discrimination result between recursion and replication in the context of a fragment of CCS [Mil89] with guarded choice and

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 307–319, 2004.

without relabelling. We showed that *termination,* i.e. all computations terminate, is undecidable in the calculus with recursion, whereas it turns out to be decidable in the calculus with replication.

In this paper we extend our previous work by taking into account other interesting properties of processes, and by investigating the decidability of these properties. More precisely, we consider process *convergence,* i.e. there exists a computation that terminates, *barb,* i.e. a process has the ability to perform a synchronization on a certain channel after a (possibly empty) internal computation, and *weak bisimulation.* We say that weak bisimulation is decidable if given any pair of processes, it is decidable whether those two processes are weakly bisimilar. The results of our investigation are reported in the following Table:[1]

	Recursion	Replication	Iteration
Termination	undecidable[BGZ03]	decidable[BGZ03]	decidable
Convergence	undecidable	undecidable	decidable
Barb	undecidable	decidable	decidable
Weak bisimulation	undecidable	undecidable	decidable

The undecidability results are proved by presenting an encoding of Random Access Machines [SS63] (RAMs), a well known deterministic Turing powerful formalism. In [BGZ03] we showed an encoding of RAMs in the calculus with recursion. The encoding is deterministic, i.e. it presents a unique possible computation that reflects the computation of the corresponding RAM. This proves that termination and convergence are undecidable. By exploiting a slightly different encoding, it is possible to prove the undecidability of barb as well as of weak bisimulation. The idea is to extend the modeling of RAMs with an observable action that can be performed on program termination; in this way we reduce the problem of testing the termination of a RAM to the problem of detecting an observable behaviour.

The decidability of process termination for the calculus with replication implies the impossibility to provide a termination preserving encoding of RAMs. The existence of encodings that preserve "weaker" properties was left as an open problem. In this paper, we answer positively to this question by showing how to model RAMs in a nondeterministic manner. The encoding is nondeterministic in the following sense: computations which do not follow the expected behaviour of the modeled RAM are introduced by the encoding, but all these computations are infinite. This proves that a process modeling a RAM has a terminating computation, i.e. converges, if and only if the corresponding RAM terminates. Thus, process convergence is undecidable for the calculus with replication.

The nondeterministic modeling of RAMs under replication permits us to prove that also weak bisimulation is undecidable, simply by following a technique

[1] In the present paper we consider the fragment of CCS with general choice, whereas in [BGZ03] we considered the calculus with guarded choice. Decidability results presented in [BGZ03] can be easily adapted to the calculus with general choice. Clearly, the undecidability results of [BGZ03] continue to hold also in this extended calculus.

Table 1. The transition system for finite core CCS (symmetric rules of omitted).

$$\textbf{PRE}: \quad \alpha.P \xrightarrow{\alpha} P \qquad \textbf{PAR}: \quad \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \textbf{SUM}: \quad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$$

$$\textbf{RES}: \quad \frac{P \xrightarrow{\alpha} P'}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \; x \notin n(\alpha) \qquad \textbf{COM}: \quad \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

similar to the one described above for the calculus with recursion. Interestingly, we have that even if weak bisimulation is undecidable under replication, barb turns out to be decidable. This is proved by resorting to the theory of well structured transition systems [FS01].

For the calculus with process iteration we have that all the properties are decidable. This is a consequence of the fact that the processes of this calculus are finite state. Intuitively, this follows from the fact that each iteration activates one copy at a time (thus only a predefined number of processes can be active at the same time) and all the copies share the same finite set of possible states.

2 The Calculi

We start considering the finite fragment of the core of CCS (that we sometimes call simply CCS for brevity). After we present the three infinite extensions.

Definition 1. (finite core CCS) *Let Name, ranged over by $x, y, \ldots$, be a denumerable set of channel names. The class of finite core CCS processes is described by the following grammar:*

$$P ::= \mathbf{0} \mid \alpha.P \mid P+P \mid P|P \mid (\nu x)P \qquad \alpha ::= \tau \mid x \mid \overline{x}$$

The term $\mathbf{0}$ denotes the empty process while the term $\alpha.P$ has the ability to perform the action α (which is either the unobservable τ action or a synchronization on a channel x) and then behaves like P. Two forms of synchronization are available, the output $\overline{x}$ or the input x. The sum construct + is used to make choice among the summands while parallel composition | is used to run parallel programs. Restriction $(\nu x)P$ makes the name x local in P. We denote the process $\alpha.\mathbf{0}$ simply with α, and the process $(\nu x_1)(\nu x_2)\ldots(\nu x_n)P$ with $(\nu \tilde{x})P$ where $\tilde{x}$ is the sequence of names $x_1, x_2, \ldots, x_n$.

For input and output actions, we write $\overline{\alpha}$ for the complementary of α; that is, if $\alpha = x$ then $\overline{\alpha} = \overline{x}$, if $\alpha = \overline{x}$ then $\overline{\alpha} = x$. We write $fn(P)$, $bn(P)$ for the *free names* and the *bound names* of P. The *names* of P, written $n(P)$, is the union of the free and bound names of P. The names in a label α, written $n(\alpha)$ is the set of names in α, i.e. the empty set if $\alpha = \tau$ or the singleton $\{x\}$ if α is either x or $\overline{x}$. Table 1 contains the set of the transition rules for finite CCS.

Table 2. The rules for the $\surd$ transitions.

$$\mathbf{0} \xrightarrow{\surd} \bullet \qquad P^* \xrightarrow{\surd} \bullet \qquad \frac{P \xrightarrow{\surd} \bullet}{(\nu x)P \xrightarrow{\surd} \bullet}$$

$$\frac{P \xrightarrow{\surd} \bullet \quad Q \xrightarrow{\surd} \bullet}{P|Q \xrightarrow{\surd} \bullet} \qquad \frac{P \xrightarrow{\surd} \bullet \quad Q \xrightarrow{\surd} \bullet}{P+Q \xrightarrow{\surd} \bullet} \qquad \frac{P \xrightarrow{\surd} \bullet \quad Q \xrightarrow{\surd} \bullet}{P;Q \xrightarrow{\surd} \bullet}$$

Definition 2. ($\mathbf{CCS}_D$) *We assume a set of constants, ranged over by D. The class of CCS_D processes is defined by adding the production $P ::= D\langle\tilde{x}\rangle$ to the grammar of Definition 1. It is assumed that each constant D has a unique defining equation of the form $D \stackrel{def}{=} (\tilde{x})P$, where $(\tilde{x})$ is a binder for the names in the sequence of names in $\tilde{x}$. Both in a constant definition $D \stackrel{def}{=} (\tilde{x})P$ and in a constant application $D\langle\tilde{x}\rangle$, the parameter $\tilde{x}$ is a tuple of all distinct names. As usual, in case the sequence $\tilde{x}$ is empty, we omit the surrounding parentheses. Moreover, we assume that $fn(P) \subseteq n(\tilde{x})$ where $n(\tilde{x})$ denotes the set of names in the sequence $\tilde{x}$.*

The transition rules for constant is

$$\frac{P\{\tilde{x}/\tilde{y}\} \xrightarrow{\alpha} P'}{D\langle\tilde{x}\rangle \xrightarrow{\alpha} P'} \quad \text{if } D \stackrel{def}{=} (\tilde{y})Q$$

where $P\{\tilde{x}/\tilde{y}\}$ is the term obtained by replacing all the free occurrences of the names in $\tilde{y}$ with the corresponding names in $\tilde{x}$.

Definition 3. ($\mathbf{CCS}_!$) *The class of $CCS_!$ processes is defined by adding the production $P ::= !P$ to the grammar of Definition 1.*

The transition rule for replication is

$$\frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

Definition 4. ($\mathbf{CCS}_*$) *The class of CCS_* processes is defined by adding the production $P ::= P^*$ to the grammar of Definition 1.*

Intuitively, the process P^* has the ability to iterate the behaviour of the process P an arbitrary number of time (possibly zero times). In order to formally describe the semantics of iteration we explicitly represent the ending of process P with the transition $P \xrightarrow{\surd} \bullet$, where $\surd \notin Name$ is a new label and $\bullet$ is

an auxiliary operator. We also exploit an auxiliary operator $P; Q$ denoting the sequential composition of processes. Informally, given the process $P; Q$ we have that the process Q can start only if $P \xrightarrow{\surd} \bullet$. Formally, the axioms and rules for $\surd$ transitions are reported in Table 2. The transition rules for iteration are

$$\frac{P \xrightarrow{\alpha} P'}{P^* \xrightarrow{\alpha} P'; P^*} \qquad \frac{P \xrightarrow{\alpha} P'}{P; Q \xrightarrow{\alpha} P'; Q} \qquad \frac{P \xrightarrow{\surd} \bullet \quad Q \xrightarrow{\alpha} Q'}{P; Q \xrightarrow{\alpha} Q'}$$

We use $\prod_{i \in I} P_i$ to denote the parallel composition of the indexed processes P_i, while we use $\prod_n P$ to denote the parallel composition of n instances of the process P (if $n = 0$ then $\prod_n P$ denotes the empty process $\mathbf{0}$).

Given a process Q, its internal runs $Q \longrightarrow Q_1 \longrightarrow Q_2 \longrightarrow \ldots$ are given by its reduction steps, (denoted with $\longrightarrow$), i.e. by those transitions $\longrightarrow$ that the process can perform in isolation, independently of the context. The internal transitions $\longrightarrow$ correspond to the transitions labeled with τ plus the ending $\surd$ transitions, i.e. $P \longrightarrow P'$ iff $P \xrightarrow{\tau} P'$ or $P \xrightarrow{\surd} P'$. We denote with $\longrightarrow^*$ the reflexive and transitive closure of $\longrightarrow$. With *Deriv(P)* we denote the set of processes reachable from P with a sequence of reduction steps: $Deriv(P) = \{Q \mid P \longrightarrow^* Q\}$.

A process Q is *dead* if there exists no Q' such that $Q \longrightarrow Q'$. We say that a process P *converges* if there exists a dead process P' in *Deriv(P)*. We say that P *terminates* if all its internal runs terminate, i.e. the process P cannot give rise to an infinite computation: formally, P *terminates* iff there exist no $\{P_i\}_{i \in \mathbf{N}}$, s.t. $P_0 = P$ and $P_j \longrightarrow P_{j+1}$ for any j. Observe that *process termination* implies *process convergence* while the vice versa does not hold.

Barbs are used to observe whether a process has the ability to perform, possibly after an internal run, an observable action on a specific channel; formally $P \Downarrow x$ iff there exist P' and P'' s.t. $P \longrightarrow^* P' \xrightarrow{\alpha} P''$ and $n(\alpha) = \{x\}$.

Definition 5. (weak bisimulation) *A binary, symmetric relation $\mathcal{R}$ on processes is a* weak bisimulation *if $(P, Q) \in \mathcal{R}$ implies that, if $P \xrightarrow{\alpha} P'$, then one of the following holds:*

- *there exist Q', Q'', Q''' s.t. $Q \longrightarrow^* Q' \xrightarrow{\alpha} Q'' \longrightarrow^* Q'''$ and $(P', Q''') \in \mathcal{R}$;*
- *$\alpha = \tau$ and there exists Q' s.t. $Q \longrightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$.*

Two processes P and Q are weakly bisimilar, *written $P \approx Q$, if there exists a weak bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.*

3 Undecidability Results for CCS!

We prove that CCS! is powerful enough to model, at least in a nondeterministic way, any Random Access Machine [SS63] (RAM), a well known register based Turing powerful formalism.

A RAM (denoted in the following with R) is a computational model composed of a finite set of registers $r_1, \ldots, r_n$, that can hold arbitrary large natural numbers, and by a program composed by indexed instructions $(1 : I_1), \ldots, (m : I_m)$,

that is a sequence of simple numbered instructions, like arithmetical operations (on the contents of registers) or conditional jumps. An internal state of a RAM is given by $(i, c_1, \ldots, c_n)$ where i is the program counter indicating the next instruction to be executed, and $c_1, \ldots, c_n$ are the current contents of the registers $r_1, \ldots, r_n$, respectively. Given a configuration $(i, c_1, \ldots, c_n)$, its computation proceeds by executing the instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached; in this case we say that the configuration $(i, c_1, \ldots, c_n)$ terminates.

In [Min67] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : Succ(r_j))$: adds 1 to the contents of register r_j;
- $(i : DecJump(r_j, s))$: if the contents of register r_j is not zero, then decreases it by 1 and go to the next instruction, otherwise jumps to instruction s.

Our encoding is nondeterministic because it introduces computations which do not follow the expected behaviour of the modeled RAM. However, all these computations are infinite. This ensures that, given a RAM, its modeling has a terminating computation if and only if the RAM terminates. This proves that *convergence* is undecidable.

Exploiting the encoding, we also prove that weak bisimulation is undecidable. The idea is to use only two observable actions, namely $\overline{w}$ and $\overline{w'}$. The former makes visible the fact that the program counter has reached an index outside the original range $1 \ldots m$ of program instructions; the latter makes visible the activation of an incorrect infinite computation. In this way, we have that a correct terminating run of the encoding has the following property; at its end it executes the action $\overline{w}$, after which it cannot produce any further observable action. Thus, if P is the encoding of a RAM R, then R terminates if and only if $P \approx \tau.P + \overline{w}$. This proves that *weak bisimulation* is undecidable.

In this section we reason up to a structural congruence $\equiv$ in order to rearrange the order of parallel composed processes and to abstract away from the terminated processes $\mathbf{0}$. We define $\equiv$ as the least congruence relation satisfying the usual axioms $P|Q \equiv Q|P$, $P|(Q|R) \equiv (P|Q)|R$, and $P|\mathbf{0} \equiv P$.

Let R be a RAM with registers $r_1, \ldots, r_n$, and instructions $(1 : I_1), \ldots, (m : I_m)$. We model separately registers and instructions.

The program counter is modeled with a message $\overline{p}_i$ indicating that the i-th instruction is the next to be executed. For each $1 \leq i \leq m$, we model the i-th instruction $(i : I_i)$ of R with a process which is guarded by an input operation p_i. Once activated, the instruction performs its operation on the registers, then waits for an acknowledgement indicating that the operation has been performed, and finally updates the program counter by producing $\overline{p_{i+1}}$ (or $\overline{p_s}$ in case of jump).

Formally, for any $1 \leq i \leq m$, the instruction $(i : I_i)$ is modeled by $[\![(i : I_i)]\!]$ which is a shorthand notation for the following processes.

$$[\![(i : I_i)]\!] \; : \quad !p_i.(\overline{inc_j} \mid inc.\overline{p_{i+1}}) \qquad \text{if } I_i = Succ(r_j)$$

$$[\![(i : I_i)]\!] \; : \quad !p_i.(\overline{dec_j} \mid (dec.\overline{p_{i+1}} + zero.\overline{p_s})) \qquad \text{if } I_i = DecJump(r_j, s)$$

It is worth noting that a program counter message $\overline{p_i}$, with the index i outside the range $1 \ldots m$, is produced on program termination. Let TI the set of the terminating indexes given by $m+1$ plus all those indexes greater than m that are target of some jump instruction. For each index $i \in TI$ we will assume the presence of a process $p_i.\overline{w}$ able to consume the program counter message and communicate program termination on the channel w.

We model each register r_j, when it contains c_j, with the following process simply denoted with $[\![r_j = c_j]\!]$ in the following:

$$\begin{aligned}[\![r_j = c_j]\!] \;:\; & \overline{nr_j} \mid !nr_j.\,(\nu\; m\; i\; d\; u)(\\ & \overline{m} \mid !m.(inc_j.\bar{i} + dec_j.\bar{d}) \mid !i.(\overline{m} \mid \overline{inc} \mid \overline{u} \mid d.u.(\overline{m}|\overline{dec})) \mid \\ & d.(\overline{zero} \mid u.DIV \mid \overline{nr_j}) \mid \textstyle\prod_{c_j}(\overline{u} \mid d.u.(\overline{m} \mid \overline{dec}))\;\;)\end{aligned}$$

where DIV is a process able to activate an infinite observable computation, for instance $\overline{w'} \mid !w'.\overline{w'}$.

Observe that the content c_j of the register is modeled by the parallel composition of a corresponding number of processes $(\overline{u} \mid d.u.(\overline{m}|\overline{dec}))$; the term $\overline{u}$ represents a unit inside the register, while $d.u.(\overline{m}|\overline{dec})$ is an auxiliary term that is responsible for removing the unit when the register is decremented.

The name nr_j is used to activate and restart the register. This is because the register is modeled as a replicated process, and the message $\overline{nr_j}$ is used to spawn a new replica. The name m is used to activate the so-called manager of the register: $(inc_j.\bar{i} + dec_j.\bar{d})$. The manager handles an increment or a decrement request and produces the local name $\bar{i}$ or $\bar{d}$, respectively. The modeling of register increment is easy, while register decrement introduces nondeterminism.

Two different processes may synchronize with $\bar{d}$: either $(d.u.(\overline{m} \mid \overline{dec}))$ or $(d.(\overline{zero} \mid u.DIV \mid \overline{nc}))$. In the first case the register is actually decremented while in the second case a jump is executed. Observe that the jump could occur even if the register is not empty, nevertheless, if the register is not empty this means that at least one instance of $\overline{u}$ is available. This ensures that the computation cannot terminate as the process $u.DIV$ is spawn. In case the the register is actually empty, the old instance of the register remains represented by the deadlocked "garbage" process

$$G_j \;:\; (\nu\; m\; i\; d\; u)(!m.(inc_j.\bar{i} + dec_j.\bar{d}) \mid !i.(\overline{m} \mid \overline{inc} \mid \overline{u} \mid d.u.\overline{dec}) \mid u.DIV)$$

Definition 6. *Let R be a RAM with program instructions $(1 : I_1), \ldots, (m : I_m)$ and registers $r_1, \ldots, r_n$. Given the configuration $(i, c_1, \ldots, c_n)$ of R we define*

$$\begin{aligned}&[\![(i, c_1, \ldots, c_n)]\!]_R \;= \\ &(\nu\; p_1, \ldots, p_m, nr_1, inc_1, dec_1, \ldots, nr_n, inc_n, dec_n, inc, dec, zero) \\ &(\;\; \overline{p_i} \mid [\![(1 : I_1)]\!] \mid \ldots \mid [\![(m : I_m)]\!] \mid \textstyle\prod_{i \in TI} p_i.\overline{w} \mid \\ &\quad [\![r_1 = c_1]\!] \mid \ldots \mid [\![r_n = c_n]\!] \mid \textstyle\prod_{k_1} G_1 \mid \ldots \mid \prod_{k_n} G_n \;\;)\end{aligned}$$

where the modeling of program instructions $[\![(i : I_i)]\!]$, the modeling of registers $[\![r_j = c_j]\!]$, the set of terminating indexes TI, and the garbage $G_1, \ldots, G_n$ have

been defined above, and $k_1 \dots k_n$ *are natural numbers. Observe that due to the presence of* $k_1 \dots k_n$ *the target of the encoding is not a unique process but it is a class of processes which differ only in the amount of garbage.*

Theorem 1. *Let R be a RAM with program* $(1 : I_1), \dots, (m : I_m)$ *and state* $(i, c_1, \dots, c_n)$*, and let the process P be in* $[\![(i, c_1, \dots, c_n)]\!]_R$*. Then* $(i, c_1, \dots, c_n)$ *terminates if and only if P converges. Moreoverm P converges if and only if* $P \approx \tau.P + \overline{w}$*.*

This proves that *convergence* and *weak bisimulation* are undecidable in $\text{CCS}_!$.

4 Decidability Results for $\text{CCS}_!$

We show that barb is a decidable property in the calculus with replication. This result is based on the theory of well-structured transition systems [FS01]; first of all, we recall the alternative semantics for $\text{CCS}_!$ defined in [BGZ03], that is equivalent to the one presented in Section 2, but is based on a finitely branching transition system. Then, by exploiting the theory developed in [FS01], we show that barb is decidable for $\text{CCS}_!$ processes.

We start recalling some basic definitions and results of [FS01], concerning well-structured transition systems, that will be used in the following. A *quasi-ordering* is a reflexive and transitive relation over a set *X*. Given a quasi-ordering $\leq$ over *X*, an *upward-closed set* is a subset $I \subseteq X$ such that the following holds: $\forall x, y \in X : (x \in I \wedge x \leq y) \Rightarrow y \in I$. Given $x \in X$, we define $\uparrow x = \{y \in X \mid x \leq y\}$. Given $Y \subseteq X$, we define $\uparrow Y = \bigcup_{y \in Y} \uparrow y$. A *finite basis* of an upward-closed set *I* is a finite set *B* such that $I = \bigcup_{x \in B} \uparrow x$.

Definition 7. *A* well-quasi-ordering *(wqo) is a quasi-ordering* $\leq$ *over a set X such that, for any infinite sequence* $x_0, x_1, x_2, \dots$ *in X, there exist indexes* $i < j$ *such that* $x_i \leq x_j$*.*

Note that, if $\leq$ is a wqo, then any infinite sequence $x_0, x_1, x_2, \dots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \dots$ (with $i_0 < i_1 < i_2 < \dots$).

Transition systems can be formally defined as follows.

Definition 8. *A* transition system *is a structure* $TS = (S, \rightarrow)$*, where S is a set of* states *and* $\rightarrow \subseteq S \times S$ *is a set of* transitions*. We write* $Succ(s)$ *(resp.* $Pred(s)$*) to denote the set* $\{s' \in S \mid s \rightarrow s'\}$ *of immediate successors (resp.* $\{s' \in S \mid s' \rightarrow s\}$ *of immediate predecessors) of* s*. We write* $\rightarrow^+$ *(resp,* $\rightarrow^*$*) for the transitive (resp. the reflexive and transitive) closure of* $\rightarrow$*. We write* $Pred^*(s)$ *to denote the set* $\{s' \in S \mid s' \rightarrow^* s\}$*. TS is* finitely branching *if all* $Succ(s)$ *are finite.*

We restrict to finitely branching transition systems. Well-structured transition system, defined as follows, provide the key tool to decide properties of computations.

Definition 9. *A* well-structured transition system with strong compatibility *is a transition system* $TS = (S, \rightarrow)$, *equipped with a quasi-ordering* $\leq$ *on* S, *such that the two following conditions hold:*

1. **well-quasi-ordering:** $\leq$ *is a well-quasi-ordering, and*
2. **strong compatibility:** $\leq$ *is (upward) compatible with* $\rightarrow$, *i.e., for all* $s_1 \leq t_1$ *and all transitions* $s_1 \rightarrow s_2$, *there exists a state* t_2 *such that* $t_1 \rightarrow t_2$ *and* $s_2 \leq t_2$.

Definition 10. *A well-structured transition system has* effective pred-basis *if there exists an algorithm accepting any state* $s \in S$ *and returning* $pb(s)$, *a finite basis of* $\uparrow Pred(\uparrow s)$.

The following proposition (a special case of a result in [FS01]) will be used to obtain our decidability result.

Proposition 1. *Let* $TS = (S, \rightarrow, \leq)$ *be a finitely branching, well-structured transition system with strong compatibility, decidable* $\leq$ *and effective pred-basis. It is possible to compute a finite basis of Pred*(I) for any upward-closed set I given via a finite basis.*

As the results on well-structured transition systems apply to finitely branching transition systems, we need to use the alternative semantics for $\mathrm{CCS}_!$ defined in [BGZ03], that is based on a finitely branching transition system and that is equivalent to the semantics presented in Section 2. The new semantics is obtained by reformulating the (non finitely branching) semantics of replication defined in Definition 3. The new transition relation $\overset{\alpha}{\mapsto}$ over $\mathrm{CCS}_!$ processes is the least relation satisfying all the axioms and rules of Table 1 (where $\overset{\alpha}{\mapsto}$ is substituted for $\xrightarrow{\alpha}$), plus the following rules REPL1 and REPL2.

$$\textsc{repl1}: \frac{P \overset{\alpha}{\mapsto} P'}{!P \overset{\alpha}{\mapsto} P' \mid !P} \qquad \textsc{repl2}: \frac{P \overset{\alpha}{\mapsto} P' \quad P \overset{\overline{\alpha}}{\mapsto} P''}{!P \overset{\tau}{\mapsto} P' \mid P'' \mid !P}$$

As done for the standard transition system, we assume that the reductions $\mapsto$ of the new semantics corresponds to the τ–labeled transitions $\overset{\tau}{\mapsto}$. Barbs in the new semantics are defined in the obvious way: $P \Downarrow x$ iff there exists P', P'', α s.t. $P \mapsto^* P' \overset{\alpha}{\mapsto} P''$ and $n(\alpha) = \{x\}$. We have the following result:

Proposition 2. *Let* $P \in CCS_!$. *Then* $P \Downarrow x$ *iff* $P \Downarrow x$.

In [BGZ03] we defined a preorder $\preceq$ on $\mathrm{CCS}_!$processes and, by exploiting Higman's lemma [Hig52], we proved that $\preceq$ is a well-quasi-ordering compatible with $\mapsto$, thus obtaining a well-structured transition system. In this section we show that the obtained well-structured transition system has an effective pred-basis. Thus, exploiting the Proposition 1 we show that $P \Downarrow x$ is decidable.

We start recalling the definition of $\preceq$ and the results of [BGZ03] that will be used to prove the decidability of barb. The definition of the wqo on processes needs the following structural congruence, that turns out to be compatible with $\mapsto$.

Definition 11. *We define* $\equiv$ *as the least congruence relation satisfying the following axioms:* $P|Q \equiv Q|P \qquad P|(Q|R) \equiv (P|Q)|R \qquad P|\mathbf{0} \equiv P$

Proposition 3. *Let* $P, Q \in CCS_!$. *If* $P = Q$ *and* $Q \overset{\alpha}{\mapsto} Q'$ *then there exists* P' *such that* $P \overset{\alpha}{\mapsto} P'$ *and* $P' \equiv Q'$.

Now we are ready to define the preorder on processes:

Definition 12. *Let* $P, Q \in CCS_!$. *We write* $P \preceq Q$ *iff there exist* n, $x_1, \ldots, x_n, P', R, P_1, \ldots, P_n, Q_1, \ldots, Q_n$ *such that* $P \equiv P'|\prod_{i=1}^{n}(\nu x_i)P_i$, $Q \equiv P'|R|\prod_{i=1}^{n}(\nu x_i)Q_i$, *and* $P_i \preceq Q_i$ *for* $i = 1, \ldots, n$.

Definition 13. *Let* $P \in CCS_!$. *With* $d_\nu(P)$ *we denote the maximum number of nested restrictions in process P:*

$d_\nu(\mathbf{0}) = 0 \qquad d_\nu(\alpha.P) = d_\nu(P)$

$d_\nu(P + Q) = max(\{d_\nu(P), d_\nu(Q)\}) \qquad d_\nu(P|Q) = max(\{d_\nu(P), d_\nu(Q)\})$

$d_\nu((\nu x)P) = 1 + d_\nu(P) \qquad d_\nu(!P) = d_\nu(P)$

The set of sequential and bang subprocesses of P is defined as:

$Sub(\mathbf{0}) = \emptyset \qquad Sub(\alpha.P) = \{\alpha.P\} \cup Sub(P)$

$Sub(P + Q) = \{P + Q\} \cup Sub(P) \cup Sub(Q) \qquad Sub(P|Q) = Sub(P) \cup Sub(Q)$

$Sub((\nu x)P) = Sub(P) \qquad Sub(!P) = \{!P\} \cup Sub(P)$

Definition 14. *Let* n *be a natural number and P a process. With* $\mathcal{P}_{P,n}$ *we denote the set of* $CCS_!$ *processes whose sequential subprocesses, bang subprocesses and bound names are contained in the corresponding elements of P, and with a nesting level of restrictions not greater than* n:

$\mathcal{P}_{P,n} = \{Q \in CCS_! \mid Sub(Q) \subseteq Sub(P) \wedge bn(Q) \subseteq bn(P) \wedge d_\nu(Q) \leq n\}$.

We define the set of processes that can immediately perform a labelled move; we show that this set is upward-closed and we provide this set with a finite basis.

Definition 15. *Let* $P \in CCS_!$. *The set of processes* $Now_\alpha(P)$ *is defined as* $\{Q \in \mathcal{P}_{P,d_\nu(P)} \mid Q \overset{\alpha}{\mapsto}\}$.

Proposition 4. *Let* $P \in CCS_!$. *The set of processes* $Now_\alpha(P)$ *is upward-closed.*

Definition 16. *Let* $P \in CCS_!$. *The set* $fbNow_\alpha(P)$ *is defined as follows:*

$$fbNow_\alpha(P) = \{(\nu x_1 \ldots x_m)Q \mid Q \in Sub(P), m \leq d_\nu(P), x_1 \ldots x_m \subseteq bn(P), Q \overset{\alpha}{\not\mapsto}, n(\alpha) \notin \{x_1, \ldots, x_m\}\}$$

Proposition 5. *Let* $P \in CCS_!$ *and* $\alpha \neq \tau$. *Then the set* $fbNow_\alpha(P)$ *is a finite basis of* $Now_\alpha(P)$.

It is possible to provide a method to construct a finite basis for the set of predecessors of a given process w.r.t. a transition $\overset{\alpha}{\mapsto}$; hence, the following holds:

Theorem 2. *Let $P \in CCS_!$. Then the transition system $(\mathcal{P}_{P,d_\nu(P)}, \mapsto, \preceq)$ is a well-structured transition system with strong compatibility, decidable $\preceq$ and effective pred-basis.*

As a consequence of this result, by Proposition 1 it is possible to compute a finite basis of $Pred^*(I)$ for any upward-closed set I specified through a finite basis. The possibility to compute a finite basis of $Pred^*(I)$, together with the decidability of $\preceq$, provides a method to decide if a given process Q belongs to $Pred^*(I)$, as it is sufficient to verify if there exists a process in the finite basis that is smaller than Q. Since the set of processes $Now_\alpha(P)$ – that can immediately perform a (not silent) move α – is upward-closed, and we provided a finite basis for it in Definition 16, we have that it is possible to decide if a process belongs to $Pred^*(Now_\alpha(P))$.

From the following Proposition, which provides a characterisation of $P\Downarrow x$ in terms of belonging to $Pred^*(Now_\alpha(P))$, we obtain the decidability of $P\Downarrow x$.

Proposition 6. *Let $P \in CCS_!$. $P\Downarrow x$ iff $P \in Pred^*(Now_x(P))$ or $P \in Pred^*(Now_{\overline{x}}(P))$.*

Corollary 1. *Let $P \in CCS_!$. Then $P\Downarrow x$ is decidable.*

5 Decidability Results for CCS_*

We show that the set of processes reachable from a given process P is finite. Hence, all the properties considered in this paper are decidable in CCS_*.

Definition 17. *Reach(P) is the set of terms reachable from P with a sequence of transitions:* $Reach(P) = \{Q \mid \exists n \geq 0, \alpha_1, \ldots, \alpha_n \text{ s.t. } P \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} Q\}$.

We provide an upper bound to the number of reachable processes:

Definition 18. *The function size on CCS_* processes is defined as follows:*

$size(\mathbf{0}) = 1$ $\quad$ $size(\alpha.P) = 1 + size(P)$

$size(P + Q) = 1 + size(P) + size(Q)$ $\quad$ $size(P|Q) = size(P) \times size(Q)$

$size((\nu x)P) = size(P)$ $\quad$ $size(P*) = size(P) + 1$

Proposition 7. *Let $P \in CCS_*$. Then $|Reach(P)| \leq size(P)$.*

Corollary 2. *Let $P \in CCS_*$. The set Reach(P) is finite.*

As a consequence of the above corollary, we obtain that termination, convergence and barb are decidable in CCS_*, as well as weak bisimulation [KS90, PT87].

6 Conclusion and Related Work

As a consequence of the results we have proved in the paper there exists a strict hierarchy of expressiveness w.r.t. weak bisimulation among the three considered infinite operators. In fact, there exist encodings of replication in recursion, and of iteration in replication that preserve weak bisimulation, while the vice versa does not hold. To encode replication using recursive definitions, we consider an encoding which is homomorphic except for $[\![!P]\!] = D\langle\tilde{x}\rangle$ with $D = (\tilde{x})([\![P]\!]|D\langle\tilde{x}\rangle)$ where $\tilde{x}$ is a list containing the free names of P. In order to model iteration using replication it is simply necessary to spawn replicas only on termination of the previous one. This can be done following the typical encoding of the continuation passing style. The encodings in the opposite direction do not exist. Replication cannot be encoded in terms of iteration because weak bisimulation is decidable only under iteration; recursion cannot be encoded into replication because barb is decidable only under replication and weak bisimulation preserves barbs.

In a related paper [GSV04] Giambagi, Schneider and Valencia consider other infinite operators in the setting of CCS. In their approach, two calculi are equally expressive if there exists a weak bisimulation preserving encoding of one calculus in the other, and vice versa. In their paper they leave as an open problem the existence of a weak bisimulation preserving encoding from recursion to replication. In this paper we close this open problem proving that such an encoding does not exist. In [NPV02] it is shown that replication is strictly less expressive than recursive definitions in the context of timed concurrent constraint languages. Because of the very different underlying computational model, the proof techniques exploited in that paper cannot be applied directly in the context of CCS. Recently, the undecidability of weak bisimulation has been proved by Srba [S03] also for PA processes. PA and our calculus are incomparable as PA considers sequential composition and does not contain restriction.

References

[BGZ03] N. Busi, M. Gabbrielli, and G. Zavattaro. Replication vs. Recursive Definitions in Channel Based Calculi. In *Proc. ICALP'03,* LNCS 2719, pages 133–144, Springer-Verlag, 2003.

[FS01] A. Finkel and Ph. Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science,* 256:63–92, 2001.

[GSV04] P. Giambiagi, G. Schneider and F.D. Valencia. On the Expressiveness of CCS-like Calculi In Proceedings of FOSSACS 04. LNCS 2987, pages 226–240, Springer-Verlag, 2004.

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. In *Proc. London Math. Soc.,* vol. 2, pages 236–366, 1952.

[KS90] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation,* 86(1):43–68, 1990

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[Min67] M. L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Englewood Cliffs, 1967.

[NPV02] M. Nielsen, C. Palamidessi, and F. D. Valencia. On the Expressive Power of Temporal Concurrent Constraint Programming Languages. In *Proc. of PPDP'02*. ACM Press, 2002.

[PT87] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing,* 16(6):973–989, 1987

[SS63] J. C. Shepherdson and J. E. Sturgis. Computability of recursive functions. *Journal of the ACM,* 10:217–255, 1963.

[S03] J. Srba. Undecidability of Weak Bisimilarity for PA-Processes. In Proc. of DLT'02, LNCS 2450, pages 197–208, Springer-Verlag, 2003.

Dynamic Price Sequence and Incentive Compatibility*
(Extended Abstract)

Ning Chen[1], Xiaotie Deng[2], Xiaoming Sun[3], and Andrew Chi-Chih Yao[4]

[1] Dept. of Computer Science, Fudan University, China
nchen@fudan.edu.cn
[2] Dept. of Computer Science, City University of Hong Kong
csdeng@cityu.edu.hk
[3] Dept. of Computer Science and Technology, Tsinghua University, China
sun_xm97@mails.tsinghua.edu.cn
[4] Dept. of Computer Science, Princeton University
yao@cs.princeton.edu

Abstract. We introduce and study a new auction model in which a certain type of goods is offered over a period of time, and buyers arrive at different times and stay until a common deadline (unless their purchase requests have been fulfilled). We examine in this model incentive compatible auction protocols (*i.e.,* those that induce participants to bid their true valuations).

We establish an interesting connection between incentive compatibility and price sequence: incentive compatibility forces a non-decreasing price sequence under some assumptions on market pricing schemes. We should point out that negation of our assumptions would require market distortions to some extent.

Our protocol may not ensure that one item must be sold everyday. Imposing such a market intervention, we show an impossibility result that deterministic incentive compatible auction protocols do not exist. With randomized relaxation, we give such an incentive compatible auction protocol. We also discuss incentive compatible protocols under other market conditions.

1 Introduction

The interplay of Computer Science and Economics has for quite a long time leaned towards the application of computer science concepts to those of economics [15,6,17,7,8]. Recently, many interesting ideas in economics, including the concept of incentive compatibility [19,5,11], which has played a central role in the studies of auction and related economic issues, started to make their ways into the studies of Computer Science and the Internet [16].

* This research is fully supported by a research grant (CityU1081/02E) from Research Grants Council of Hong Kong SAR, China, and research grants (60223004, 60321002, 60273045) from Natural Science Foundation of China.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 320–331, 2004.

The new economic platform of the Internet and electronic goods has brought renewed interests and new insight into the age-old problem. In recent work of digital-goods, where items can be sold in unlimited number of copies [10, 9], the main concerns have been incentive-compatibility and profit-maximizing for the auctioneer. One interesting result states that a digital-goods auction is incentive-compatible if and only if it is bid-independent [10,9]. As an example of bid-independent auction, the auctioneer can choose an arbitrary price at every instance of sales. Lavi and Nisan [12] studied the online auction where the auctioneer is required to respond to each bid as it arrives at different times, and characterized the incentive compatible protocols. Blum et al. [4] studied a general model in which both buyers and sellers stay for some periods after arriving, and discussed competitive strategies. For similar discussions on online auction and incentive compatibility see, *e.g.,* [1,2,3].

In the markets for many products, however, the price sequence exhibits certain patterns. For example, air-ticket price tends to rise towards the take-off date (of course, there are exceptions such as the last-minute price). In this paper we establish an interesting connection between incentive compatibility and price sequence towards a deadline in a semi-dynamic time auction setting. Our model is different from the standard online auction model in the following way: In our model, buyers may arrive at different times for a certain type of goods and they stay until a deadline or be offered to get goods, while in the ordinary online model, buyers arrive and leave at the same time with or without the goods. Both the standard model and ours may be practical models, relevant for modelling reality in different situations.

The price is shown going up if we want to have an incentive compatible auction protocol over days some items are sold, under mild assumptions. Our assumptions require that the market allows anyone to win if he bids sufficiently higher than all others, that the price does not depend on the particular buyers but on the bids submitted, and that the price may not go down if all bids are higher or equal. It is clear that lifting of those restrictions may be viewed as market interventions. In this case, social harmony relies on an inflationary economy. It is interesting to see such a phenomenon even under this limited constraint, in particular without introduction of interest rate or discounted future utility.

Our work may reveal an interesting direction in the study of price dynamics. The dynamics of goods prices is a difficult problem and is proposed, under the general equilibrium pricing model, by Smale as one of the most important mathematical problems for the 21st century [18]. Our study is based on an alternative economic pricing model.

We introduce some properties of incentive compatible auction protocols in Section 2, together with notations. Central to the concepts discussed here is that of critical value. The main idea is that the winning buyer pays a price that is dependent on the submitted bids of the other buyers. This idea is well known, and has been used in [13,14] to study combinatorial auctions, where interesting characterizations are obtained for incentive compatible protocols. Our presentation is motivated by, but slightly different from, their works in that the

payment scheme is a little bit different and the starting axioms are also somewhat different. Therefore, we supply the lemmas in our model for completeness of the presentation, which by no means claims the results in Subsection 2.1 are our own.

In Section 3, we propose a deterministic incentive compatible protocol for the semi-dynamic auction model. Noticeably, the protocol forces a non-decreasing price sequence. In Section 4, we give strong evidence that this is to some extent unavoidable, by proving that the price sequence is non-decreasing for any deterministic incentive compatible protocol.

In Section 5, we discuss the necessity of those assumptions and present various cases of price sequences under other market conditions. Note that our incentive compatible auction protocols may not sell one item every day. We show that introducing such a market restriction will result in an impossibility result for deterministic incentive compatible protocols. Whereas for randomized relaxation, we give such an auction protocol to reach incentive compatibility in expectation. We also discuss auction protocols that utilize customer discriminating strategies to obtain incentive compatibility. Finally, we conclude our work with remarks and future studies in Section 6.

2 Preliminaries

We consider a price-based auction model in which an auctioneer sells a set of homogeneous goods to potential buyers. Each buyer desires exactly one item of the goods (buyers with multiple units requests can be reduced to this one). We denote buyers by $i, i = 1, 2, \ldots, n$. Each buyer i has a privately known *valuation* $v_i \in \mathbb{N}$, representing the maximal value that i would like to pay for the goods.

Each buyer i submits a *bid* $b_i \in \mathbb{N} \cup \{0\}$ to the auctioneer. When receiving all submitted bids from buyers, the auctioneer specifies the *winners* and the *price* $p \in \mathbb{R}^+ \cup \{0\}$ of the goods. If buyer i wins the goods, *i.e.,* i is a winner, his *utility* is $u_i = v_i - p$. If i does not win the goods, his *utility* is zero.

Here we assume all buyers are rational and aim to maximize their utilities. Note that to maximize the utility value, buyers might not submit their valuations truthfully according to different auction protocols. We say an auction is *incentive compatible* (or *truthful*) if for any buyer i and the submitted bids of other buyers, buyer i's utility is maximized by submitting his true valuation, *i.e.,* $b_i = v_i$.

We shall discuss some properties of incentive compatible auction protocols and then introduce notations for our semi-dynamic model.

2.1 Critical Values for Buyers Under Incentive Compatible Auctions

In this paper, we consider auctions with the *non-trivial property:* Any buyer with $b_i = 0$ will not win the goods; whereas if a buyer bids sufficiently large (*e.g.,* $b_i = +\infty$), he must win the goods.

Lemma 1 *For any incentive compatible auction, the non-trivial property implies the* participation constraints: *If buyer i with bid b_i wins, then we must have $b_i \geq p$.*

We establish the following observations to winners and losers, respectively. Most of the similar properties are previously known (see, *e.g.*, for single-minded auction [14]). We present them for the completeness of our discussion.

Lemma 2 *In incentive compatible auction ψ with non-trivial property, assume buyer i with bid b_i wins the goods at price p. If i bids $b'_i > p$, $b'_i \neq b_i$, rather than b_i, as long as the submitted bids of other buyers do not change, he still wins the goods at the same price p. In addition, if i bids $b'_i < p$, he will not win the goods.*

Lemma 3 *In incentive compatible auction ψ with non-trivial property, assume buyer i with bid b_i does not win the goods. Then there exists a unique minimal integer (critical value) $r_i(\psi, b_{-i}) > b_i$ such that i always wins the goods when he bids $b'_i \geq r_i(\psi, b_{-i})$, where b_{-i} is the collection of submitted bids of buyers except i.*

The above two lemmas define the concept of *critical value:* the one for all the winners is the same: the price; and the one for the losers may not be the same and be the price. We will make use of the concept in the following discussions.

2.2 Semi-dynamic Auction Model

We consider a special type of auction model, *semi-dynamic auction.* An auctioneer sells a type of goods to potential buyers. The process of auction will last for several consecutive (and discrete) time units. For convenience, we shall use *day* as the time unit, denoted by t. Some units of the goods (determined by the auction protocol) will be sold each day.

Let $b_{i,t} \in \mathbb{N} \cup \{0\}$ be the submitted *bid* of buyer i on the t-th day, and $p_t \in \mathbb{R}^+ \cup \{0\}$ be the *price* of the goods on the t-th day. Note that for any buyer, we allow he submits different bids on different days. If buyer i wins the goods on the t-th day, his *utility* is $u_i = v_i - p_t$, where v_i is the true valuation of i. Otherwise, his *utility* is zero. We will use the following notations:

- D: The time span, *i.e.,* the number of days.
- $d_i \in \{1, \ldots, D\}$: The first day that i can appear as a buyer. It may choose to arrive later as an adversary action but not earlier than d_i. We assume that i appears in the continuous days of the domain $\{d_i, \ldots, D\}$, unless he wins the goods (and consequently, quit).
- $r_{i,t}$: The *critical value* of buyer i at time t. Let A_t be the collection of buyers that appear on the t-th day, $1 \leq t \leq D$. For any time t and $i \in A_t$, if i is a loser, define $r_{i,t} = r_i(\psi, b_{-i})$ (the value defined in Lemma 3). If i is a winner, define $r_{i,t} = p_t$. Let $R_t = \max_{i \in A_t} r_{i,t}$.

An auction protocol is called *incentive compatible* if for any time t and any set of submitted bids of other buyers, the utility of buyer i is maximized by submitting his true valuation, *i.e.,* $b_{i,t} = v_i$, for all $d_i \leq t \leq D$.

Here, we should get the meaning of price p_t: It is possible that not every buyer i with bid $b_{i,t} > p_t$ would win the goods. In this case, there may be a fixed quantity, say δ_t, of the goods for sale on each day. There might be more than δ_t buyers bidding higher than p_t, some buyers would still lose while others are selected winners according to the auction protocol.

3 An Incentive Compatible Semi-dynamic Auction Protocol

Let Ψ be the collection of all incentive compatible auction protocols for the ordinary one period case (*i.e.,* $D = 1$) satisfying all buyers with bids higher than the price win the goods (for example, Vickrey auction [19]). For any $\psi \in \Psi$, let $p(\psi, Z)$ be the price of the goods when the auctioneer selects auction protocol ψ, upon receiving submitted bids vector Z.

Deterministic Auction Scheme:

1. The auctioneer selects $\psi \in \Psi$ arbitrarily, and sets $R_0 = 0$.
2. For $t = 1, \ldots, D$
   (i) let $p_t = \max\{R_{t-1}, p(\psi, Z_t)\}$ be the price of the goods on the t-th day, where Z_t is the submitted bids vector this day,
   (ii) all buyers with bids higher than p_t win the goods,
   (iii) compute the critical value $r_{i,t}$ for each buyer in A_t, and let $R_t = \max_{i \in A_t} r_{i,t}$.

Example 1 We assume that for each day, the auctioneer always selects 1-item Vickrey (second-price) auction [19]. On the first day, for instance, buyers $A_1 = \{1, \ldots, k_1\}$ appear to the auction with submitted bids $b_{1,1} \geq b_{2,1} \geq \cdots \geq b_{k_1,1}$ (ties are broken arbitrarily), respectively. Therefore, according to the above Deterministic Auction Protocol, the price of the goods is the second highest bid $b_{2,1}$ (*i.e.,* $p_1 = b_{2,1}$). If $b_{1,1} > b_{2,1}$, then buyer 1 wins the goods; otherwise, no buyer wins the goods. In this case, the critical value for every loser is $b_{1,1}$. Hence, $R_1 = b_{1,1}$. On the next day, if the second highest bid is not less than R_1, then price p_2 is set to be that bid; otherwise, $p_2 = R_1$.

Theorem 1 *The above Deterministic Auction Protocol is incentive compatible.*

Intuitively and informally, since the price goes up, the best chance of the buyers is at the first day of entry to the market. They would not lie by the

incentive compatible requirement for the single period. The detailed proof is omitted here and will be presented in the journal version.

We comment that if we change R_{t-1} in determining p_t to anything smaller, say $R_{t-1} - \epsilon$, the protocol is no longer incentive compatible. In particular we cannot replace R_{t-1} by p_{t-1} in the protocol, as the following example shows.

Example 2 We still consider 1-item Vickrey auction. On the first day, three buyers come to auction with submitted bids 20,15,10, respectively. Specifically, we consider the behavior of buyer 2, let his valuation be 15 (*i.e.,* $v_2 = 15$). If he bids 15 truthfully on the first day, then we know that (i) buyer 1 (with submitted bid 20) wins, (ii) $p_1 = 15$, and (iii) on the second day, $p_2 = \max\{p_1, 10\} = 15$, which implies that the utility of buyer 2 is always zero. If buyer 2 bids 11 untruthfully, however, then $p_1 = 11$ and $p_2 = \max\{p_1, 10\} = 11$. Thus, he wins the goods on the second day with utility $15 - 11 > 0$.

4 Non-decreasing Property of Price Sequence

We prove here that the price sequence is non-decreasing in general if we assume the auction protocol is required to be incentive compatible. We make the following mild assumptions on the pricing protocols:

- *Non-trivial:* As defined in Section 2.
- *Non-discriminating:* The price p_t only depends on the sets of submitted bids in the previous t rounds: $p_t(B_1, B_2, \ldots, B_t)$, where B_j is the (multi) set of submitted bids on the j-th day, $1 \le j \le t$. That is, the bids are detached from buyers when determining prices. As a special case, if two buyers exchange their bids at a given time, the price does not change: $p(i : \alpha; j : \beta) = p(i : \beta; j : \alpha)$. where $p(i : \alpha; j : \beta)$ denotes the price of the goods when i bids α and j bids β.
- *Monotone:* For any time t, t_1, $1 \le t_1 \le t$, $p_t(B_1, B_2, \ldots, B_{t_1} \cup \{\alpha\}, \ldots, B_t) \ge p_t(B_1, B_2, \ldots, B_{t_1} \cup \{\alpha'\}, \ldots, B_t)$, for any $\alpha > \alpha'$.

Note that, non-trivial property and non-discriminating property are related but the former statement is about the winners of the goods and the latter one is about the winning price. Both are axioms describing the anonymity of the buyers.

Lemma 4 *Let* $t > 0$ *and* $p_t = p(i : b_{i,t}; j : b_{j,t})$. *If* $b_{i,t} \ge b_{j,t} > p_t$ *or* $b_{i,t} > b_{j,t} \ge p_t$, *and if buyer* j *wins, then buyer* i *also wins the goods at time* t.

Sketch of the Proof. Assume, to the contrary, that buyer i does not win the goods. Let $r_{i,t}$ be the critical value of i at time t. Due to Lemma 3, we know that $r_{i,t} > b_{i,t}$. Since $r_{i,t}, b_{i,t} \in \mathbb{N}$, we have $r_{i,t} - 1 \ge b_{i,t}$. Note that $b_{i,t} > p_t$, it follows that

$$p_t < r_{i,t} - 1. \tag{1}$$

If buyer i bids $r_{i,t}$, he wins the goods at price $p(i : r_{i,t}; j : b_{j,t})$. We claim that

$$r_{i,t} - 1 \leq p(i : r_{i,t}; j : b_{j,t}) \tag{2}$$

Otherwise, $p(i : r_{i,t}; j : b_{j,t}) < r_{i,t} - 1$. By Lemma 2, if buyer i bids $r_{i,t} - 1$, he also wins the goods. This contradicts to $r_{i,t}$'s definition.

Since $b_{j,t} \leq b_{i,t}$, we have

$$p(i : r_{i,t}; j : b_{j,t}) \leq p(i : r_{i,t}; j : b_{i,t}), \tag{3}$$

due to the monotone property. By Lemma 2 and $p(i : b_{i,t}; j : b_{j,t}) = p_t < r_{i,t}$, we have

$$p(i : b_{i,t}; j : r_{i,t}) = p(i : b_{i,t}; j : b_{j,t}). \tag{4}$$

Combining (1), (2), (3), (4), we have $p(i : b_{i,t}; j : r_{i,t}) = p(i : b_{i,t}; j : b_{j,t}) = p_t < r_{i,t} - 1 \leq p(i : r_{i,t}; j : b_{j,t}) \leq p(i : r_{i,t}; j : b_{i,t})$, which contracts to the non-discriminating property. □

Lemma 5 *For any time $t > 0$, assume the price set by the auctioneer is p_t. Then any buyer with bid $b_{i,t} > p_t$ must win the goods at time t.*

Sketch of the Proof. Assume to the contrary, that there exists a loser i with bid $b_{i,t} > p_t$. Suppose buyer $1', \ldots, \delta_t'$ win the goods with submitted bids $b_{1',t} \geq b_{2',t} \geq \cdots \geq b_{\delta_t',t}$, respectively, where δ_t is the fixed quantity of the goods for sale at time t. For buyer $1'$, due to Lemma 2, we know that if he bids $b'_{1',t} = b_{i,t} > p_t$, he still wins the goods at price p_t. Note that $b'_{1',t} = b_{i,t} < b_{\delta_t',t} \leq \cdots \leq b_{2',t}$. From Lemma 4, we know that buyer i, along with $1', 2', \ldots, \delta_t'$, should also win the goods when $1'$ bids $b'_{1',t}$. That is, there are at least $\delta_t + 1$ items to be sold at time t, which contradicts to our assumption. □

Lemma 6 *For any two days $t, t+1$, $1 \leq t < D$, if at least one item of the goods is sold, the price must satisfy $p_t \leq p_{t+1}$.*

Sketch of the Proof. Assume buyers $A_t = \{1, \ldots, k_t\}$ appear on the t-th day, with submitted bids $b_{1,t} \geq b_{2,t} \geq \cdots \geq b_{k_t,t}$, and buyers $A_{t+1} = \{1', \ldots, k'_{t+1}\}$ appear on the $(t+1)$st day, with submitted bids $b_{1',t+1} \geq b_{2',t+1} \geq \cdots \geq b_{k'_{t+1},t+1}$, respectively.

Due to Lemma 4 and our assumptions, we know that buyer 1 (and $1'$) win the goods at price p_t (and p_{t+1}) on the t-th (and $(t+1)$st) day, respectively. Assume to the contrary that $p_t > p_{t+1}$. We following consider two cases.

Case 1. There exists a winner j' on the $(t+1)$st day such that $d_{j'} < t+1$, *i.e.,* he loses on the t-th day. Let $r_{j',t} > b_{j',t}$ be his critical value on that day. Then $p(j' : r_{j',t}) \geq p(j' : b_{j',t}) = p_t > p_{t+1}$, where the first inequality is due to the monotone property of the price function. Consider the state that $r_{j',t}$ be the true valuation of buyer j', if he bids truthfully on the t-th day, his utility is $r_{j',t} - p(j' : r_{j',t})$. Whereas if he bids $b_{j',t}$ on the t-th day, and bids $b_{j',t+1}$ on the $(t+1)$st day, he will win the goods with utility $r_{j',t} - p_{t+1} > r_{j',t} - p(j' : r_{j',t})$. A contradiction to incentive compatibility.

Case 2. For all winner j' on the $(t+1)$st day, $d_{j'} = t+1$. Specially, $d_{1'} = t+1$. For the buyer 1 on the t-th day, let $b_{1,t}$ be his true valuation, *i.e.*, $v_1 = b_{1,t}$. Hence, if he bids truthfully on the t-th day, his utility is $v_1 - p_t$. Following we consider the case that buyer 1 bids zero on the t-th day (which implies that he loses on the t-th day), and bids $b_{1',t+1}$ on the $(t+1)$st day. We remove buyer $1'$ from the auction on the $(t+1)$st day (note that $d_{1'} = t+1$). Assume the new price is p'_{t+1}. Note that the set of bids on $(t+1)$st day is the same now, due to the monotone property of the price function, we have $p'_{t+1} \leq p_{t+1}$. Now the utility of buyer 1 is $v_1 - p'_{t+1} \geq v_1 - p_{t+1} > v_1 - p_t$. A contradiction.

Therefore, the lemma follows. □

Theorem 2 *Let $p_1, \ldots, p_D$ be a price sequence of consecutive transactions, then $p_1 \leq p_2 \leq \cdots \leq p_D$.*

Sketch of the Proof. We may skip all the days with no transactions, and the protocol and the transactions will not change. Then the theorem follows by Lemma 6. □

5 Incentive Compatibility Under Other Market Conditions

In this section, we discuss incentive compatible protocols under various market conditions.

5.1 An Impossibility Result

Theorem 3 *For any buyers with arbitrary bids, if $D > 1$ and at least one item of the goods is sold each day, then the deterministic incentive compatible auction protocol satisfying non-trivial, non-discriminating and monotone properties does not exist.*

The key point of the theorem is the non-decreasing property showed by Lemma 6 and the fact that at least one item of the goods is sold each day.

Sketch of the Proof of Theorem 3. Note that on the t-th day, all buyers with bids higher than p_t win the goods. According to our requirement, however, at least one buyer will win the goods on the $(t+1)$st day, no matter what bids of buyers are submitted. Hence, we may consider a special case that the submitted bid of each buyer on the $(t+1)$st day is strictly smaller than p_t. Therefore, we must have $p_t > p_{t+1}$, where p_{t+1} is the price of the goods on the $(t+1)$st day, which contradicts to Lemma 6. □

5.2 A Randomized Incentive Compatible Auction Protocol

The impossibility result leaves open the question whether we can ensure incentive compatibility when a fixed number of items are required to be sold each time. In this subsection, we introduce one randomized solution under the following restrictions:

- For convenience, we assume that one item is sold each time. That is, there are totally m items to be sold in $D = m$ continuous days, one item each day. The general case is similar.
- As in a ration system of war time, we assume that each buyer bids for the goods only once, and his following bid is the same to his first commitment. That is, $b_{i,t} = b_{i,d_i}$, for $d_i \leq t \leq D$.

Note that for randomized protocols, the meaning of incentive compatibility is to guarantee that truthful bid always maximizes a buyer's expected utility, *i.e.,* the auction is *incentive compatible in expectation.*

Randomized Auction Protocol:

1. For $t = 1, \ldots, D$
   (i) For each buyer, its entry bid is taken as its bids at subsequent time,
   (ii) let the price p_t be the $(m + 2 - t)$st highest submitted bid at time t,
   (iii) sell one item to one of the first $(m+1-t)$ buyers whose bids are not less than p_t, with probability $\frac{1}{m+1-t}$ each (*i.e.,* exactly one buyer wins).

For example, if $m = 2$, the auctioneer sells two items in two continuous days. On the first day, define the price to be the third highest submitted bid, and sell one item to the first two buyers with probability 1/2 each. On the second day, sell the remaining item in terms of 1-item Vickrey auction [19].

Lemma 7 *The price of the goods is non-decreasing, i.e.,* $p_1 \leq p_2 \leq \ldots \leq p_m$.

Therefore if the buyer wins the goods with zero probability on the t-th day, he still can not win in the following days.

Theorem 4 *The above Randomized Auction Protocol sells exactly one item of the goods and is incentive compatible in expectation.*

Sketch of the Proof. For convenience, we denote the submitted bid of buyer i by b_i. For arbitrary fixed submitted bids of other buyers, we only need to prove that for any b_i, we have $E[u_i(v_i)] \geq E[u_i(b_i)]$, where $E[u_i(b_i)]$ is the expected utility of i when submitting b_i. Without loss of generality, assume that $d_i = 1$, *i.e.,* buyer i appears on the first day.

Let $S_t = \{j \mid d_j = t, j \neq i\}$, $t = 1, \ldots, m$. Let S_0 denote the losers before buyer i appears. Next, we prove that for any $S_0, S_1, \ldots, S_m$, and b_i,

$$E[u_i(v_i; S_0, S_1, \ldots, S_m)] \geq E[u_i(b_i; S_0, S_1, \ldots, S_m)], \tag{5}$$

by mathematical induction on the number of items m.

If $m = 1$, it is equivalent to the deterministic 1-item Vickrey auction, so we always have $u_i(v_i; S_0, S_1) \geq u_i(b_i; S_0, S_1)$. Assume (5) holds for the case of $(m - 1)$. Following we consider there are m items to be sold.

Case 1. $v_i \leq p_1(i : v_i)$, where $p_1(i : v_i)$ is the price of the goods of the first day when i bids v_i. Therefore, $u_i(v_i; S_0, S_1, \ldots, S_m) = 0$. If $b_i \leq p_1(i : b_i)$, then $u_i(b_i; S_0, S_1, \ldots, S_m) = 0$, and (5) holds. Otherwise, $b_i > p_1(i : b_i)$. It is easy to see that $p_1(i : b_i) \geq v_i$. By Lemma 7 we know that the price is non-decreasing. Thus $E[u_i(b_i; S_0, S_1, \ldots, S_m)] \leq 0$.

Case 2. $v_i > p_1(i : v_i)$, then

$$E[u_i(v_i; S_0, S_1, \ldots, S_m)]$$

$$= \frac{1}{m}(v_i - p_1(i : v_i)) + \frac{1}{m} \sum_{\substack{j:j\neq i \\ b_{j,1} > p_1(i:v_i)}} E[u_i(v_i; S_0 \cup S_1 - \{j\}, S_2, \ldots, S_m)] \quad (6)$$

We may assume that $b_i > p_1(i : b_i)$, otherwise

$$E[u_i(b_i; S_0, \ldots, S_m)] \leq 0 \leq E[u_i(v_i; S_0, \ldots, S_m)].$$

It is easy to see that $p_1(i : b_i) = p_1(i : v_i)$, and

$$E[u_i(b_i; S_0, S_1, \ldots, S_m)]$$

$$= \frac{1}{m}(v_i - p_1(i : b_i)) + \frac{1}{m} \sum_{\substack{j:j\neq i \\ b_{j,1} > p_1(i:b_i)}} E[u_i(b_i; S_0 \cup S_1 - \{j\}, S_2, \ldots, S_m)]. \quad (7)$$

By the induction hypothesis, we have

$$E[u_i(v_i; S_0 \cup S_1 - \{j\}, S_2, \ldots, S_m)] \geq E[u_i(b_i; S_0 \cup S_1 - \{j\}, S_2, \ldots, S_m)]. \quad (8)$$

Combining (6), (7), (8), we have

$$E[u_i(v_i; S_0, S_1, \ldots, S_m)]$$

$$= \frac{1}{m}(v_i - p_1(i : v_i)) + \frac{1}{m} \sum_{\substack{j:j\neq i \\ b_{j,1} > p_1(i:v_i)}} E[u_i(v_i; S_0 \cup S_1 - \{j\}, S_2, \ldots, S_m)]$$

$$\geq \frac{1}{m}(v_i - p_1(i : b_i)) + \frac{1}{m} \sum_{\substack{j:j\neq i \\ b_{j,1} > p_1(i:b_i)}} E[u_i(b_i; S_0 \cup S_1 - \{j\}, S_2, \ldots, S_m)]$$

$$= E[u_i(b_i; S_0, S_1, \ldots, S_m)].$$

Hence (5) holds for any m, and the theorem follows. □

5.3 Discriminative Incentive Compatible Auction Protocols

If discriminative pricing scheme is allowed, such as the case in many information products, software systems, for example, the price sequence over time may not be decreasing.

As an example, we may sort the customers according to their names. At time t, we consider the first buyer in the ordered list. If his submitted bid is not less than that of the second buyer, he wins the goods at the price of the second buyer's bid. Otherwise, we remove this buyer from the list and consider the next one. In this protocol, we exactly sell one item every day. It is not hard to see this is a bid-independent protocol. And it is not difficult to verify it is incentive compatible.

Other interesting incentive compatible auction protocols exist when discriminative pricing protocols are used.

6 Conclusion and Discussions

In this paper, we discuss the connections between incentive compatibility and price sequence for the semi-dynamic auction model, where the auction lasts for several consecutive time units and buyers appear to the auction in the continuous time units until he wins the goods. The problem deserves further investigation into other different models.

- As an example, suppose that all buyers come to auction on the first day with different maximum departure dates, what is the characterization on price dynamics for incentive compatible protocols?

Note that there is a symmetry with respect to time in comparison with the model discussed here. However, it is not very clear how would the approach be carried over for it. More generally, it would be interesting to understand the full dynamics of price system in response to the dynamics of participating agents of the market.

References

1. A. Bagchi, A. Chaudhary, R. Garg, M. T. Goodrich, V. Kumar, *Seller-Focused Algorithms for Online Auctioning,* WADS 2001, 135-147.
2. Z. Bar-Yossef, K. Hildrum, F. Wu, *Incentive-Compatible Online Auctions for Digital Goods,* SODA 2002, 964-970.
3. A. Blum, V. Kumar, A. Rudra, F. Wu, *Online Learning in Online Auctions,* SODA 2003, 202-204.
4. A. Blum, T. Sandholm, M. Zinkevich, *Online Algorithms for Market Clearing,* SODA 2002, 971-980.
5. E. H. Clarke, *Multipart Pricing of Public Goods,* Public Choice, V.11, 17-33, 1971.
6. X. Deng, C. H. Papadimitriou, *On the Complexity of Cooperative Solution Concepts,* Mathematics of Operations Research, V.19(2), 257-266, 1994.
7. X. Deng, T. Ibaraki, H. Nagamochi, *Algorithmic Aspects of the Core of Combinatorial Optimization Games,* Mathematics of Operations Research, V.24(3), 751-766, 1999.
8. X. Deng, Z. Li, S. Wang, *On Computation of Arbitrage for Markets with Friction,* LNCS 1858, 310-319, 2000.

9. A. Fiat, A. V. Goldberg, J. D. Hartline, A. R. Karlin, *Competitive Generalized Auctions,* STOC 2002, 72-81.
10. A. Goldberg, J. Hartline, A. Wright, *Competitive Auctions and Digital Goods,* SODA 2001, 735-744.
11. T. Groves, *Incentive in Teams,* Econometrica, V.41, 617-631, 1973.
12. R. Lavi, N. Nisan, *Competitive Analysis of Incentive Compatible On-Line Auctions,* Theoretical Computer Science, V.310(1-3), 159-180, 2004.
13. D. Lehmann, L. I. O'Callaghan, Y. Shoham, *Truth Revelation in Approximately Efficient Combinatorial Auctions,* JACM, V.49(5), 577-602, 2002.
14. A. Mu'alem, N. Nisan, *Truthful Approximation Mechanisms for Restricted Combinatorial Auctions,* AAAI 2002, 379-384.
15. N. Megiddo, *Computational Complexity and the Game Theory Approach to Cost Allocation for a Tree,* Mathematics of Operations Research, V.3, 189-196, 1978.
16. C. H. Papadimitriou, *Algorithms, Games, and the Internet,* STOC 2001, 749-753.
17. C. H. Papadimitriou, M. Yannakakis, *On Complexity as Bounded Rationality (Extended Abstract),* STOC 1994, 726-733.
18. S. Smale, *Mathematical Problems for the Next Century,* Mathematical Intelligencer, V.20(2), 7-15, 1998.
19. W. Vickrey, *Counterspeculation, Auctions and Competitive Sealed Tenders,* Journal of Finance, V.16, 8-37, 1961.

The Complexity of Equivariant Unification

James Cheney

Cornell University
jcheney@cs.cornell.edu

Abstract. Nominal logic is a first-order theory of names and binding based on a primitive operation of *swapping* rather than substitution. Urban, Pitts, and Gabbay have developed a *nominal unification* algorithm that unifies terms up to nominal equality. However, because of nominal logic's *equivariance principle,* atomic formulas can be provably equivalent without being provably equal as terms, so resolution using nominal unification is sound but incomplete. For complete resolution, a more general form of unification called *equivariant unification,* or "unification up to a permutation" is required. Similarly, for rewrite rules expressed in nominal logic, a more general form of matching called *equivariant matching* is necessary.

In this paper, we study the complexity of the decision problem for equivariant unification and matching. We show that these problems are **NP**-complete in general. However, when one of the terms is essentially first-order, equivariant and nominal unification coincide. This shows that equivariant unification can be performed efficiently in many interesting common cases: for example, any purely first-order logic program or rewrite system can be run efficiently on nominal terms.

1 Introduction

Nominal logic [13] is a first-order theory of names and binding formalizing the novel Gabbay-Pitts approach to abstract syntax with binding inspired by Fraenkel-Mostowski permutation models of set theory [6]. In nominal logic, names are modeled as *atoms* a, b drawn from a countable set $\mathbb{A}$. Atoms can be tested for equality ('$a = b$') or freshness relative to other terms ('$a \# t$'), bound in abstractions ('$\langle a \rangle t$'), and used in swaps acting on terms ('$(a\ b) \cdot t$'). Nominal logic can serve as a foundation for specifying and reasoning about logics and programming languages encoded using nominal terms and relations; we call this approach to representing such languages *nominal abstract syntax.*

The state of the art of reasoning about languages with binding is *higher-order abstract syntax* [12] (HOAS), in which object-language variables and binders are encoded as meta-variables and λ-abstraction in a higher-order metalanguage. For example, in HOAS, an object-term $\lambda X.F\ X$ would be translated to a meta-language expression $lam(\lambda X.app\ F\ X)$, where $app : exp \to exp \to exp$ and $lam : (exp \to exp) \to exp$ are constants. In contrast, in nominal abstract syntax, variables and binders are translated to atoms $a \in \mathbb{A}$ and atom-abstractions $\langle a \rangle t \in \langle \mathbb{A} \rangle T$. For example, an object-term $\lambda X.F\ X$ is translated to

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 332–344, 2004.

$lam(\langle x\rangle app(var(f), var(x)))$, where $x, f : \mathbb{A}$, $var : \mathbb{A} \to exp$, $lam : \langle\mathbb{A}\rangle exp \to exp$, and app is as before. Abstractions are identified up to α-equivalence, e.g. $\langle a\rangle a$ and $\langle b\rangle b$ are considered equal terms.

Nominal logic is of interest because it may be much easier to reason about languages with binding using its first-order techniques than using higher-order techniques. For example, unification up to equality in nominal logic is efficiently decidable and unique most general unifiers (MGUs) exist [16], whereas unification up to equality in higher-order logic is undecidable and MGUs may not exist. However, higher-order unification is practically useful despite these theoretical drawbacks: Huet's semi-unification algorithm [9] performs well in practice, and higher-order unification is decidable in linear time and has unique MGUs for the broad special case of *higher-order patterns* [11]. A more serious problem is that reasoning by induction about languages with constructors like $lam : (\underline{exp} \to exp) \to exp$ is difficult because of the (underlined) negative occurrence of exp (see for example Hofmann [8] for a category-theoretic analysis of this problem). In contrast, there is no such negative occurrence in the nominal abstract syntax encoding $lam : \langle\mathbb{A}\rangle exp \to exp$, and induction principles can be derived directly from nominal language specifications (see [6,13]).

In this paper we consider a significant technical problem with automating reasoning in nominal logic. Nominal logic's *equivariance principle* states that the validity of an atomic formula is preserved by name-swapping operations:

$$p(\bar{X}) \Rightarrow p((a\ b) \cdot \bar{X})\ .$$

Since usually $p((a\ b) \cdot \bar{X}) \neq p(\bar{X})$, atomic formulas may differ as nominal terms but still be logically equivalent. For example, if $tc(g, e, t)$ is a predicate encoding simple typing for λ-terms (e has type t in type-context g), then equivariance alone (independent of the definition of tc) guarantees that

$$tc([], lam(\langle x\rangle var(x)), t) \iff tc([], lam(\langle y\rangle var(y)), t)$$
$$tc([(x,t)], app(var(y), var(x)), t') \iff tc([(y,t)], app(var(z), var(y)), t')$$

assuming no atoms appear in the types t, t'.

The *resolution principle* [14] is an important tool in automated deduction and logic programming. It states that given $A \vee P$ and $\neg B \vee Q$, where A, B are atomic and $A \Leftrightarrow B$, we can conclude $P \vee Q$. In first-order logic, atomic formulas are equivalent precisely when they are equal as first-order terms; moreover, we can decide whether and how two atomic formulas A, B can be instantiated to be logically equivalent simply by unifying them. Similarly, if terms (but not predicates) are higher-order, higher-order unification can be used for a complete resolution principle in automated deduction. For nominal logic, however, atomic formulas may be logically equivalent but not equal as nominal terms, so resolution based on nominal unification alone is incomplete. As a result, programs may behave incorrectly if nominal unification is used for backchaining in an implementation of nominal logic programming such as αProlog [2]. For example, given the following (nondeterministic) program:

$$bv(var(A),[]).\quad bv(lam(\langle a\rangle E),a :: L) :- bv(E,L)$$

the query $bv(lam(\langle x\rangle var(x)),[x])$ fails but should succeed.

Rewriting rules defined in nominal logic [4] are also subject to equivariance. For example, nominal rewrite rules such as

$$sub(var(a),a,T) \to T \quad sub(var(b),a,T) \to var(b)$$

define some cases for a substitution function. Here, T is a variable whereas $a, b \in \mathbb{A}$ are distinct *atom constants,* so the two rules do not overlap (as they would if a, b were variables that could be unified). To rewrite an expression like $sub(var(c),c,var(a))$ to $var(a)$, we must match the terms $sub(var(a),a,T)$ and $sub(var(c),c,var(a))$. These terms do not nominally match because the atoms c and a clash. However, by equivariance the first rule is still true if we apply the permutation $(a\ c)$ to it, yielding $sub(var(c),c,(a\ c)\cdot T) \to (a\ c)\cdot T$. This rule's left-hand side does match $sub(var(c),c,var(a))$ via substitution $[T := v(c)]$, so we can rewrite the term to $T = (a\ c)\cdot v(c) = v(a)$, as desired.

In order to obtain a complete resolution procedure, a new form of *equivariant unification* that unifies "up to a permutation" is required. Similarly, for nominal term rewriting rules involving atoms, an *equivariant matching* algorithm that matches a term to a ground term "up to a permutation" is needed. The aim of this paper is to study the complexity of the underlying decision problems of determining whether an equivariant unification or matching problem has a solution. In order to simplify matters, we consider only a special case, that for sequences of terms of sort $\mathbb{A}$. Despite its apparent simplicity, all the computational complexity resides in this case (though the details of the reduction from the general case are beyond the scope of this paper).

Our main results are that equivariant matching and satisfaction are both **NP**-complete. Thus, the situation is not as good as for first-order, nominal or higher-order pattern unification (all of which are in **P**), and in particular, equivariant unification cannot be reduced to nominal or higher-order pattern unification unless **P** = **NP**. However, in practice the situation may not be so bad. We identify an important special case which is in **P**: If two terms have no variables in common and one does not mention atoms or swaps, then equivariant unification reduces to nominal unification. This result can be generalized to show that ordinary first-order logic programs or rewrite rules can be applied to nominal terms efficiently using nominal unification.

2 Fundamentals

Due to space limits, we cannot provide a full introduction to nominal logic here; the interested reader is referred to Pitts [13].

We use the notation $\boldsymbol{x}$ for an *n-tuple* (or *sequence,* when n is not important) $(x_1,\ldots,x_n) \in X^n$ of elements of a set X.

Fix a countable set $\mathbb{A} = \{a_1, a_2, \ldots,\}$ of *atoms.* Recall that a (finitary) permutation of $\mathbb{A}$ is a bijection $\pi : \mathbb{A} \to \mathbb{A}$ that moves at most finitely many

elements of $\mathbb{A}$. The *support* of a permutation is the set of atoms it moves: $supp(\pi) = \{a \in \mathbb{A} \mid \pi(a) \neq a\}$; a permutation is finitary if and only if it has finite support. The *finitary permutation group over* $\mathbb{A}$, written $FSym(\mathbb{A})$, is the permutation group consisting of all finitary permutations of $\mathbb{A}$. Henceforth in this paper, all permutations are taken to be elements of $FSym(\mathbb{A})$, and we omit the adjective 'finitary'.

We write id for the identity permutation and write other permutations in transposition notation $(b_1\ c_1)\cdots(b_n\ c_n)$, where each $b_i, c_i \in \mathbb{A}$. In this notation, functional composition $\pi \circ \pi'$ is just the concatenation of π, π' as transposition lists. Permutations are equal when they denote the same function; equivalently, $\pi = \pi'$ when $supp(\pi \circ \pi'^{-1}) = \emptyset$. For example, $(a\ b) = (c\ d)(b\ a)(d\ c)$. We write $\pi \cdot_{\mathbb{A}} a$ for the result of applying π to a. For example $(a\ b) \cdot_{\mathbb{A}} a = b$ and $(a\ b) \cdot_{\mathbb{A}} c = c$ if $c \notin \{a, b\}$. Permutations act componentwise on sequences and sets of atoms: $\pi \cdot_{\mathbb{A}^n} (b_1, \ldots, b_n) = (\pi \cdot_{\mathbb{A}} b_1, \ldots, \pi \cdot_{\mathbb{A}} b_n)$, $\pi \cdot_{\mathcal{P}(\mathbb{A})} B = \{\pi \cdot_{\mathbb{A}} b \mid b \in B\}$. We omit the subscript on '$\cdot$' when there is no ambiguity.

One convenient property of $FSym(\mathbb{A})$ is that given a finite subset of $\mathbb{A}$, we can always find a disjoint finite subset of $\mathbb{A}$ (or a finite family of pairwise disjoint subsets) of the same size, together with permutations translating between them. The (easy) proof is omitted.

Proposition 1. *Suppose $B \subset \mathbb{A}$ is finite. Then there exists a permutation $\pi \in FSym(\mathbb{A})$ such that $\pi \cdot B$ and B are disjoint. More generally, if I is a finite index set, then there exists a family $(\tau_i \in FSym(\mathbb{A}) \mid i \in I)$ such that every pair of sets in $\{B\} \cup \{\tau_i \cdot B \mid i \in I\}$ is disjoint.*

Definition 1. *Two sequences $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{A}^n$ are* similar *(written $\boldsymbol{a} \sim \boldsymbol{b}$) if there is a permutation $\pi \in FSym(\mathbb{A})$ such that $\pi \cdot \boldsymbol{a} = \boldsymbol{b}$.*

Example 1. For example, $(a, b) \sim (b, c)$ as witnessed by $(a\ b)(b\ c)$, but $(a, a) \not\sim (b, d)$, and finally

$$(a, b, a, c, a, d) \sim (c, d, c, b, c, a)$$

as witnessed by $(a\ c)(c\ b)(b\ d)$.

Note that similarity is obviously an equivalence relation; in group-theoretic terms, its equivalence classes are *orbits* of $FSym(\mathbb{A})$ acting on $\mathbb{A}^n$. It is important to note that similarity is not a congruence with respect to pairing (or in general, composition of sequences): for example, $a \sim b$ and $a \sim c$ but $(a, a) \not\sim (b, c)$.

Let $\mathbb{V} = \{X, Y, \ldots\}$ be a countable set of variables. Variables are always capitalized. Terms s, t are either atoms $a \in \mathbb{A}$, or *suspensions* $\pi \cdot X$, where $\pi \in FSym(\mathbb{A})$ and $X \in \mathbb{V}$. The set of all terms is $\mathbb{T}$. The functions $V : \mathbb{T} \to \mathcal{P}(\mathbb{V})$ and $A : \mathbb{T} \to \mathcal{P}(\mathbb{A})$ calculate the sets of variables and atoms appearing in a term respectively. When the suspended permutation is id, the suspension $\mathsf{id} \cdot X$ is abbreviated to X. A term or sequence of terms is *ground* if no variables occur in it.

A *valuation* is a function $\theta : \mathbb{V} \to \mathbb{A}$. Valuations are extended to terms $\theta_{\mathbb{T}} : \mathbb{T} \to \mathbb{A}$ as follows:

$$\theta_{\mathbb{T}}(a) = a \qquad \theta_{\mathbb{T}}(\pi \cdot X) = \pi \cdot_{\mathbb{A}} \theta(X)$$

Suspended permutations come into effect after a valuation has been applied. Valuations operate componentwise on n-tuples: $\theta_{\mathbb{T}^n}(s_1, \ldots, s_n) = (\theta_{\mathbb{T}}(s_1), \ldots, \theta_{\mathbb{T}}(s_n))$. We omit the subscript on θ when there is no possibility of confusion. We write valuations using shorthand such as $\theta = [X := a, Y := b]$. For example

$$\theta(X, (a\ b) \cdot X, Y, (a\ c) \cdot Y) = (a, (a\ b) \cdot_{\mathbb{A}} a, b, (a\ c) \cdot_{\mathbb{A}} b) = (a, b, b, b)\ .$$

Definition 2. *An* equivariant satisfiability problem *is a pair* $(\boldsymbol{s}, \boldsymbol{t}) \in \mathbb{T}^n \times \mathbb{T}^n$ *(written* $\boldsymbol{s} \sim ?\ \boldsymbol{t}$*) for which it is desired to find a valuation* θ *such that* $\theta(\boldsymbol{s}) \sim \theta(\boldsymbol{t})$. *An* equivariant matching problem *is an equivariant satisfiability problem with* $\boldsymbol{t}$ *ground; then we write* $\boldsymbol{s} \lesssim ?\ \boldsymbol{t}$.

Example 2. Assume that different letters $a, b, a', b', \ldots$ refer to different atoms. The equivariant matching problem

$$(X, (a\ b) \cdot X, (a\ c) \cdot X) \sim ?\ (b', a', c')$$

has solution $[X := a]$. In fact, this is the only solution. On the other hand,

$$(a, X) \sim ?\ (c, d)$$

has infinitely many solutions $[X := b]$ for $b \neq a$, and neither of the following have any solutions:

$$(a, b) \sim ?\ (X, X) \qquad (X, (a\ b) \cdot X, Y, (a\ c) \cdot Y) \sim ?\ (a, b, c, d)$$

Definition 3. *A sequence* $\boldsymbol{a} \in \mathbb{A}^n$ *is* distinct *if no atom is repeated in it. For each* n*, the set of* distinct sequences of length n *is* $\mathbb{A}^{(n)} = \{\boldsymbol{a} \in \mathbb{A}^n \mid \boldsymbol{a}\ \textit{distinct}\}$. *A* separation problem $\boldsymbol{t} \in ?\ \mathbb{A}^{(n)}$ *is the problem of determining whether* $\boldsymbol{t} \lesssim ?\ \boldsymbol{a}$ *for some* $\boldsymbol{a} \in \mathbb{A}^{(n)}$.

Observe that $\boldsymbol{a}$ is distinct if and only if $i \mapsto a_i$ is injective; also, $\boldsymbol{t} \in ?\ \mathbb{A}^{(n)}$ if and only if $\boldsymbol{t} \lesssim ?\ \boldsymbol{a}$ for some fixed $\boldsymbol{a} \in \mathbb{A}^{(n)}$. Therefore, separation amounts to a very restricted case of equivariant matching.

Remark 1 (Atoms as Constants vs. Variables). Following Urban, Pitts, and Gabbay, our term language treats *atoms as constants:* we use concrete atom symbols as individual terms $\boldsymbol{a}$ and in permutations π. They are not subject to replacement by substitution, only to swapping. In contrast, Pitts' nominal logic treats *atoms as variables:* in fact, theories with atom constants are inconsistent because any atom is fresh for any constant, but no atom is fresh for itself. However, this is easy to fix: we are working on a revised version of nominal logic with a consistent treatment of atom-constants. This is however beyond the scope of this paper.

3 Complexity

We define the following decision problems:

$$\begin{aligned}
\text{EV} &= \{\boldsymbol{a} \sim \boldsymbol{b} \mid \boldsymbol{a}, \boldsymbol{b} \text{ ground}\} \\
\textsc{Sep} &= \{\boldsymbol{s} \in ?\ \mathbb{A}^{(n)} \mid \exists\theta.\theta(\boldsymbol{s}) \in \mathbb{A}^{(n)}\} \\
\textsc{EVMat} &= \{\boldsymbol{s} \lesssim ?\ \boldsymbol{b} \mid \boldsymbol{b} \text{ ground}, \exists\theta.\theta(\boldsymbol{s}) \sim \boldsymbol{b}\} \\
\textsc{EVSat} &= \{\boldsymbol{s} \sim ?\ \boldsymbol{t} \mid \exists\theta.\theta(\boldsymbol{s}) \sim \theta(\boldsymbol{t})\}
\end{aligned}$$

Note that $\textsc{Sep} \leq \textsc{EVMat} \leq \textsc{EVSat}$ by simple reductions. We now establish that EV is in **P**, and the rest of the problems are in **NP**.

For a ground sequence $\boldsymbol{a}$, let $E_{\boldsymbol{a}} = \{(i,j) \mid a_i = a_j\}$. That is, $E_{\boldsymbol{a}}$ is an equivalence relation whose equivalence classes are the indices of equal members of $\boldsymbol{a}$.

Proposition 2. *For ground sequences $\boldsymbol{a}, \boldsymbol{b}$ of equal length, $\boldsymbol{a} \sim \boldsymbol{b}$ if any only if $E_{\boldsymbol{a}} = E_{\boldsymbol{b}}$.*

Proof. If $\boldsymbol{a} \sim \boldsymbol{b}$, assume $\pi \cdot \boldsymbol{a} = \boldsymbol{b}$ and suppose $(i,j) \in E_{\boldsymbol{a}}$. Then $a_i = a_j$. So $b_i = \pi \cdot a_i = \pi \cdot a_j = b_j$. Hence $(i,j) \in E_{\boldsymbol{b}}$ and so $E_{\boldsymbol{a}} \subseteq E_{\boldsymbol{b}}$. A symmetric argument shows $E_{\boldsymbol{b}} \subseteq E_{\boldsymbol{a}}$, so the two sets are equal.

If $E_{\boldsymbol{a}} = E_{\boldsymbol{b}} = E$, note that the functions $f : i \mapsto a_i$ and $g : i \mapsto b_i$ are both constant on equivalence classes of E. Hence, the functions $f_E : [i]_E \mapsto a_i$ and $g_E : [i]_E \mapsto b_i$ are well-defined. Moreover, both are injective, since if $a_i = a_j$ then $[i]_E = [j]_E$ and similarly for $\boldsymbol{b}$; consequently the functions (considered on range $A(\boldsymbol{a})$ and $A(\boldsymbol{b})$ respectively) are invertible. Then the function $g \circ f_E^{-1} : A(\boldsymbol{a}) \to A(\boldsymbol{b})$ is also invertible. Any bijection between finite sets $B, C \subseteq \mathbb{A}$ can be extended to a permutation $\pi : \mathbb{A} \to \mathbb{A}$, so by choosing such an extension we have $\pi \cdot a_i = g_E \circ f_E^{-1}(a_i) = g_E([i]_E) = b_i$ for each i $(1 \leq i \leq n)$, so $\pi \cdot \boldsymbol{a} = \boldsymbol{b}$. QED.

The relations $E_{\boldsymbol{a}}, E_{\boldsymbol{b}}$ can obviously be represented as graphs which can be constructed from $\boldsymbol{a}, \boldsymbol{b}$ and compared in polynomial time.

Corollary 1. EV *is in* **P**.

Furthermore, the remaining four problems obviously have polynomial-time checkable certificates, namely minimal witnessing valuations θ.

Corollary 2. EVSAT, EVMAT, *and* SEP *are in* **NP**.

In the rest of this section we prove

Theorem 1. *The problem* SEP *is* **NP***-complete.*

Proof. Having already shown $\textsc{Sep} \in \mathbf{NP}$, we show **NP**-hardness only. We reduce from the **NP**-complete problem GRAPH 3-COLORABILITY, that is, determining whether a graph's vertices can be colored with one of three colors so that no neighboring vertices are the same color.

Let a (directed) graph $G = (V, E)$ with n vertices and m edges be given. We assume without loss of generality that $V = \{1, \ldots, n\}$ and $E = \{e_1, \ldots, e_m\}$. We write e^s, e^t for the source and target of the edge $e \in E$. Let $C = \{r, g, b\}$ be a three-element subset of $\mathbb{A}$. We define a 3-coloring as an n-tuple $\boldsymbol{c} \in C^n$ such that $c_{e^s} \neq c_{e^t}$ whenever $e \in E$.

Define $\pi_C = (r\ g)(g\ b)$, a cyclic permutation on $\mathbb{A}$ with support C. Choose (by Prop. 1) $n + m$ permutations $\tau_1, \ldots, \tau_n, \sigma_1, \ldots, \sigma_m$ so that if $T_i = \tau_i \cdot C$ for each $i \in \{1, \ldots, n\}$, and $S_j = \sigma_j \cdot C$ for each $j \in \{1, \ldots, m\}$, then all of the sets $C, T_1, \ldots, T_n$, and $S_1, \ldots, S_m$ are disjoint.

Let $X_1, \ldots, X_n \in \mathbb{V}$ be n distinct variables.

Idea of the proof. We will construct an instance of SEP such that for any solution θ, $\boldsymbol{c} = (\theta(X_1), \ldots, \theta(X_n))$ is a 3-coloring. To do this, we need to force all of the X_i to be elements of C and for each edge e force X_{e^s} and X_{e^t} to be different.

Observe $X \neq \pi_C \cdot X$ if and only if $X \in \mathit{supp}(\pi_C) = C$. So it is easy to encode a single set constraint $X \in C$ as a SEP problem

$$(X, \pi_C \cdot X) \in? \mathbb{A}^{(2)} .$$

However, for two variables this does not quite work:

$$(X_1, \pi_C \cdot X_1, X_2, \pi_C \cdot X_2) \in? \mathbb{A}^{(4)}$$

forces $X_1, X_2 \in C$ but also forces $X_1 \neq X_2$, $\pi_C \cdot X_1 \neq X_2$, etc. This is too strong. To prevent interference between subproblems, we isolate them using the permutations τ_1, τ_2:

$$(\tau_1 \cdot X_1, \tau_1 \circ \pi_C \cdot X_1, \tau_2 \cdot X_2, \tau_2 \circ \pi_C \cdot X_2) \in? \mathbb{A}^{(4)}$$

First note that $\tau_1 \cdot X_1 \neq \tau_1 \circ \pi_C \cdot X_1$ implies $X_1 \neq \pi_C \cdot X_1$ so $X_1 \in C$ and similarly $X_2 \in C$, as before. On the other hand, if X_1, X_2 are in C, then all four components are different, since the first two lie in T_1 and the last two in T_2, and the two sets are disjoint. It is not hard to show by induction that

$$s = (\tau_1 \cdot X_1, \tau_1 \circ \pi_C \cdot X_1, \ldots, \tau_n \cdot X_n, \tau_n \circ \pi_C \cdot X_n) \in? \mathbb{A}^{(2n)}$$

is in SEP if and only if $X_1, \ldots, X_n \in C$.

Now we need to enforce that whenever $e \in E$, we have $X_{e^s} \neq X_{e^t}$. For a single edge, the following instance suffices:

$$(X_{e^s}, X_{e^t}) \in? \mathbb{A}^{(2)}$$

However, as was the case earlier, problems cannot always be combined correctly because they might interfere. For example, for two edges (1,2), (1,3), the problem

$$(X_1, X_2, X_1, X_3) \in? \mathbb{A}^{(4)}$$

is unsatisfiable because the value of X_1 is repeated in any valuation, but $[X_1 := r, X_2 := g, X_3 := b]$ is a 3-coloring. To get around this problem, we use the permutations σ_i to isolate the constraints for each edge e_i. For example,

$$(\sigma_1 \cdot X_1, \sigma_1 \cdot X_2, \sigma_2 \cdot X_1, \sigma_2 \cdot X_3) \in? \mathbb{A}^{(4)}$$

ensures $X_1 \neq X_2$ and $X_1 \neq X_3$. Also, if $X_1, X_2, X_3 \in C$ then the first two components are in S_1 and the second two in S_2, and $S_1 \cap S_2 = \emptyset$. So more generally, the problem

$$\boldsymbol{t} = (\sigma_1 \cdot X_{e_1^s}, \sigma_1 \cdot X_{e_1^t}, \ldots, \sigma_m \cdot X_{e_m^s}, \sigma_m \cdot X_{e_m^t}) \in? \mathbb{A}^{(2m)}$$

enforces the coloring property for each edge and permits all valid colorings.

Define $\boldsymbol{u}$ to be the $2n+2m$-tuple *st.* Then $\boldsymbol{u} \in? \mathbb{A}^{(2n+2m)}$ is the SEP problem corresponding to the instance G of GRAPH 3-COLORABILITY.

Correctness of the reduction. So far we have only described the construction and the intuition behind it. It is easy to see that the size of $\boldsymbol{u}$ is $O(m+n)$, since π_C, τ_i, and σ_j each have representations consisting of at most three transpositions. We now show carefully that the reduction is correct, that is, G has a 3-coloring $\boldsymbol{c} \in C^n$ if and only if $\boldsymbol{u}$ has a separating valuation θ. The backward direction is easy, since (as outlined above) it is easy to show that any solution θ separating $\boldsymbol{u} = \boldsymbol{st}$ corresponds to a 3-coloring $c_i = \theta(X_i)$.

The difficulty is showing that $\boldsymbol{u}$ is not over-constrained: that is, if $\boldsymbol{c}$ is a 3-coloring then the valuation $\theta(X_i) = c_i$ separates $\boldsymbol{u}$. Suppose $\boldsymbol{c}$ is a 3-coloring and $\theta(X_i) = c_i$. We need to show that $i \neq j$ implies $\theta(u_i) \neq \theta(u_j)$ for each $i, j \in \{1, \ldots, |\boldsymbol{u}|\}$. Assume $i, j \in \{1, \ldots, |\boldsymbol{u}|\}$ and $i \neq j$. Suppose without loss of generality that $i < j$. There are three cases.

If i is even or $j > i+1$, then $u_i = \rho \cdot X_k$ and $u_j = \rho' \cdot X_{k'}$ for some permutations ρ, ρ' and $X_k, X_{k'}$, and $\rho \cdot C$ and $\rho' \cdot C$ are disjoint, so

$$\theta(u_i) = \rho \cdot c_k \neq \rho' \cdot c_{k'} = \theta(u_j)$$

If i is odd and $i+1 = j$ and $j \leq 2n$, then j is even; set $k = j/2$. Then $u_i = \tau_k \cdot X_k, u_j = \tau_k \circ \pi_C \cdot X_k$, and we have

$$\theta(u_i) = \tau_k \cdot c_k \neq \tau_k \circ \pi_C \cdot c_k = \theta(u_j)$$

since $\pi_C \cdot c_k \neq c_k$.

If i is odd and $j = i+1$ and $2n+1 \leq i$, then j and $j-2n$ are even; set $k = (j-2n)/2$. Then $u_i = \sigma_k \cdot X_{e_k^s}$, $u_j = \sigma_k \cdot X_{e_k^t}$, and

$$\theta(u_i) = \sigma_k \cdot c_{e_k^s} \neq \sigma_k \cdot c_{e_k^t} = \theta(u_j)$$

where $c_{e_k^s} \neq c_{e_k^t}$ since $\boldsymbol{c}$ is a 3-coloring. So, in any case, $\theta(u_i) \neq \theta(u_j)$. QED.

Corollary 3. EVMAT *and* EVSAT *are* **NP**-*complete.*

4 Tractable Special Cases

There are several special cases of equivariant satisfiability or matching that are tractable. We present a one such special case, a simple syntactic restriction that guarantees that equivariant satisfiability can be reduced to nominal unification. We describe some additional special cases at the end of this section.

To discuss nominal unification, we first need to extend permutation action to terms and define substitutions and renamings. Permutations act on terms as follows:

$$\pi \cdot_{\mathbb{T}} (a) = \pi \cdot_{\mathbb{A}} a \qquad \pi \cdot_{\mathbb{T}} (\pi' \cdot X) = (\pi \circ \pi') \cdot X$$

and act componentwise on sequences of terms. A substitution is a function $\sigma : \mathbb{V} \to \mathbb{T}$ from variables to terms, extended as follows to $\sigma_{\mathbb{T}} : \mathbb{T} \to \mathbb{T}$:

$$\sigma_{\mathbb{T}}(a) = a \qquad \sigma_{\mathbb{T}}(\pi \cdot X) = \pi \cdot_{\mathbb{T}} \sigma(X)$$

and extended componentwise to $\sigma_{\mathbb{T}^n} : \mathbb{T}^n \to \mathbb{T}^n$. Note that substitutions may activate delayed permutation actions:

$$((a\ b) \cdot X, (a\ c) \cdot Y)[X := a, Y := (b\ c) \cdot Z] = (b, (a\ c)(b\ c) \cdot Z)\ .$$

Moreover, note that $\pi \cdot (\sigma(x)) = \sigma(\pi \cdot x)$, for x a term or sequence.

A term s (or sequence $\boldsymbol{s}$) is a *renaming* of another term t (sequence $\boldsymbol{t}$) if $s = \rho(t)$ (or $\boldsymbol{s} = \rho(\boldsymbol{t})$) for some *invertible* substitution ρ. Note that invertible substitutions may involve swapping: for example, $[X := \pi \cdot Y, Y := X]$ has inverse $[X := Y, Y := \pi^{-1} \cdot X]$. Two terms s, t (or sequences $\boldsymbol{s}, \boldsymbol{t}$) *unify* if there is an idempotent substitution σ such that $\sigma(s) = \sigma(t)$ (or $\sigma(\boldsymbol{s}) = \sigma(\boldsymbol{t})$). For example, $(a\ b) \cdot X$ unifies with $(b\ c) \cdot X$ with substitution $[X := d]$, for any $d \notin \{a, b, c\}$. The algorithm of Urban et al. decides a more general case of nominal unification, and finds unique MGUs (up to renaming) when they exist. Although their algorithm is not polynomial time as presented, a polynomial-time algorithm can be obtained by modifying the quadratic unification algorithm of Martelli and Montanari [10]; further improvements may be possible.

We say $\boldsymbol{s}$ is *pure* if no atoms appear in $\boldsymbol{s}$: that is, $\boldsymbol{s}$ is a list of variables with suspended permutation id. We say $\boldsymbol{s}$ is *semi-pure* if it is a renaming of a pure $\boldsymbol{s}'$. For example, (X, Y, X) is pure and $((a\ b) \cdot X, Y, (c\ a)(c\ b)(c\ a) \cdot X)$ is semi-pure. We say $\boldsymbol{s}, \boldsymbol{t}$ are *variable-disjoint* when $V(\boldsymbol{s}) \cap V(\boldsymbol{t}) = \emptyset$.

Theorem 2. *If $\boldsymbol{s}$ is semi-pure and $\boldsymbol{s}, \boldsymbol{t}$ are variable-disjoint, then $\boldsymbol{s} \sim ?\ \boldsymbol{t}$ can be decided in polynomial time.*

Proof. We show this in two steps. First, assuming $\boldsymbol{s}$ is pure, we show that deciding $\boldsymbol{s} \sim ?\ \boldsymbol{t}$ reduces to nominal unification. Second, we show that if $\boldsymbol{s}$ is semi-pure and $\boldsymbol{s}'$ is a pure renaming of $\boldsymbol{s}$, then $\boldsymbol{s} \sim ?\ \boldsymbol{t}$ is satisfiable if and only if $\boldsymbol{s}' \sim ?\ \boldsymbol{t}$ is.

For the first part, if $\boldsymbol{s}$ and $\boldsymbol{t}$ have a nominal unifier, note that any unifier has a ground instance, any instance of a unifier is also a unifier, and any ground substitution is a valuation. So we can find a valuation θ such that $\theta(\boldsymbol{s}) = \theta(\boldsymbol{t})$; hence, $\mathsf{id} \cdot \theta(\boldsymbol{s}) = \theta(\boldsymbol{t})$ so $\theta(\boldsymbol{s}) \sim \theta(\boldsymbol{t})$. Conversely, suppose that $\pi \cdot \theta(\boldsymbol{s}) = \theta(\boldsymbol{t})$. Let θ' be defined as follows:

$$\theta'(X) = \begin{cases} \pi \cdot \theta(X) : X \in V(\boldsymbol{s}) \\ \theta(X) \quad : \text{otherwise} \end{cases}$$

Since $\boldsymbol{s}, \boldsymbol{t}$ are variable-disjoint, θ' agrees with θ on $V(\boldsymbol{t})$ so $\theta(\boldsymbol{t}) = \theta'(\boldsymbol{t})$. Also, since $\boldsymbol{s}$ is pure, we know $\boldsymbol{s} = (X_1, \ldots, X_n)$ for $\{X_1, \ldots, X_n\} = V(\boldsymbol{s})$ (where some

of the X_i may be repeated). Hence

$$\theta'(\boldsymbol{s}) = (\theta'(X_1), \ldots, \theta'(X_n)) = (\pi \cdot \theta(X_1), \ldots, \pi \cdot \theta(X_n))$$
$$= \pi \cdot \theta(\boldsymbol{s}) = \theta(\boldsymbol{t}) = \theta'(\boldsymbol{t})$$

So $\theta'(\boldsymbol{s}) = \theta'(\boldsymbol{t})$ and θ' is a nominal unifier of $\boldsymbol{s}, \boldsymbol{t}$. The existence of a nominal unifier can be decided in polynomial time by nominal unification.

For the second part, note that since $\boldsymbol{s}$ is semi-pure, there exists a pure $\boldsymbol{s}'$ and invertible ρ such that $\rho(\boldsymbol{s}) = \boldsymbol{s}'$. Since $\boldsymbol{s}, \boldsymbol{t}$ are variable-disjoint, we may choose $\boldsymbol{s}', \rho$ such that $\boldsymbol{s}', \boldsymbol{t}$ are also variable-disjoint and ρ fixes all the variables $V(\boldsymbol{t})$ of $\boldsymbol{t}$. Since $\rho(X) = X$ whenever $X \in V(\boldsymbol{t})$, we also have $\rho(\boldsymbol{t}) = \boldsymbol{t}$. We will show that $\boldsymbol{s}' \sim? \boldsymbol{t}$ is satisfiable if and only if $\boldsymbol{s} \sim? \boldsymbol{t}$ is; since the former can be decided efficiently, so can the latter. Assume $\boldsymbol{s}' \sim? \boldsymbol{t}$ is satisfiable, and suppose $\pi \cdot \theta(\boldsymbol{s}') = \theta(\boldsymbol{t})$. Let $\theta' = \theta \circ \rho$. Then

$$\pi \cdot \theta'(\boldsymbol{s}) = \pi \cdot \theta \circ \rho(\boldsymbol{s}) = \pi \cdot \theta(\boldsymbol{s}') = \theta(\boldsymbol{t}) = \theta \circ \rho(\boldsymbol{t}) = \theta'(\boldsymbol{t})$$

so $\boldsymbol{s} \sim? \boldsymbol{t}$ has a solution. A symmetric argument (using the equation $\rho^{-1}(\boldsymbol{s}') = \boldsymbol{s}$) shows that if $\boldsymbol{s} \sim? \boldsymbol{t}$ has a solution then so does $\boldsymbol{s}' \sim? \boldsymbol{t}$. QED.

Remark 2. Theorem 2 can be generalized to unification over full nominal terms, in which case pure terms are simply first-order terms with no atoms, abstractions, or swaps. Suppose we have a purely first-order logic program P (i.e., a set of first-order Horn clauses). Since the variables of program clauses are always freshened prior to attempting resolution, resolution behaves the same using equivariant unification as nominal unification, so for atomic A, $P \vdash A$ can be derived using equivariant unification if any only if $P \vdash A$ can also be derived using nominal unification. Similarly, suppose we have a purely first-order term rewriting system R. Then $s \rightarrow_R t$ using equivariant matching if and only if $s \rightarrow_R t$ using nominal matching. These results can be generalized to permit program clauses with semi-pure heads and unrestricted bodies, and rewriting rules with semi-pure left-hand sides and arbitrary right-hand sides. So broad classes of nominal logic programs and rewrite systems (including all first-order logic programs and rewrite systems) can be executed efficiently without sacrificing completeness.

Remark 3. There are other known tractable special cases, but they depend on aspects of nominal logic beyond the scope of this paper. Urban and Cheney [15] have identified a broad class of proper αProlog programs (i.e., programs that are not equivalent to first-order logic programs) for which full equivariant unification is unnecessary and nominal unification can be used instead. Also, equivariant matching problems involving nominal terms *in which the swapping operation does not appear* seem to admit efficient solutions.

5 Related and Future Work

Permutations of variables arise in natural ways in first-order and higher-order pattern unification. In first-order unification, any two MGUs for a given problem

are equivalent up to permutative renamings of variables. In higher-order unification, the cases that cause problems involve free variables applied to arbitrary lists of bound and free variables, and this case is avoided by the higher-order pattern restriction that free variables are only applied to lists of *distinct* bound variables [11]. Consequently, whenever two subterms $X\ x_1 \cdots x_n$, $Y\ y_1 \cdots y_m$ are to be unified (where X, Y are free and $\bar{x}, \bar{y}$ are bound variables), there is always a partial permutation relating the variables $\bar{x}$ and $\bar{y}$. Then all the nondeterministic choices in Huet's full higher-order semi-unification algorithm can be avoided; unification can be performed efficiently, and MGUs are unique when they exist.

An alternative view of equivariant satisfiability to the one taken in this paper is as the search for a solution for the equation $P \cdot s(\bar{X}) = t(\bar{X})$ (over a permutation variable P and atom variables $\bar{X}$). In light of this fact, prior work on satisfiability for equations over groups may be relevant to equivariant unification. Many mathematicians from Frobenius onward have studied the problem of solving (and counting the solutions to) specific group equations such as $P^n = \mathsf{id}$ [5]. Albert and Lawrence studied elementary unification in varieties of nilpotent groups [1]. They showed that MGUs may not exist in that setting, but are unique when they do, and described a polynomial time algorithm that computes a MGU or determines that none exists for a specific problem. Goldmann and Russell [7] showed that for finite groups, solving systems of equations (possibly involving constants) is polynomial time if the group is Abelian, otherwise **NP**-complete. They also showed that solving a single group equation is **NP**-complete if the group is non-solvable and in **P** if it is nilpotent; the complexity of solvable but non-nilpotent groups is not settled. Engebretsen et al. [3] showed that approximating the number of solutions to a single group equation to within $|G| - \epsilon$ is **NP**-hard for any $\epsilon > 0$.

Our first proof of **NP**-completeness for equivariant satisfiability reduced from Goldmann and Russell's single-equation group satisfiability problem for non-solvable groups (since full finite symmetric groups are not solvable in general). That approach required several intermediate reductions and showed only to the weaker result that EVSAT is **NP**-complete, leaving the complexity of equivariant matching unresolved. Except for Goldmann and Russell's work, we have not found any applications of the above research on unification and satisfaction for group equations to equivariant unification.

There are two immediate directions for future work. First, we are developing practical algorithms for equivariant matching and unification for use in resolution and term rewriting in αProlog, a logic programming language based on nominal logic [2]. Second, in this paper we asserted without proof that equivariant unification is necessary and sufficient for complete nominal resolution. Though this seems clear, it requires proof. We plan to present practical equivariant unification and matching algorithms and establish the soundness and completeness of equivariant unification for nominal resolution (at least in the context of logic programming) in future work.

6 Conclusions

Equivariant satisfiability and matching, or deciding whether two terms involving swapping can be made equal "up to a permutation", are important decision problems for automated reasoning, logic programming, and term rewriting in nominal logic. We have shown that both are **NP**-complete. We have also found an interesting tractable special case, for which nominal unification suffices. Consequently, first-order logic programs and term rewriting systems can be run efficiently on nominal terms. Only those programs or rewrite systems that actually use the novel features of nominal logic need pay for them. Determining the impact of these **NP**-completeness results on practical automated deduction, logic programming, and rewriting in nominal logic is important future work.

subsubssection* Acknowledgements. I am grateful to Andrew Pitts, Christian Urban, and the anonymous referees for helpful comments on earlier versions of this paper.

References

1. Michael H. Albert and John Lawrence. Unification in varieties of groups: nilpotent varieties. *Canadian Journal of Mathematics,* 46(6): 1135–1149, 1994.
2. J. Cheney and C. Urban. Alpha-Prolog: A logic programming language with names, binding and alpha-equivalence. In *Proceedings of the 20th International Conference on Logic Programming (ICLP 2004),* 2004. To appear.
3. Lars Engebretsen, Jonas Holmerin, and Alexander Russell. Inapproximability results for equations over finite groups. *Theoretical Computer Science,* 312(1):17–45, 2004.
4. Maribel Fernández, Murdoch Gabbay, and Ian Mackie. Nominal rewriting. Submitted, January 2004.
5. H. Finkelstein. Solving equations in groups: a survey of Frobenius' Theorem. *Periodica Mathematica Hungarica,* 9(3): 187–204, 1978.
6. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing,* 13:341–363, 2002.
7. Mikael Goldmann and Alexander Russell. The complexity of solving equations over finite groups. *Information and Computation,* 178:253–262, 2002.
8. Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *Proc. 14th Symp. on Logic in Computer Science,* pages 204–213. IEEE, July 1999.
9. Gerard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science,* 1:27–67, 1975.
10. A. Martelli and U. Montanari. An efficient unification algorithm. *Transactions on Programming Languages and Systems,* 4(2):258–282, 1982.
11. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Logic and Computation,* l(4):497–536, 1991.
12. Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI '89),* pages 199–208. ACM Press, 1989.
13. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation,* 183:165–193, 2003.

14. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM,* 12(1):23–41, 1965.
15. C. Urban and J. Cheney. Avoiding equivariance in Alpha-Prolog. Submitted, 2004.
16. C. Urban, A. M. Pitts, and M. J. Gabbay. Nominal unification. In M. Baaz, editor, *Computer Science Logic and 8th Kurt Gödel Colloquium (CSL'03 & KGC),* volume 2803 of *Lecture Notes in Computer Science,* pages 513–527, Vienna, Austria, 2003. Springer-Verlag.

Coordination Mechanisms*

George Christodoulou[1], Elias Koutsoupias[1], and Akash Nanavati[2]

[1] Department of Informatics, University of Athens,
Panepistimiopolis Ilisia, Athens 15784, Greece.
{gchristo,elias}@di.uoa.gr
[2] Computer Science Department, University of California Los Angeles,
Los Angeles, CA 90095, USA.
akash@cs.ucla.edu

Abstract. We introduce the notion of coordination mechanisms to improve the performance in systems with independent selfish and non-colluding agents. The quality of a coordination mechanism is measured by its price of anarchy—the worst-case performance of a Nash equilibrium over the (centrally controlled) social optimum. We give upper and lower bounds for the price of anarchy for selfish task allocation and congestion games.

1 Introduction

The price of anarchy [11,18] measures the deterioration in performance of systems on which resources are allocated by selfish agents. It captures the lack of coordination between independent selfish agents as opposed to the lack of information (competitive ratio) or the lack of computational resources (approximation ratio). However unlike the competitive and approximation ratios, the price of anarchy failed to suggest a framework in which coordination algorithms for selfish agents should be designed and evaluated.

In this work we attempt to remedy the situation. We propose a framework to study some of these problems and define the notion of coordination mechanisms (the parallel of online or approximation algorithms) which attempt to redesign the system to reduce price of anarchy. To introduce the issues, we consider first two different situations from which the notion of coordination mechanisms emerges in a natural way.

Consider first the selfish task allocation problem studied in [11]. There is a simple network of m parallel links or m identical machines and a set of n selfish users. Each user i has some load w_i and wants to schedule it on one of the machines. When the users act selfishly at a Nash equilibrium the resulting allocation may be suboptimal. The price of anarchy, that is, the worst-case ratio of the maximum latency at a Nash equilibrium over the optimal allocation can be as high as $\Theta(\log m/\log\log m)$ [11,5,10]. The question is "*How can we improve*

* Research supported in part by the IST (FLAGS, IST-2001-33116) program and by NSF.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 345–357, 2004.

the price of anarchy?"; and what mechanisms one can use to improve the overall system performance even in the face of selfish behavior? We will assume that the system designer can select the scheduling policies of each machine; we then ask whether some scheduling policies can reduce the price of anarchy and by how much. An important aspect of the problem is that the designer must design the system once and for all, or equivalently that *the scheduling policies should be defined before the set of loads* is known. Another important and natural condition is the decentralized nature of the problem: *the scheduling on a machine should depend only on the loads assigned to it* and should be independent of the loads assigned to other machines (otherwise an optimal allocation can be easily enforced by a centralized authority and all game-theoretic issues vanish). This framework is very similar to competitive analysis, especially if we consider the worst-case price of anarchy: *We, the designers, select the scheduling policies for each machine. Then an adversary selects a set of loads. We then compute the makespan of the worst Nash equilibrium and divide by the makespan of the optimal allocation.* It is important to clarify that we divide with the absolute (original) optimum which is independent of our choice of scheduling policies.

As a second example, consider the selfish routing problem whose price of anarchy was studied by Roughgarden and Tardos [23]. In a network in which the latency experienced by the traffic on an edge depends on the traffic traversing the edge, selfish users route traffic on minimum-latency paths. The price of anarchy can be as high as 4/3 for linear latency functions and unbounded for arbitrary latency functions. How can we improve the price of anarchy in this situation? For the famous Braess' paradox case, a simple solution is to remove some edges. The removal of edges however does not improve the price of anarchy in general; even for the Braess' paradox network, the removal of an edge can make the situation much worse for other amounts of traffic. We propose to study mechanisms that slow down the traffic on some edges to improve the performance. More precisely, we, the designers select for each edge e a new latency function $\hat{c}^e$ which is equal or greater than the original latency function c^e; then the adversary selects a flow and we evaluate the price of anarchy. Notice that, as in the case of the selfish task allocation, we should divide the Nash equilibrium latency (computed using the new latency functions $\hat{c}^e$) by the optimal latency (of the original latency functions c^e).

1.1 Our Contributions

To study the above and similar problems, we introduce a unifying framework: the notion of *coordination models* which is an appropriate generalization of congestion games and the notion of *coordination mechanisms* which generalizes the scheduling policies and the increase in the cost and latency functions of the above examples.

Using this framework, we study the selfish task allocation problem (Section 3). We give a coordination mechanism (i.e., scheduling policies) with price of anarchy $4/3 - 1/(3m)$, improving significantly over the original $\Theta(\log m / \log \log m)$. We conjecture that this bound is tight, but we were able

to show only that every coordination mechanism has price of anarchy strictly greater than 1 (this still allows the infimum price of anarchy to be 1).

We also study coordination mechanisms for congestion games (Section 4). We show an interesting relation between the potential and the social cost of a set of strategies; based on these we give a coordination mechanism with price of anarchy n for the single-commodity congestion games. We also show that the bound n is tight. We conjecture that the same bound holds for the general congestion games; but we were able to show only that the coordination mechanism that we employed for the single commodity games fails in the general case (details in the full version).

Finally, for the case of selfish routing, non-continuous coordination mechanisms may perform arbitrarily better than continuous ones; this asks for removing the assumptions of continuity in the work of Roughgarden and Tardos [23]. We have positive results only for very special cases of small networks (details in the full version).

1.2 Related Work

Mechanisms to improve coordination of selfish agents is not a new idea and we only mention here work that directly relates to our approach. A central topic in game theory [17] is the notion of mechanism design in which the players are paid (or penalized) to "coordinate". The differences between mechanism design and the coordination mechanism model are numerous. The most straightforward comparison can be exhibited in the selfish routing problem: both aim at improving coordination, but mechanism design can be seen as a way to introduce *tolls* (see for example [2,3]), while coordination mechanism is a way to introduce *traffic lights*. Also, the algorithmic and communication issues involved in mechanism design seem to be completely different than the ones involved in coordination mechanisms [16,15,19,1].

The idea of designing games to improve coordination appears also in the work of Korilis, Lazar, and Orda [9] but there the goal is to design games with a unique Nash equilibrium; there is no attempt to compare it with the potential optimum.

In an attempt to reduce total delay at Nash equilibrium in the selfish routing problem, [2,3] analyzes the problem of assigning *taxes* on network edges. Also, [14] analyzes how much total money one has to spend in order to influence the outcome of the game, when the interested party gives payments to agents on certain outcomes.

A problem that relates to coordination mechanisms for selfish routing, and studied in [21], asks to find a subnetwork of a given network that has optimal price of anarchy for a given total flow. This can be also cast as a special case of coordination mechanisms that allow either a given specific delay function or infinity (and fixed total flow).

2 The Model

Congestion games [20,13,6], introduced by Rosenthal, is an important class of games that capture many aspects of selfish behavior in networks. A congestion game is defined by a tuple $(N, M, (\Sigma_i)_{i\in N}, (c^j)_{j\in M})$ where N is the set of players, M is the set of facilities, Σ_i is a collection of strategies for player i, and c^j is the cost (delay) function of facility j. The characterizing property of congestion games is that the cost of players for using facility j is the same for all players and depends only on the number of players using the facility: when k players use facility j, the cost of each player for using the facility is $c^j(k)$. The total cost of each player is the sum of the individual cost of each facility used by the player.

There are three important classes of congestion games: the single-commodity, the multi-commodity, and the general congestion games. In the most restricted class, the single-commodity congestion game, there are n selfish players that want to establish a path from a fixed node s to a fixed destination t. The facilities are the edges of the network and the strategies for each player are the paths from s to t. In the more general class of multi-commodity games, each player may have its own source and destination. Finally, in the most general class there is no network. It is well-known that every congestion game has at least one pure Nash equilibrium.

To define the price of anarchy of a congestion game, we need first to agree on the social cost (i.e., the system cost) of a set of strategies. Two natural choices are the maximum or the average cost per player —the first one was used in the selfish task allocation problem of [11] and corresponds to the makespan, and the second one was used in the selfish routing problem in [23]. The price of anarchy is then defined as the worst-case ratio, among all Nash equilibria, over the optimal social cost, among all possible set of strategies.

One can generalize congestion games in two directions: First, to allow the players to have loads or weights and second, to allow asymmetric cost functions where players experience different cost for using a facility [12]. These generalizations are realized by cost functions c_i^j, one for each player —the cost of player i for using facility j is now $c_i^j(w^j)$ where w^j is the sum of weights of the players using facility j.

How can we improve the price of anarchy of congestion games? There are two simple ways: First, by introducing delays, and second, by distinguishing between players and assigning priorities to them. Given a generalized congestion game $(N, M, (\Sigma_i)_{i\in N}, (c_i^j)_{j\in M, i\in N})$, we shall define the set of all possible games that result when we add delays and priorities; we will call these games coordination mechanisms. The introduction of delays is straightforward: the set of allowed games have cost functions $\hat{c}_i^j$ where $\hat{c}_i^j(w) \geq c_i^j(w)$. We will call these *symmetric coordination mechanisms.* The way to introduce priorities is less obvious but we can approach the problem as follows: Let facility j assign priorities to players so that it services first player t_1, then player t_2 and so on. The cost (delay) of the first player t_1 cannot be less than $c_{t_1}^j(w_{t_1})$, the cost of using the facility itself. Similarly, the cost of the k-th player t_k cannot be less than $c_{t_k}^j(w_{t_1} + \cdots + w_{t_k})$.

The natural problem is to select a coordination mechanism with small price of anarchy among all those coordination mechanisms with delays and priorities. To define this problem precisely and generalize the above discussion, we introduce the notion of coordination model in the next subsection.

2.1 Coordination Models

A Coordination Model is a tuple $(N, M, (\Sigma_i)_{i\in N}, (C^j)_{j\in M})$ where $N = \{1, \ldots, n\}$ is the set of players, M is a set of facilities, Σ_i is a collection of strategies for player i: a strategy $A_i \in \Sigma_i$ is a set of facilities, and finally C^j is a collection of cost functions associated with facility j: a cost function $c^j \in C^j$ is a function that takes as input a set of loads, one for each player that uses the facility, and outputs a cost to each participating player. More precisely, c^j is a cost function from R^N to R^N. A natural property is that $c^j_i(w_1, \ldots, w_{i-1}, 0, w_{i+1}, \ldots, w_n) = 0$ which expresses exactly the property that players incur no cost when they don't use the facility.

In most coordination models, the strategies and cost functions are defined implicitly; for example, by introducing delays and priorities to a given congestion game. We remark however that the congestion model corresponds to a particular game —there is only one cost function for each facility— while in our model there is a collection of games —a set of cost functions for each facility.

Example 1. The coordination model for **selfish task allocation** that corresponds to the problem studied in [11] is as follows: $N = \{1, \ldots, n\}$ is the set of players, $M = \{1, \ldots, m\}$ the set of facilities is a set of machines or links, all Σ_i's consists of all singleton subsets of M, $\Sigma_i = \{\{1\}, \ldots, \{m\}\}$, i.e., each player uses exactly one facility, and the cost functions are the possible finish times for scheduling the loads on a facility. More precisely, a function c^j is a cost function for facility j if for every set of loads $(w_1, \ldots, w_n)$ and every subset S of N, the maximum finish time of the players in S must be at least equal to the total length of the loads in S: $\max_{i\in S} c^j_i(w_1, \ldots, w_n) \geq \sum_{i\in S} w_i$. Notice that a facility is allowed to order the loads arbitrarily and introduce delays, but it cannot speed up the execution. As an example, a facility could schedule two loads w_1 and w_2 so that the first load finishes at time $w_1 + w_2/2$ and the second load at time $2w_1 + w_2$.

2.2 Coordination Mechanisms

The notion of coordination model defined in the previous subsection sets the stage for an adversarial analysis of the deterioration in performance due to lack of coordination. The situation is best understood when we compare it with competitive analysis. The following table shows the correspondence.

Coordination model	↔ Online problem
Coordination mechanism	↔ Online algorithm
Price of anarchy	↔ Competitive ratio

It should be apparent from this correspondence that one cannot expect to obtain meaningful results for *every possible coordination model* in the same way that we don't expect to be able to find a unifying analysis of *every possible online problem.* Each particular coordination model that arises in "practice" or in "theory" should be analyzed alone. We now proceed to define the notion of coordination mechanism and its price of anarchy.

A *coordination mechanism* for a coordination model $(N, M, (\Sigma_i)_{i\in N}, (c^j)_{j\in M})$ is simply a set of cost functions, one for each facility. The simplicity of this definition may be misleading unless we take into account that the set of cost functions may be very rich. A coordination mechanism is essentially a *decentralized algorithm;* we select once and for all the cost functions for each facility, before the input is known. For example, for the coordination model for selfish task allocation, a coordination mechanism is essentially a set of *local scheduling policies,* one for each machine; the scheduling on each machine depends only on the loads that use the machine. Fix a coordination mechanism $c = (c^1, \ldots, c^m)$, a set of player loads $w = (w_1, \ldots, w_n)$, and a set of strategies $A = (A_1, \ldots, A_n) \in \Sigma_1 \times \cdots \times \Sigma_n$. Let $(\text{cost}_1, \ldots, \text{cost}_n)$ denote the cost incurred by the players. We define the *social cost* $\text{sc}(w; c; A)$ as the maximum (or sometimes the sum) cost among the players, i.e., $\text{sc}(w; c; A) = \max_{i\in N} \text{cost}_i$.

We also define the *social optimum* $opt(w)$ for a given set of player loads w as the minimum social cost of all coordination mechanisms and all strategies in $\Sigma_1 \times \cdots \times \Sigma_n$, i.e., $opt(w) = \inf_{c,A} \text{sc}(w; c; A)$.

It is important to notice that the definition of $opt(w)$ refers to the absolute optimum which is independent of the coordination mechanism. For example, for the coordination model of the selfish task allocation, a coordination mechanism is allowed to slow down the facilities, but *the optimum* $opt(w)$ *is computed using the original speeds.*

To a coordination mechanism c and set of player loads w corresponds a game; *the cost of a player* is the sum of the cost of all facilities used by the player. Let $Ne(w; c)$ be the set of (mixed) Nash equilibria of this game. We define the *price of anarchy* (or coordination ratio) of a coordination mechanism c as the maximum over all set of loads w and all Nash equilibria E of the social cost over the social optimum.

$$\text{PA}(c) = \sup_{w} \sup_{E\in \text{Ne}(w;c)} [\text{sc}(w; c; E)/opt(w)]$$

We define the price of anarchy of a coordination model as the minimum price of anarchy over all its coordination mechanisms.

The situation is very similar to the framework of competitive analysis in online algorithms or the analysis of approximation algorithms. Online algorithms address the lack of information by striving to reduce the competitive ratio; approximation algorithms address the lack of sufficient computational resources by striving to reduce the approximation ratio. In a similar way, coordination mechanisms address the lack of coordination due to selfish behavior by striving to reduce the price of anarchy.

The analogy also helps to clarify one more issue: *Why do we need to minimize the price of anarchy and not simply the cost of the worst-case Nash equilibrium?*

In the same way that it is not in general possible to have an online algorithm that minimizes the cost for *every input,* it is not in general possible to have a mechanism that minimizes the cost of the worst-case Nash equilibrium for *every possible game of the coordination model.*

3 Selfish Task Allocation

We now turn our attention to the coordination model for selfish task allocation. There are n players with loads and m identical facilities (machines or links). The objective of each player is to minimize the finish time. The mechanism designer has to select and announce a scheduling policy on each facility once and for all (without the knowledge of the loads). The scheduling policy on each facility must depend only on its own loads (and not on loads allocated to the other machines).

Let's first consider the case of $m = 2$ facilities. In retrospect, the coordination mechanism considered in [11] schedules the loads on each link in a random order resulting in the price of anarchy of 3/2. Consider now the following mechanism:

Increasing-Decreasing: "*The loads are ordered by size. If two or more loads have the same size, their order is the lexicographic order of the associated players. Then the first facility schedules its loads in order of* increasing *size while the second facility schedules its loads in order of* decreasing *size.*"

This mechanism aims to break the symmetry of loads. It is easy to see that the agent with the minimum load goes always to the first link. Similarly, the agent with the maximum load goes to the second link.

Proposition 1. *The above increasing-decreasing coordination mechanism has price of anarchy 1 for* $n \leq 3$ *and* 4/3 *for* $n \geq 4$.

Is there a better coordination mechanism for 2 or more facilities? To motivate the better coordination mechanism consider the case of $n = m$ players each with load 1. Symmetric coordination mechanisms in which all facilities have the same scheduling policy have very large price of anarchy: The reason is that there is a Nash equilibrium in which each player selects randomly (uniformly) among the facilities; this is similar to the classical bins-and-balls random experiment, and the price of anarchy is the expected maximum: $\Theta(\log m / \log \log m)$.

It is clear that the large price of anarchy results when players "collide". Intuitively this can be largely avoided in pure equilibria. To make this more precise consider the case where all loads have distinct sizes and furthermore all partial sums are also distinct. Consider now the coordination mechanism for m machines where every machine schedules the jobs in decreasing order; furthermore to break the "symmetry" assume that machine i has a multiplicative delay $i\epsilon$ for each job and for some small $\epsilon > 0$. Then in the only Nash equilibrium the largest job goes to the first machine, the next job goes to second machine and so on; the next job in decreasing size goes to the machine with the minimum load. There is a small complication if the multiplicative delays $i\epsilon$ create some tie, but we can select small enough ϵ so that this never happens.

It should be clear that this is a mechanism with small price of anarchy. But what happens if the jobs are not distinct or the multiplicative delays $i\epsilon$ create ties? We can avoid both problems with the following coordination mechanism that is based on two properties:

- Each facility schedules the loads in decreasing order (using the lexicographic order to break any potential ties).
- For each player, the cost on the facilities are different. To achieve this, the cost $c_i^j(w_1, \ldots, w_n)$ is a number whose representation in the $(m+1)$-ary system ends at j. To achieve this, the facility may have to introduce a small delay (at most a multiplicative factor of δ, for some fixed small δ). For example for $m = 9$ machines and $\delta = 0.01$, if a job of size $w_i = 1$ is first (greatest) on machine 7 it will not finish at time 1 but at time 1.007.

Theorem 1. *The above coordination mechanism for n players and m facilities has price of anarchy $4/3 - 1/(3m)$.*

Proof. There is only one Nash equilibrium: The largest load is "scheduled" first on every facility independently of the remaining loads, but there is a unique facility for which the players' cost is minimum. Similarly for the second largest load there is a unique facility with minimum cost independently of the smaller loads. In turn this is true for each load. Notice however that this is exactly the greedy scheduling with the loads ordered in decreasing size. It has been analyzed in Graham's seminal work [8] where it was established that its approximation ratio is $4/3 - 1/(3m)$. Given that the total delay introduced by the δ terms increases the social cost by at most a factor of δ, we conclude that the price of anarchy is at most $4/3 - 1/(3m) + \delta$. The infimum as δ tends to 0 is $4/3 - 1/(3m)$.

To see that this bound is tight we reproduce Graham's lower bound: Three players have load m and for each $k = m+1, \ldots, 2m-1$, two players have load k. The social optimal is $3m$ but the coordination mechanism has social cost $4m-1$ (plus some δ term). □

Notice some additional nice properties of this coordination mechanism: there is a unique Nash equilibrium (thus players are easy to "agree") and it has low computational complexity. In contrast, computing Nash equilibria is potentially a hard problem —its complexity is in general open.

The above theorem shows that good coordination mechanisms reduce the price of anarchy from $\Theta(\log m / \log \log m)$ to a small constant. Is there a coordination mechanism with better price of anarchy than $4/3 - 1/3m$? We conjecture that the answer is negative.

Finally we observe that the above mechanism reduces the question about the price of anarchy to the question of the approximation ratio of the greedy algorithm. This naturally extends to the case of machines with speeds. In this case, the price anarchy is $2 - 2/(m+1)$ and it follows from results in [7].

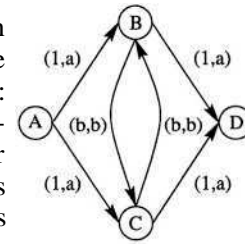
Theorem 2. *The above coordination mechanism for n players and m facilities with different speeds has price of anarchy $2 - 2/(m+1)$.*

The mechanism is appropriate for congestion games on any *network with linear cost functions* (the above discussion concerns the special case of m parallel edges). In this case, if we apply the same mechanism to every edge of the network, the price of anarchy is the approximation ratio of the greedy algorithm for selecting n paths. We point out that the price of anarchy is not known for these congestion games, yet we can still analyze the price of anarchy of the associated coordination mechanisms (in analogy, the analysis of Graham's algorithm is easier than determining the exact price of anarchy for m machines). For lack of space, we leave the analysis of these extensions for the full version of the paper.

4 Congestion Games

In the previous section, we discussed coordination mechanisms for linear delay functions. In this section we will discuss coordination mechanisms for arbitrary delay functions. We will also consider pure equilibria —these games have at least one pure equilibrium.

Consider the single-commodity congestion game with $n = 2$ players defined by the network of the figure, where the labels on the edges represent facility/edge costs: $(c^e(1), \ldots, c^e(n))$. For $a \gg b \gg 1$, there is a Nash equilibrium where player 1 selects path *ABCD* and player 2 selects path *ACBD;* its social cost is $2 + b$. *opt* is (*ABD, ACD*) with cost 2. Hence the price of anarchy is $(2 + b)/2$ which can be arbitrarily high. Therefore



Proposition 2. *Without a coordination mechanism, the price of anarchy of congestion games (even of single-commodity ones) is unbounded.*

We consider symmetric coordination mechanisms that can increase the cost $c^j(k)$ of each facility. Can coordination mechanisms reduce the price of anarchy for congestion games? We believe that the answer is positive for general congestion games with *monotone* facility costs, i.e., when $c^j(k) \leq c^j(k+1)$ for all j and k[1]. But we were able to establish it only for single-commodity games.

4.1 Single-Commodity Congestion Games

Let n denote the number of players. Our lower bound is (proof in the full version):

Theorem 3. *There are congestion games (even single-commodity ones) for which no coordination mechanism has price of anarchy less than* n.

We will now show that this lower bound is tight.

[1] For the unnatural case of non-monotone facility costs, it can easily shown that no coordination mechanism has bounded price of anarchy.

Theorem 4. *For every single-commodity congestion game there is a coordination mechanism with price of anarchy at most* n.

The proof uses the notion of potential [20,13] of a set of strategies/paths. To define it, let $A = (A_1, \ldots, A_n)$ be strategies for the n players and let $n^e = n^e(A)$ denote the number of occurrences of edge e in the paths $A_1, \ldots, A_n$. The potential $P(A)$ is defined as $\sum_e \sum_{k=1}^{n^e} c^e(k)$ and plays a central role: The set of strategies A is a Nash equilibrium if and only if $P(A)$ is a local minimum (i.e., when we change the strategy of only one player, the potential can only increase). It is also useful to bound the social cost as suggested by the following lemma (proof in the full version).

Lemma 1. *For every strategy* A: $sc(A) \le P(A) \le n \cdot sc(A)$.

The idea of a coordination mechanism for Theorem 4 is simple: Let $A^* = (A_1^*, \ldots, A_n^*)$ be a set of strategies that minimize the social cost (and achieve the social optimal). Let $n^e(A^*)$ be the number of occurrences of edge e in the paths $A_1^*, \ldots, A_n^*$. The coordination mechanism keeps the same cost $c^e(k)$ for $k \le n^e(A^*)$, but changes the cost $c^e(k) = a$ for $k > n^e(A^*)$ to some sufficiently large constant $a \gg 1$:

$$\hat{c}^e(k) = \begin{cases} c^e(k) & k \le n^e(A^*) \\ a^2 & \text{for every } k \text{ when } n^e(A^*) = 0 \\ a & \text{otherwise} \end{cases}$$

The last two cases assign very high cost to edges that are used beyond the capacity determined by the optimal solution A^*. The middle case assigns even higher cost to edges not used at all by A^* to guarantee that they are not used by any Nash equilibrium also.

The idea of the mechanism is that the high cost a will discourage players to use each edge e more than $n^e(A^*)$ times and therefore will end up at a set of strategies A with the same occurrences of edges as in A^*. This in turn would imply that A and A^* have the same potential and the theorem will follow from Lemma 1. However natural this idea for coordination mechanism may be, it is not guaranteed to work —there may exist Nash equilibria that use some edges more than A^* (with cost a) but each individual player cannot switch to a path consisting entirely of low cost edges. We have an example for general congestion games where this happens, but the following lemma shows that this cannot happen for single-commodity games (details in the full version):

Lemma 2. *Let* G *be a directed acyclic (multi) graph (dag) whose edges can be partitioned into* n *edge-disjoint paths from* s *to* t. *Let* $A_1, \ldots, A_n$ *be any paths from* s *to* t. *Then there is some* i *and a path* A_i' *from* s *to* t *which is edge-disjoint from the paths* $A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n$.

Proof (of Theorem 4). Consider an optimal set of strategies $A^* = (A_1^*, \ldots, A_n^*)$. The multigraph G formed by these n paths from s to t should be acyclic. Consider also a Nash equilibrium $A = (A_1, \ldots, A_n)$ for the above-defined coordination

mechanism $\hat{c}$. The paths use only edges of G, otherwise some player would benefit by switching to a (any) s-t path of G. Using Lemma 2 we can also guarantee that the paths use edges of G with multiplicity equal or smaller than the multiplicity of G. In conclusion, the potential $P(A)$ is no greater than the potential $P(A^*)$ and the theorem follows from Lemma 1. □

Another interesting fact that follows easily from similar considerations is that the above coordination mechanism $\hat{c}$ has price of anarchy at most $V - 1$ for single-commodity networks of V nodes.

It is open whether the above coordination mechanism works well for multi-commodity games. But, as mentioned above, it does not work for general games (details in the full version). We conjecture however that there are (other) coordination mechanisms with price of anarchy n for every congestion game with positive monotone costs.

5 Open Problems

There are many variants of congestion games for which we don't know their price of anarchy, let alone the price of anarchy of the corresponding coordination models and mechanisms. The problems are parameterized by whether we consider *pure or mixed* Nash equilibria, by whether the flow is *splittable or unsplittable,* and by whether the social cost is the *maximum or the average* cost of the players. Then there is the class of delay functions: linear $(c(x) = a \cdot x)$, affine $(c(x) = a \cdot x + b)$, or general. Finally, we can distinguish between the *weighted and unweighted* cases (where the loads are all equal or not) and between *symmetric or asymmetric* coordination mechanisms (in the latter case the mechanism can prioritize the players).

The immediate problems that are left open by our results include the gap between the upper and the lower bound for the task allocation problem. Also in Section 4.1, we considered only congestion games with no weights (and no adversary). What is the price of anarchy when the players have weights w_i or simply when an adversary can select which players will participate (this corresponds to 0-1 weights)? A more distributed mechanism is required in this case.

Finally, in mechanism design there is the notion of truthfulness (strategyproof). Similar issues arise for coordination mechanisms. For example, the coordination mechanism for the task allocation problem that achieves price of anarchy $4/3 - 1/(3m)$ has the property that it favors (schedules first) large loads. This is undesirable since it gives incentive to players to lie and pretend to have larger load. Consider now the mechanism that is exactly the same but schedules the loads in increasing order. Using the same ideas as in the proof of Theorem 1, we can show that this coordination mechanism has price of anarchy $2 - 2/(m+1)$. Although this is greater than $4/3 - 1/(3m)$, the mechanism is very robust (truthful) in that the players have no incentive to lie (if we, of course, assume that they can't shrink their loads). Are there other robust coordination mechanisms with better price of anarchy? Also, for the case of different speeds,

the mechanism that orders the job in increasing size has non-constant price of anarchy (at least logarithmic [4]). Are there truthful mechanisms with constant price of anarchy for this case?

References

1. A. Archer and E. Tardos. Frugal Path Mechanisms. In *ACM-SIAM SODA,* 2002.
2. R. Cole, Y. Dodis, and T. Roughgarden. How much can taxes help selfish routing. In *ACM EC,* pages 98–107, 2003.
3. R. Cole, Y. Dodis, and T. Roughgarden. Pricing network edges for heterogeneous selfish users. In *ACM STOC,* pages 521–530, 2003.
4. Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM J. Comput.* 9(1):91–103, February 1980.
5. A. Czumaj and B. Vöcking. Tight Bounds for Worst-case Equilibria. In *ACM-SIAM SODA,* pp. 413–420, 2002.
6. A. Fabrikant, C. Papadimitriou, and K. Tulwar. On the complexity of pure equilibria. `www.cs.berkeley.edu/~christos/papers/pure.ps`
7. T. Gonzalez, O. Ibarra, and S. Sahni. Bounds for LPT schedules on uniform processors. *SIAM J. Comput.* 6(1):155–166, March 1977.
8. R. L. Graham. Bounds for certain multiprocessing anomalies, *Bell System Technical Journal,* 45: 1563-1581, 1966.
9. Y. Korilis, A. Lazar, and A. Orda, Architecting Noncooperative Networks. *IEEE Journal on Selected Areas in Communications,* Vol. 13, No. 7, pp. 1241–1251, September 1995.
10. E. Koutsoupias, M. Mavronicolas, and P. Spirakis. Approximate Equilibria and Ball Fusion. In *Proceedings of the 9th International Colloquium on Structural Information and Communication Complexity (SIROCCO),* 2002
11. E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. In *STACS,* pages 404-413, 1999.
12. I. Milchtaich. Congestion Games with Player-Specific Payoff Functions. *Games and Economic Behavior* 13, pages 111-124, 1996.
13. D. Monderer and L. S. Shapley. Potential Games. *Games and and Economic Behavior* 14, pages 124-143, 1996.
14. D. Monderer and M. Tennenholtz. k-Implementation. In *ACM EC,* pp 19–28, 2003.
15. N. Nisan. Algorithms for selfish agents: Mechanism design for distributed computation. In *STACS,* pp 1–15, 1999.
16. N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior,* 35:166–196, 2001.
17. M. J. Osborne and A. Rubinstein. *A Course in Game Theory.* The MIT Press, 1994.
18. C. H. Papadimitriou. Algorithms, games, and the Internet. In *ACM STOC,* pp 749-753, 2001.
19. A. Ronen. Algorithms for rational agents. In *Conference on Current Trends in Theory and Practice of Informatics,* pages 56–70, 2000.
20. R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory,* 2:65-67, 1973.
21. T. Roughgarden. Designing networks for selfish users is hard. In *IEEE FOCS,* pp 472–481, 2001.

22. T. Roughgarden. The price of anarchy is independent of the network topology. In *ACM STOC,* pp 428-437, 2002.
23. T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of the ACM,* 49(2):236-259, 2002.
24. J. G. Wardrop. Some theoretical aspects of road traffic research. In *Proceedings of the Institute of Civil Engineers,* Pt. II, volume 1, pages 325-378, 1952.

Online Scheduling of Equal-Length Jobs: Randomization and Restarts Help

Marek Chrobak[1], Wojciech Jawor[1], Jiří Sgall[2] ,and Tomáš Tichý[2]

[1] Department of Computer Science, University of California, Riverside, CA 92521.
{marek,wojtek}@cs.ucr.edu
[2] Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic.
{sgall,tichy}@math.cas.cz

Abstract. The input of the studied scheduling problem is a set of jobs with equal processing times, where each job is specified by its release time and deadline. The goal is to determine a single-processor, non-preemptive schedule that maximizes the number of completed jobs. In the online version, each job arrives at its release time.
First, we give a barely random $\frac{5}{3}$-competitive algorithm that uses only one random bit; we also show a lower bound of $\frac{3}{2}$ for barely random algorithms that choose one of two deterministic algorithms. Second, we give a deterministic $\frac{3}{2}$-competitive algorithm in the model that allows restarts, and we show that in this model the ratio $\frac{3}{2}$ is optimal.

1 Introduction

We consider the following fundamental problem in the area of real-time scheduling. The input is a collection of jobs with equal processing times p, where each job j is specified by its release time r_j and deadline d_j. The desired output is a single-processor non-preemptive schedule. Naturally, each scheduled job must be executed between its release time and deadline, and different jobs cannot overlap. The term "non-preemptive" means that each job must be executed without interruptions. The objective is to maximize the number of completed jobs.

In the *online version,* each job j arrives at time r_j, and its deadline d_j is revealed at this time. The number of jobs and future release times are unknown. At each time step when no job is running, we have to decide whether to start a job, and if so, to choose which one, based only on the information about the jobs released so far. An online algorithm is called *c-competitive* if on every input instance it schedules at least $1/c$ as many jobs as the optimum.

Our results. It is known that a simple greedy algorithm is 2-competitive for this problem, and that this ratio is optimal for deterministic algorithms. We present two ways to improve the competitive ratio of 2.

First, addressing an open question in [8,9], we give a $\frac{5}{3}$-competitive randomized algorithm. Interestingly, our algorithm is *barely random;* it chooses with probability $\frac{1}{2}$ one of two deterministic algorithms, i.e., needs only one random bit. These two algorithms are two *identical* copies of the same deterministic algorithm, that are run concurrently and use a shared lock to break the symmetry

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 358–370, 2004.

and coordinate their behaviors. We are not aware of previous work in the design of randomized online algorithms that uses such mechanism to coordinate identical algorithms—thus this technique may be of its own, independent interest. We then show a lower bound of $\frac{3}{2}$ on the competitive ratio of barely random algorithms that choose one of two deterministic algorithms, with any probability.

Second, we give a deterministic $\frac{3}{2}$-competitive algorithm in the *preemption-restart* model. In this model, an online algorithm is allowed to abort a job during execution, in order to start another job. The algorithm gets credit only for jobs that are executed contiguously from beginning to end. Aborted jobs can be restarted (from scratch) and completed later. Note that the final schedule produced by such an algorithm is *not* preemptive. Thus the distinction between non-preemptive and preemption-restart models makes sense only in the online case. (The optimal solutions are always the same.) In addition to the algorithm, we give a matching lower bound, by showing that no deterministic online algorithm with restarts can be better than $\frac{3}{2}$-competitive. We also show a lower bound of $\frac{6}{5}$ for randomized algorithms with restarts.

We remark that both our algorithms are natural, easy to state and implement. The competitive analysis is, however, fairly involved, and it relies on some structural lemmas about schedules of equal-length jobs. Some technical details are omitted in this version due to the page limit.

Previous work. The problem of scheduling equal-length jobs to maximize the number of completed jobs has been well studied in the literature. In the offline case, an $O(n \log n)$-time algorithm for the feasibility problem (checking if *all* jobs can be completed) was given by Garey *et al.* [7] (see also [15,4].) The maximization version can also be solved in polynomial time [5,2], although the known algorithms are rather slow. (Carlier [4] claimed an $O(n^3 \log n)$ algorithm but, as pointed out in [5], his algorithm is not correct.)

For the online version, Goldman *et al.* [8] gave a lower bound of $\frac{4}{3}$ on the competitive ratio of randomized algorithms and the tight bound of 2 for deterministic algorithms. We sketch these lower bounds, to see which behavior our algorithms need to avoid. Let $p \geq 2$. The jobs, written in the form $j = (r_j, d_j)$, are $1 = (0, 2p+1)$, $2 = (1, p+1)$, $3 = (p, 2p)$. The instance contains jobs 1,2 or jobs 1,3; in both cases the optimum is 2. In the deterministic case, release job 1; if at time 0 the online algorithm starts job 1, then release job 2, otherwise release job 3. The online algorithm completes only one job and the competitive ratio is no better than 2. For the randomized case, using Yao's principle, we choose each of the two instances with probability $\frac{1}{2}$. The expected number of completed jobs of any deterministic online algorithm is at most 1.5, thus the competitive ratio is no better than $\frac{2}{1.5} = \frac{4}{3}$.

Goldman *et al.* [8] show that the lower bound of 2 can be beaten if the jobs on input have sufficiently large "slack"; they prove that a greedy algorithm is $\frac{3}{2}$-competitive for instances where $d_j - r_j \geq 2p$ for all jobs j. This is closely related to our algorithm with restarts: On such instances, our algorithm never uses restarts and becomes identical to the greedy algorithm. Thus in this special case our result constitutes an alternative proof of the result from [8]. Goldwasser [9]

obtained a parameterized version of this result: if $d_j - r_j \geq \lambda p$ for all jobs j, where $\lambda \geq 1$ is an integer, then the competitive ratio is $1 + 1/\lambda$.

In our brief overview of the literature given above we focused on the case when jobs are of equal length and the objective function is the number of completed jobs. There is vast literature on real-time scheduling problems where a variety of other models is considered. Other or no restrictions can be placed on processing times, jobs may have different weights (benefits), we can have multiple processors, and preemption may be allowed.

The model with restarts was studied before by Hoogeveen *et al.* [11]. They present a 2-competitive deterministic algorithm with restarts for jobs with arbitrary processing times and objective to maximize the number of completed jobs. They also give a matching lower bound. Their algorithm does not use restarts on the instances with equal processing times, and thus it is no better than 2-competitive for our problem.

Real-time scheduling is an area where randomized algorithm have been found quite effective. Most randomized algorithms in the general scenarios use the classify-and-randomly-select technique by Lipton and Tomkins [12]. Typically, this method decreases the dependence of competitive ratio from linear to logarithmic in certain parameters (e.g., the maximum ratio between job weights), but it does not apply to the case of jobs with equal lengths and weights.

Barely random algorithms have been successfully applied in the past to a variety of online problems, including the list update problem [13], the k-server problem [3] and makespan scheduling [1,6,14]. In particular, the algorithm of Albers [1] involves two deterministic processes in which the second one keeps track of the first and corrects its potential "mistakes"—a coordination idea somewhat similar to ours, although in [1] the two processes are not symmetric.

2 Preliminaries

Notation and terminology. The instance on input is a set of jobs $J = \{1, 2, ...\}$. Each job j is given by its release time r_j and deadline d_j. All jobs have processing time p. (We assume that all numbers are positive integers and that $d_j \geq r_j + p$ for all j.) The *expiration time* of a job j is $x_j = d_j - p$, i.e., the last time when it can be started. A job j is called *admissible* at time t if $r_j \leq t \leq x_j$. A job j is called *tight* if $x_j - r_j < p$.

A *non-preemptive schedule* A assigns to each completed job j an interval $[S_j^A, C_j^A)$, with $r_j \leq S_j^A \leq x_j$ and $C_j^A = S_j^A + p$, during which it is executed. These intervals are disjoint for distinct jobs. S_j^A and C_j^A are called the *start time* and *completion time* of job j. Both are assumed to be integer, w.l.o.g. We adopt a convention that "job running (a schedule being idle, etc.) at time t" is an equivalent shortcut for "job running (a schedule being idle, etc.) in the interval $[t, t+1)$". Given a schedule A, a job is *pending* at time t in A if it is admissible at t (that is, $r_j \leq t \leq x_j$) but not yet completed in A.

A set of jobs P is called *feasible* at time t if there exists a schedule which completes all jobs in P such that no job is started before t. P is *flexible* at time

t if it is feasible at time $t+p$. We say that a job started by a schedule A at time t is *flexible in A* if the set of all jobs pending in A at t is flexible; otherwise the job is called *urgent.* Intuitively, a job is flexible if we could possibly postpone it and stay idle for time p, without losing any of the currently pending jobs; this could improve the schedule if a tight job arrives. On the other hand, postponing an urgent job can bring no advantage to the algorithm.

An *online algorithm* constructs a schedule incrementally, at each step t making decisions based only on the jobs released at or before t. Each job j is revealed (including its deadline) to the algorithm at its release time r_j. A *non-preemptive online algorithm* can start a job only when no job is running; thus, if a job is started at time t the algorithm has no choice but to let it complete by the time $t+p$. An *online algorithm with restarts* can start a job at any time. If we start a job j when another job, say k, is running, then k is aborted and started from scratch when (and if) it is started again later. The unfinished portion of k is removed from the final schedule, which is considered to be idle during this time interval. Thus the final schedule generated by an online algorithm with restarts is non-preemptive.

An online algorithm is called *c-competitive* if, for any set of jobs J and any schedule ADV for J, the schedule A generated by the algorithm on J satisfies $|\mathrm{ADV}| \leq c|A|$. If the algorithm is randomized, the expression $|A|$ is replaced by the expected (average) number of jobs completed on the given instance.

The definitions above assume the model (standard in the scheduling literature) with integer release times and deadlines, which implicitly makes the time discrete. Some papers on real-time scheduling work with continuous time. Both our algorithms can be modified to the continuous time model and unit processing time jobs without any changes in performance, at the cost of somewhat more technical presentation.

Properties of schedules. For every instance J, we fix a canonical linear ordering $\prec$ of J such that $j \prec j'$ implies $d_j \leq d_{j'}$. In other words, we order the jobs by their deadlines, breaking the ties consistently for all applications of the deadline ordering. The term *earliest-deadline,* or briefly ED, now refers to the $\prec$-minimal job.

A schedule A is called EDF (earliest-deadline-first) if, whenever it starts a job, it chooses the ED job of all the pending jobs that are later completed in A.

A schedule is *normal* if (i) whenever it starts a job, it chooses the ED job from the set of all pending jobs, and (ii) whenever the set of all pending jobs is not flexible, it starts a job. Both conditions (i) and (ii) are reasonable, in the sense that any algorithm can be modified, using a standard exchange argument, to satisfy them without reducing the number of scheduled jobs. Furthermore, the conditions can be guaranteed by an online algorithm; indeed, all our algorithms generate normal schedules. Obviously, any normal schedule is EDF, but the reverse is not true. The following property is useful, the simple proof is omitted.

Lemma 2.1. *Suppose that a job j is urgent in a normal schedule A. Then at any time t, $S_j^A \leq t \leq x_j$, an urgent job is running in A.*

Two schedules D and D' for an instance J are called *equivalent* if D starts a job at t if and only if D' starts a job at t; furthermore, the job started in D is flexible if and only if the job started in D' is flexible. Obviously, $|D| = |D'|$ for equivalent schedules.

To facilitate analysis, we modify normal schedules into equivalent EDF schedules with better structural properties. The idea of the construction in the next lemma is straightforward: Keep a list of jobs that we plan to schedule. If the set of all pending jobs is feasible, we plan to schedule them all. If this set is flexible and some more jobs arrive, we have a choice. Namely, for any scheduled flexible job j, we can specify one job $f(j)$ that is then guaranteed to be scheduled by property (1); we use it in our proofs with $f(j)$ depending on the optimal schedule. Property (2) guarantees that any job planned to be scheduled is indeed scheduled in the future. Property (3) is a technical condition needed in the analysis of the algorithm with restarts.

Lemma 2.2. *Let A be a normal schedule for and $f(j) : J \to J$ a partial function such that if $f(j)$ is defined then j is scheduled as flexible in A and $r_{f(j)} < C_j^A \leq x_{f(j)}$. Then there exists an EDF schedule A' equivalent to A such that:*

(1) *All jobs $f(j)$ are completed in A'.*

(2) *If j is admissible at time t when A' is idle or A' starts a job and the set of jobs pending in A' is feasible at t, then j is completed in A'. In particular, if j is admissible when A' starts a flexible job then j is completed in A'.*

(3) *Let j be a job completed in A, let $t = S_j^A$, and let R be the set of all jobs j' with $r_{j'} < t + p$ that are pending at $t + p$. If R is feasible at $t + p$ then all the jobs in R are completed in A'.*

Furthermore, if A is constructed by an online algorithm and $f(j)$ can be determined online at time C_j^A, then A' can be produced by an online algorithm.

Lemma 2.2 gives an easy proof that any normal schedule A schedules at least half as many jobs as the optimum. Take the modified schedule A' from Lemma 2.2. Charge any job j completed in ADV to a job completed in A' as follows: (i) If A' is running a job k at time S_j^{ADV}, charge j to k. (ii) Otherwise charge j to j. This is well defined, since if at time S_j^{ADV}, j is admissible and A' is idle, then A' completes j by Lemma 2.2(2). Furthermore, only one job can be charged to k using (i), as only one job can be started in ADV during the interval when k is running in A'. Thus overall at most two jobs are charged to each job in A' and $|\mathrm{ADV}| \leq 2|A'| = 2|A|$, as claimed. This shows that any online algorithm that generates a normal schedule is 2-competitive. In particular, this includes the known result that the greedy algorithm which always schedules the ED pending job when there are any pending jobs is 2-competitive. We use similar but more refined charging schemes to analyze our improved algorithms.

A concept of algorithms that upon release of a job immediately commit if it will be completed or not was recently introduced [10]. We do not formulate our algorithms in this form, but Lemma 2.2 can be applied to normal schedules generated by our algorithms, with f undefined, to obtain equivalent online algorithms with immediate notification. (Note that with restarts this implies that any preempted job is completed later.)

3 Randomized Algorithms

In this section we present our $\frac{5}{3}$-competitive barely random algorithm. This algorithm needs only one random bit; at the beginning of computation it chooses with probability $\frac{1}{2}$ between two schedules. We also show a lower bound for barely random algorithms: any randomized algorithm that randomly chooses between two schedules has ratio at least $\frac{3}{2}$.

Algorithm RANDLOCK. At the beginning we choose with probability $\frac{1}{2}$ one of two identical processes A or B. The schedule that the algorithm will output is generated by the chosen process. Next, each process computes one schedule of its own copy of the instance J. (This means that with the exception of the lock, the processes are independent; e.g., a given job can be executed by both processes at the same or different times.) Note that even though the algorithm outputs only one of the two schedules A or B, it actually needs to simulate both processes to compute it. Each process works as follows:

(1) If there is no pending job, wait for the next arrival.
(2) If the set of pending jobs is not flexible, execute the ED pending job.
(3) If the set of pending jobs is flexible and the lock is available, acquire the lock (ties broken arbitrarily), execute the ED pending job, and release the lock upon its completion.
(4) Otherwise wait until the lock becomes available or the set of pending jobs becomes non-flexible (due to progress of time and/or job arrivals).

Theorem 3.1. RANDLOCK *is a* $\frac{5}{3}$*-competitive non-preemptive randomized algorithm for scheduling equal-length jobs.*

Proof. Let A and B denote the schedules generated by the corresponding processes on a given instance J. It is easy to see that RANDLOCK is a non-preemptive online algorithm and both schedules are normal. Fix an arbitrary schedule ADV for the given instance J.

We start by modifying the schedules A and B according to Lemma 2.2. We define partial functions f^D, $D \in \{A, B\}$. Define $f^D(j) = k$ if j is a flexible job completed in D and k is a job started in ADV during the execution of j in D and admissible at the completion of j in D, i.e., such that $S_j^D \le S_k^{\mathrm{ADV}} < C_j^D \le x_k$. Otherwise (if j is not flexible or no such k exists), $f^D(j)$ is undefined. Note that if k exists, it is unique for a given j.

Let D' be the schedule constructed in Lemma 2.2 using $f = f^D$. We stress that D' cannot be constructed online as its definition depends on ADV; it is only a tool for the analysis of RANDLOCK. Since D' is equivalent to a normal schedule D, Lemma 2.1 still applies and the number of completed jobs remains the same as well.

To avoid clutter, we slightly abuse the notation and from now on we use A and B to denote the modified schedules A' and B'. Whenever D denotes one of the processes or schedules A and B, then $\bar{D}$ denotes the other one.

Observation: An important property guaranteed by the lock mechanism is that if D is idle at time t and the lock is available (i.e., $\bar{D}$ is idle or executing an urgent

job), then each job j admissible at t is completed by time t in D, as otherwise D would schedule some job at time t. Furthermore, any such j is executed as flexible: otherwise, by Lemma 2.1, D cannot be idle at time t, $S_j^D \le t \le x_j$.

The charging scheme. Let j be a job started in ADV at time $t = S_j^{\mathrm{ADV}}$. This job generates several charges of different weights to (the occurrences of) the jobs in schedules A and B. There are two types of charges: *self-charges* from job j to the occurrences of j in A or B, and *up-charges*, from j to the jobs running at time t in A and B. The total of charges generated by j is always 1.

Case (I): Both schedules A and B are idle. By the observation above, in both A and B, j is flexible and completed by time t. We generate two self-charges of $\frac{1}{2}$ to the two occurrences of j in A and B.

Case (II): One schedule $D \in \{A, B\}$ is running an urgent job k and the other schedule $\bar{D}$ is idle. By the observation, in $\bar{D}$, j is flexible and completed by time t. We generate a self-charge of $\frac{1}{2}$ to the occurrence of j in $\bar{D}$ and an up-charge of $\frac{1}{2}$ to k in D.

Case (III): One schedule $D \in \{A, B\}$ is running a flexible job k and the other schedule $\bar{D}$ is idle. We claim that j is completed in both A and B. This follows from Lemma 2.2(2) for $\bar{D}$ and also for D, if $r_j \le S_k^D$. If $x_j \ge C_k^D$ then $f^D(k) = j$ and D completes j by Lemma 2.2(1). In the remaining case, we have $S_k^D < r_j \le t \le x_j \le C_k^D$; thus j is a tight job admissible at t and $\bar{D}$ cannot be idle, contradicting the case condition.

In this case we generate one up-charge of $\frac{1}{3}$ to k in D and two self-charges of $\frac{1}{2}$ and $\frac{1}{6}$ to the occurrences of j according to the subcases as follows. Let $E \in \{A, B\}$ be the schedule which starts j first (breaking ties arbitrarily).

Case (IIIa): If E schedules j as an urgent job and the other schedule $\bar{E}$ is idle at some time t' satisfying $S_j^E \le t' \le x_j$, then charge $\frac{1}{6}$ to the occurrence of j in E and $\frac{1}{2}$ to the occurrence of j in $\bar{E}$. Note that by the observation above, in $\bar{E}$, j is flexible and completed by time t'.

Case (IIIb): Otherwise charge $\frac{1}{2}$ to the occurrence of j in E and $\frac{1}{6}$ to the occurrence of j in $\bar{E}$.

Case (IV): Both processes A and B are running jobs k_A and k_B, respectively, at time t. We show in Lemma 3.2 that one of k_A and k_B receives a self-charge of at most $\frac{1}{6}$ from its occurrence in ADV. This job receives an up-charge of $\frac{2}{3}$ from j and the other one of k_A and k_B an up-charge $\frac{1}{3}$ from j.

Lemma 3.2. *In case* (IV), *either* k_A *or* k_B *receives a self-charge of at most* $\frac{1}{6}$.

Proof. Any self-charge has weight $\frac{1}{2}$ or $\frac{1}{6}$. Assume, towards contradiction, that both k_A and k_B receive a self-charge of $\frac{1}{2}$. At least one of k_A and k_B is scheduled as urgent in the corresponding schedule, due to the lock mechanism. Thus $k_A \neq k_B$, as (I) is the only case when two self-charges $\frac{1}{2}$ to the same job are generated and then both occurrences are flexible. Furthermore, if $j = k_D$, $D \in \{A, B\}$, then k_D has no self-charge. Thus k_A, k_B, and j are three distinct jobs.

Claim: If k_D, $D \in \{A, B\}$, receives a self-charge of $\frac{1}{2}$ in case (IIIb) (applied to k_D) and $S_{k_D}^{\mathrm{ADV}} \le t - p$ (i.e., k_D is scheduled before j in ADV), then $k_{\bar{D}} \prec k_D$.

Proof: If (IIIb) applies, generating a self-charge of $\frac{1}{2}$ to k_D then $\bar{D}$ schedules k_D after $k_{\bar{D}}$: we have $S^{\bar{D}}_{k_D} \geq S^{D}_{k_D} > t - p$, on the other hand $S^{\bar{D}}_{k_{\bar{D}}} < t$ and $k_D \neq k_{\bar{D}}$. Furthermore, $S^{\bar{D}}_{k_{\bar{D}}} > t - p \geq S^{\mathrm{ADV}}_{k_D} \geq r_{k_D}$ and thus k_D is pending in $\bar{D}$ when $k_{\bar{D}}$ is started. Since $\bar{D}$ is EDF, we have $k_{\bar{D}} \prec k_D$, as claimed. □

Choose D such that k_D is urgent in D (as noted above, such D exists). The only case when an urgent job receives a self-charge of $\frac{1}{2}$ is (IIIb). By Lemma 2.1, D executes urgent jobs at all times t', $t \leq t' \leq x_{k_D}$, which implies that $S^{\mathrm{ADV}}_{k_D} \leq t$ (otherwise (III) does not apply to k_D). As $j \neq k_D$, it follows that $S^{\mathrm{ADV}}_{k_D} \leq t - p$ By the claim, $k_{\bar{D}} \prec k_D$ and $x_{k_{\bar{D}}} \leq x_{k_D}$. Furthermore, since (IIIa) does not apply, $\bar{D}$ is also not idle at any time t', $t \leq t' \leq x_{k_D}$.

If $k_{\bar{D}}$ is self-charged $\frac{1}{2}$ in cases (I), (II), (IIIa) or the subcase of (IIIb) when $S^{\mathrm{ADV}}_{k_{\bar{D}}} > t$, then at least one process is idle at some time t', $t < t' \leq x_{k_{\bar{D}}} \leq x_{k_D}$, which is a contradiction with previous paragraph. If $k_{\bar{D}}$ is self-charged $\frac{1}{2}$ in the subcase of (IIIb) when $S^{\mathrm{ADV}}_{k_{\bar{D}}} \leq t$, then $S^{\mathrm{ADV}}_{k_{\bar{D}}} \leq t - p$ as $j \neq k_{\bar{D}}$, and the claim above applies to $k_{\bar{D}}$; however the conclusion that $k_D \prec k_{\bar{D}}$ contradicts the linearity of $\prec$ as $k_D \neq k_{\bar{D}}$ and we have already shown that $k_{\bar{D}} \prec k_D$. We get a contradiction in all the cases, completing the proof of the lemma. □

Finally, we show that the total charge to each occurrence of a job in A or B is at most $\frac{5}{6}$. During the time when a job is running in A or B, at most one job is started in ADV, thus each job gets at most one up-charge in addition to a possible self-charge (in certain degenerate cases these two may come from the same job in ADV). If a job does not receive any up-charge, it is self-charged $\frac{1}{2}$ or $\frac{1}{6}$, i.e., less than $\frac{5}{6}$. If a job k in D receives an up-charge in (II), it is an urgent job and, since the $\bar{D}$ is idle, it is already completed in $\bar{D}$; thus (IIIb) does not apply to k, the self-charge is at most $\frac{1}{6}$ and the total is at most $\frac{1}{6} + \frac{1}{2} < \frac{5}{6}$. If a job receives an up-charge in (III), the up-charge is only $\frac{1}{3}$ and thus the total is at most $\frac{1}{3} + \frac{1}{2} = \frac{5}{6}$. If a job receives an up-charge in (IV), Lemma 3.2 implies that the up-charges can be defined as claimed in the case description. The total charge is then bounded by $\frac{1}{6} + \frac{2}{3} = \frac{5}{6}$ and $\frac{1}{2} + \frac{1}{3} = \frac{5}{6}$, respectively.

The expected number of jobs completed by RANDLOCK is $\frac{1}{2}(|A| + |B|)$ and $\frac{5}{3}$-competitiveness now follows by summing the charges over all jobs. □

Theorem 3.3. *Suppose that $\mathcal{A}$ is a barely-random non-preemptive algorithm for scheduling equal-length jobs that chooses one of two deterministic algorithms. Then $\mathcal{A}$ is not better than $\frac{3}{2}$-competitive.*

Proof. Assume that we have two deterministic algorithms, A and B, of which one is chosen as the output schedule randomly, with arbitrary probability. Let $p \geq 3$ and write the jobs as $j = (r_j, d_j)$. We start with job $1 = (0, 4p)$. Let t be the first time when one of the algorithms, say A, schedules job 1. If B schedules it at t as well, release a job $1' = (t + 1, t + p + 1)$; the optimum schedules both jobs while both A and B only one, so the competitive ratio is at least 2.

So we may assume that B is idle at t. Release job $2 = (t+1, t+2p+2)$. If B starts any job (1 or 2) at $t+1$, release job $3 = (t+2, t+p+2)$, otherwise release

$4 = (t+p+1, t+2p+1)$. By the choice of the last job, B completes only one of the jobs 2, 3, 4. Since A is busy with job 1 until time $t+p$, it also completes only one of the jobs 2, 3, 4, as their deadlines are smaller than $t+3p$. So both A and B complete two jobs. The optimum completes three jobs: If 3 is issued, schedule 3 and 2, back to back, starting at time $t+2$. If 4 is issued, schedule 2 and 4, back to back, starting at time $t+1$. In either case, two jobs fit in the interval $[t+1, t+2+2p)$. If $t \geq p-1$, schedule job 1 at time 0, otherwise schedule job 1 at time $3p \geq t+2+2p$. Thus the competitive ratio is at least $\frac{3}{2}$. □

4 Scheduling with Restarts

Our algorithm with restarts is very natural. At any time, it greedily schedules the ED job. However, if a tight job arrives that would expire before the running job is completed, we consider a preemption. If all pending jobs can be scheduled, the preemption occurs. If not, it means that some pending job is necessarily lost and the preemption would be useless—so we continue running the current job and let the tight job expire.

We need an auxiliary definition. Suppose that a job j is started at time s by the algorithm. We call a job k a *preemption candidate* if $s < r_k \leq x_k < s+p$.

Algorithm TIGHTRESTART. At time t:

(1) If no job is running, start the ED pending job, if there are any pending jobs, otherwise stay idle.
(2) Otherwise, let j be the running job. If no preemption candidate is released at t, continue running j.
(3) Otherwise, choose a preemption candidate k released at t (use the ED job to break ties.) Let P be the set of all jobs pending at time t, excluding any preemption candidates (but including j). If P is feasible at $t+p$, preempt j and start k at time t. Otherwise continue running j.

Theorem 4.1. TIGHTRESTART *is a* $\frac{3}{2}$*-competitive algorithm with restarts for scheduling equal-length jobs.*

Proof. Let A be the final schedule generated by TIGHTRESTART, after removing the preempted parts of jobs. We stress that we distinguish between A being idle and TIGHTRESTART being idle: at some time steps TIGHTRESTART can process a job that will be preempted later, in which case A is considered idle but TIGHTRESTART is not.

Obviously, TIGHTRESTART is an online algorithm with restarts, and any job it starts is the ED pending job. To prove that A is a normal schedule, we need a few more observations:

(A) A job j that was started as urgent is never preempted: Let R be the set of pending jobs at time t' when j is started, and suppose that at time t a preemption candidate arrives. If $x_j < t$ then j itself is not feasible at t. Otherwise all jobs in R are pending at t (as j is the ED job in R) and thus $P \supseteq R$ cannot be feasible at $t+p$ since already R is not feasible at $t'+p < t+p$.

(**B**) If j is preempted, then this happens on the first release of a preemption candidate: The condition in step (3) only gets stronger with further jobs released. Also, by (A), j is flexible and thus no job pending at its start expires.

(**C**) If A is idle at t' but a job j is running at t' and preempted at time $t > t'$, the set R of all jobs pending at time t' (including j) is flexible: Since j is flexible and R does not contain any preemption candidates by (B), we have $R \subseteq P$ where P is the set in step (3) of the algorithm at time t. If j is preempted at time t, P is flexible at t, thus R is flexible at $t' < t$.

Summarizing, A always starts the ED pending job; if a preemption occurs, we use (B) and the choice of the scheduled preemption candidate to see that it is ED. (A) implies that if an urgent job is started, it is also completed, and (C) implies that if A is idle then the set of pending jobs is flexible. Thus A is a normal schedule and we can proceed towards application of Lemma 2.2.

Define a partial function $f : J \to J$ as follows. Let j be a job scheduled as flexible in A.

- If at some time t, $S_j^A \le t < C_j^A$, ADV starts a job k which is not a preemption candidate then let $f(j) = k$.
- Otherwise, if there exists a job k with $S_j^A < r_k < C_j^A \le x_k$ such that ADV does not complete k, then let $f(j) = k$ (choose arbitrarily if there are more such k's).
- Otherwise, $f(j)$ is undefined.

Let A' be the schedule constructed in Lemma 2.2 from A and the function f. As before, we abuse A to denote the modified schedule A' as well.

Call a job j scheduled in ADV a *free* job if TIGHTRESTART is idle at time S_j^{ADV}. This implies that at time S_j^{ADV} no job is pending in A; in particular, j is completed by time S_j^{ADV} in A. (These jobs need special attention, as TIGHTRESTART was "tricked" into scheduling them too early.)

If a job j in ADV is started while a job k is running in A, we want to charge j to k. However, due to preemptions, the jobs can become misaligned, so we replace this simple matching by a more technical definition. We match the jobs from the beginning of the schedule, a job k in A is matched to the next job in ADV, provided that it starts later than k; an exception is that if k is free and no j starts in ADV while k is running in A, then we prefer to match k to itself.

Formally, define a partial function $M : J \to J$ which is a matching of (some) occurrences of jobs in A to those in ADV. Process the jobs k scheduled in A in the order of increasing S_k^A. Let j be the first unmatched job started in ADV after S_k^A, i.e., a job with smallest S_j^{ADV} among those with $S_j^{\mathrm{ADV}} \ge S_k^A$ and not in the current range of M (i.e., for no k' with $S_{k'}^A < S_k^A$, $j = M(k')$). If no such j exists, $M(k)$ is undefined. If k is a free job, not in the current range of M, and $S_j^{\mathrm{ADV}} \ge C_k^A$, then let $M(k) = k$. Otherwise let $M(k) = j$.

The definition implies that M is one-to-one. Furthermore, for any j scheduled in ADV, if A is executing a job k at S_j^{ADV}, then $j = M(k')$ for some k': if j is not in the range of M before k is processed then $M(k)$ is defined as j.

Lemma 4.2. *If j is free and $f(j)$ is undefined then j is in the range of M.*

Proof. Since j is free, it is completed in A before it is started in ADV. Let k be the job started in ADV at some time t, $S_j^A \le t < C_j^A$. If no such k exists or $M(j) \neq k$ then j is in the range of M and the lemma holds: if j is not in the range of M before j is processed, then $M(j)$ is defined to be j.

Since $f(j)$ is undefined, k is a preemption candidate. Thus it remains to handle the case when k is a preemption candidate, yet TIGHTRESTART does not preempt, and $M(j) = k$.

The idea is this: Since j is not preempted, A schedules many jobs after j and before d_j. Intuitively, one of these jobs should overlap in time with the occurrence of j in ADV, so eventually in the definition of M one of these jobs matches j. This gets a bit technical, first because of possible gaps in the schedules, and second because we need to verify that these jobs are not free (and thus not matched to their occurrence in ADV which may be after j). Details are omitted. □

Charging scheme. Let j be a job started at time t in ADV. Note that case (I) below always applies when A is not idle at t, so the cases exhaust all possibilities.
Case (I): $j = M(k)$ for some k: Charge j to k.
Case (II): Otherwise, if A and TIGHTRESTART are idle at t, i.e., j is free: Since (I) does not apply, Lemma 4.2 implies that $f(j)$ is defined. Charge $\frac{1}{2}$ of j to the occurrence of j in A and $\frac{1}{2}$ of j to the occurrence of $f(j)$ in A.
Case (III): Otherwise, if A is idle at t, but TIGHTRESTART is running a job k' which is later preempted by a job k: By Lemma 2.2(2), j is completed in A. The job k is tight and thus it is completed as well. Charge $\frac{1}{2}$ of j to k and $\frac{1}{2}$ of j to the occurrence of j in A.

Analysis. We prove that each job scheduled in A is charged at most $\frac{3}{2}$. Each job is charged at most 1 in case (I), as M defines a matching.

We claim that the total charge from cases (II) and (III) is $\frac{1}{2}$. The jobs j receiving self-charges in cases (II) and (III) are obviously distinct. The case analysis below shows that the other jobs receiving charges in (II) and (III) can uniquely determine the corresponding j and that if they are scheduled in ADV then (I) applies to them and thus they cannot play the role of j in (II) and (III).

In (II), $f(j)$ either is started in ADV during the execution of j in A, or it is not executed in ADV at all and arrives during the execution of j in A; this uniquely determines the corresponding j. Also, in the first case, at $S_{f(j)}^{\mathrm{ADV}}$, A is running j, and thus (I) applies to $f(j)$. By definition, $f(j)$ is not a preemption candidate, so it cannot play the role of k in (III).

In (III), job k, as a preemption candidate, is tight, and since it preempts another job, $S_k^A = r_k$. Thus if ADV schedules k, at S_k^{ADV}, A is executing k, and (I) applies to k. The corresponding job j is uniquely determined as the job j running in ADV at time r_k.

We conclude that each job completed in A gets at most one charge of $\frac{1}{2}$ and thus is charged a total of at most $\frac{3}{2}$. The competitive ratio of $\frac{3}{2}$ now follows by summing the charges over all jobs. □

Theorem 4.3. *For scheduling equal-length jobs with restarts, no deterministic algorithm is less than* $\frac{3}{2}$*-competitive and no randomized algorithm is better than* $\frac{6}{5}$*-competitive.*

Proof. For $p \geq 2$, consider four jobs given in the form $j = (r_j, d_j)$: $1 = (0, 3p+1)$, $2 = (1, 3p)$, $3 = (p, 2p)$, $4 = (p+1, 2p+1)$. The instance consists of jobs 1,2,3 or 1,2,4. The optimum is 3. In the deterministic case, choosing the instance based on the action of the algorithm we can guarantee that the online algorithm schedules only 2 jobs. In the randomized case, we choose each instance with probability $\frac{1}{2}$. Each online algorithm then on average schedules 2.5 jobs. Details omitted. □

Acknowledgments. We are grateful for useful comments of anonymous referees. Chrobak and Jawor supported by NSF grants CCR-9988360 and CCR-0208856. Sgall and Tichý partially supported by Institute for Theoretical Computer Science, Prague (project LN00A056 of MŠMT ČR) and grant IAA1019401 of GA AV ČR.

References

1. S. Albers. On randomized online scheduling. In *Proc. 34th Symp. Theory of Computing (STOC),* pages 134–143. ACM, 2002.
2. P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *J. of Scheduling,* 2:245–252, 1999.
3. Y. Bartal, M. Chrobak, and L. L. Larmore. A randomized algorithm for two servers on the line. *Information and Computation,* 158:53–69, 2000.
4. J. Carlier. Problèmes d'ordonnancement à durées égales. *QUESTIO,* 5(4):219–228, 1981.
5. M. Chrobak, C. Dürr, W. Jawor, Ł. Kowalik, and M. Kurowski. A note on scheduling equal-length jobs to maximize throughput. manuscript, 2004.
6. L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger. Randomized on-line scheduling for two related machines. *J. of Scheduling,* 4:71–92, 2001.
7. M. Garey, D. Johnson, B. Simons, and R. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM J. on Computing,* 10(2):256–269,1981.
8. S. A. Goldman, J. Parwatikar, and S. Suri. Online scheduling with hard deadlines. *J. of Algorithms,* 34:370–389, 2000.
9. M. H. Goldwasser. Patience is a virtue: The effect of slack on the competitiveness for admission control. *J. of Scheduling,* 6:183–211, 2003.
10. M. H. Goldwasser and B. Kerbikov. Admission control with immediate notification. *J. of Scheduling,* 6:269–285, 2003.
11. H. Hoogeveen, C. N. Potts, and G. J. Woeginger. On-line scheduling on a single machine: Maximizing the number of early jobs. *Operations Research Letters,* 27:193–196, 2000.
12. R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. 5th Symp. on Discrete Algorithms (SODA),* pages 302–311. ACM/SIAM, 1994.
13. N. Reingold, J. Westbrook, and D. D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica,* 11:15–32, 1994.

14. S. Seiden. Barely random algorithms for multiprocessor scheduling. *J. of Scheduling*, 6:309–334, 2003.
15. B. Simons. A fast algorithm for single processor scheduling. In *Proc. 19th Symp. on Foundations of Computer Science (FOCS)*, pages 246–252, IEEE, 1978.

Efficient Computation of Equilibrium Prices for Markets with Leontief Utilities

Bruno Codenotti* and Kasturi Varadarajan**

Department of Computer Science, The University of Iowa
Iowa City IA 52242 (USA)
{bcodenot,kvaradar}@cs.uiowa.edu.

Abstract. We present a polynomial time algorithm for the computation of the market equilibrium in a version of Fisher's model, where the traders have Leontief utility functions. These functions describe a market characterized by strict complementarity. Our algorithm follows from a representation of the equilibrium problem as a concave maximization problem, which is of independent interest. Our approach extends to a more general market setting, where the traders have utility functions from a wide family which includes CES utilities.

1 Introduction

Back in 1891, Fisher [2,18] introduced a market model given by a set of buyers and a set of divisible goods. Buyers have specified incomes, goods are available in given amounts, and the preferences of each buyer are expressed in terms of a concave utility function. The equilibrium problem consists of finding prices (of goods) and allocations (of goods to buyers) which *clear* the market and allow each buyer to maximize her utility function.

Devanur et al. [5] introduced a polynomial time algorithm for the *linear version* of Fisher's model, i.e., for the special case where the buyers have linear utility functions. Their approach is based on a primal-dual scheme, and boils down to a number of max-flow computations. A polynomial time algorithm for the linear case of Fisher's model was already implicit from the work of Gale ([11], pp. 281-287). Gale showed that the allocation which leads to equilibrium prices can be obtained by maximizing a concave function subject to linear constraints. The solution to this program can be approximated in polynomial time by using the ellipsoid algorithm. As pointed out in [5], since the equilibrium prices (in the linear case) are rational, the ellipsoid method indeed returns the exact solution.

The above results for the linear case are a valuable starting point for our understanding of computational and structural properties of equilibrium problems. However linear utility functions are realistic only for very particular markets or for the analysis of small price variations. In real world markets, utilities are typically concave functions.

* The first author is on leave from IIT-CNR, Pisa, Italy.
** The second author is supported by an NSF CAREER award CCR-0237431.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 371–382, 2004.

In this paper, we consider the equilibrium problem for a more realistic version of Fisher's model, where traders have utility functions which are known as *Leontief utility functions,* or *fixed proportions utility functions.* These functions have the form $u(z_1, \ldots, z_n) = \min\{\frac{z_1}{b_1}, \ldots, \frac{z_n}{b_n}\}$, where $b_j > 0$, and $z = (z_1, \ldots, z_n)$ represent a bundle (or basket) of goods. These utilities express strict complementarity in the preferences. Indeed a buyer with this utility wants to get a basket of goods proportional to $(b_1, \ldots, b_n)$ (see, e.g., [19], p. 1009).

Leontief utilities are an important special case of a rather general and widely used family of utility functions, known as *constant elasticity of substitution* (CES). (See the next section for definitions and properties.)

Our result builds upon the construction of a constrained nonlinear maximization problem, where the constraints are linear and express the feasibility of the allocations. The function to be maximized is simply the product, over all buyers, of the individual utility of each buyer raised to her income. Using a duality-type argument, we prove that from the solution to this maximization problem we can derive equilibrium allocations, from which we can in turn compute equilibrium prices by Linear Programming.

We also show that, unlike in the linear case, the equilibrium prices need not be rational. Therefore there is no choice but to settle for an approximation.

From the above properties, we immediately derive a polynomial time approximation scheme for computing market clearing prices: (1) we first use the ellipsoid or some other efficient convex programming method to find the values of the individual utilities which maximize the product of the utilities raised to the income; (2) from such values, we then compute the optimal allocations, by a straightforward calculation; (3) we finally use Linear Programming to compute the equilibrium prices by finding a nonnegative solution to a system of linear equations which relates equilibrium prices to optimal allocations.

This paper shows that there is a realistic market setting for which the equilibrium problem can be solved in polynomial time; it also shed some further light on the structure of the market equilibrium problem. Indeed, as it happens for Gale's construction for the linear case, our solution is based on the existence of a function which aggregates the preferences of the buyers in a simple way, and which leads, at equilibrium prices, to the optimization of the individual preferences, thus essentially translating an equilibrium problem into an optimization problem. More precisely, our work shows that the optimal allocations are given by the values which maximize the product of the buyers' utilities raised to their respective incomes, subject to constraints dictated by the data which describe the market problem.

We also show that our approach extends to a more general scenario when the utility functions can be chosen from a fairly general class, which includes constant elasticity of substitution utility functions. We sketch the ideas of this extension in this paper, and report the details in a joint paper with Jain and Vazirani [3], which also contains other extensions.

Polynomial time algorithms for market equilibrium problems are known only in a few other instances. There is a polynomial time algorithm for markets with

Cobb-Douglas utilities, which are concave functions maximized when the buyers spend a fixed fraction of their income on each of the goods (see next section for precise definitions). The algorithm has been introduced by Curtis Eaves [9], and is based on simple linear algebra considerations which make it possible to translate the equilibrium problem into that of finding the nonnegative solution to a particular linear system. Eaves shows how to compute such solution in a direct way, avoiding Linear Programming, and achieving a cubic time bound.

The rest of this paper is organized as follows. In Section 2 we provide background on both existential and computational results concerning market equilibria. In Section 3 we present our main results. We show that equilibrium prices for the Leontief setting can be computed from allocations which solve a convex optimization problem. We also show that the equilibrium prices can be irrational, thus pointing out the need of shooting for approximate solutions. In Section 4 we sketch some more general results, which show that the approach of Section 3 extends to a fairly general class of utility functions, which include CES functions.

2 Background and Some History

We now concisely describe the market model. Let us consider m economic agents which represent producers and/or consumers of n goods. Each agent has a utility function $u : \mathbf{R}^n \to \mathbf{R}$, which represents her preferences for the different baskets of goods, and an initial, typically suboptimal, endowment of goods $w \in \mathbf{R}^n$. At given prices $\pi \in \mathbf{R}^n$, each agent will sell her endowment, and get the basket of goods $z \in \mathbf{R}^n$ which maximizes u subject to her budget constraint[1] $\pi^T z \leq \pi^T w$.

The celebrated Arrow-Debreu theorem [1] states that, under some quite mild assumptions, there is a price vector $\hat{\pi}$ such that the solution to the above maximization problem by each agent leads to an allocation $z(\hat{\pi})$ which clears the market. These prices are called *equilibrium prices.*

The proof of the Arrow-Debreu Theorem uses Kakutani's Fixpoint theorem, which is a generalization of Brouwer's Fixpoint theorem.

The above described market model is usually called *Arrow-Debreu model.* If we restrict the model by getting rid of the production component, we obtain a market where all the agents are traders which want to exchange their initial endowments in order to get a basket of goods which maximizes their utilities. This setting is called the *exchange model.* If we assume that the goods are initially available in the market, and that agents go to the market with fixed amounts of money (their income), then we get *Fisher's model,* where all the agents are buyers.

The market equilibrium problem, as well as the related problem of finding a Nash equilibrium in the mixed strategies for a two person nonzero sum game, has been analyzed from a computational viewpoint by Papadimitriou in [16]. Papadimitriou explores the nature of these problems, as revealed by their intimate connection with fixpoint theorems, pointing out that they are characterized by

[1] We use $x^T y$ to denote the inner product of two vectors x and y.

inefficient proofs of existence. Indeed the fixpoint theorems provide existential proofs which can be used a basis for (inefficient) computations. The actual computational complexity of this family of problems turns out to be wide open and of great relevance in computational complexity (see [17], Sections 2 and 5).

Many attempts have been made to find efficient algorithms for the solution of the market equilibrium problem, using the interplay with the computation of fixed points. In particular, it is worth mentioning the work of Scarf and some coauthors [10,12,18,19]. For example, in [18] Scarf suggested an algorithm for the approximation of fixed points of continuous mappings of a simplex into itself. In [14], Kuhn showed the connection between Scarf's result and Sperner's lemma and proposed a technique for the subdivision of the simplex, which yields a simple algorithm for the simplicial approximation of fixed points.

Unfortunately, none of these results lead to algorithms with polynomial running time.

As already mentioned in the Introduction, efficient algorithms have been obtained for Fisher's model, when the utilities are linear [5]. The result in [5] has been extended in several directions. For instance, it has been used as the main ingredient in approximation schemes for the exchange model with linear utilities [13,7]. It has also inspired the definition of a new model, the *spending constraint model* [6], to which the technique used in [5] can still be applied.

Another instance where price equilibria can be found efficiently arises when the utilities are Cobb-Douglas functions. A Cobb-Douglas utility function is a function of the form $u(\mathbf{z}) = \prod_{j=1}^{n} z_j^{\alpha_j}$ where $\sum \alpha_j = 1$. In this case, Eaves has shown a nice and simple cubic time algorithm which works for the exchange model [9]. It is not difficult to show that a trader with a Cobb-Douglas utility spends a fixed fraction of her income on each good. In what follows we will assume that there are m traders and n goods. Then the market can be described in terms of two $n \times m$ matrices, A and W, whose entries are the utility exponents and the initial endowments, respectively. More precisely, the i-th column of the matrix A (W, resp.) contains the vector of utilities (initial endowments, resp.) of player i. Using the special properties of Cobb-Douglas utility functions, Eaves has shown that the equilibrium prices can be obtained from the solution of a linear system associated with the matrix $E = WA^T$, which leads to an algorithm consisting of one matrix multiplication and one application of Gaussian elimination.

Leontief, Cobb-Douglas, and linear utilities are special cases of constant elasticity of substitution (CES, for short) utility functions. A CES function has the form

$$u(z_1, \ldots, z_n) = \left(\sum_{j=1}^{n} \alpha_j^{\frac{1}{\sigma}} z_j^{\frac{\sigma-1}{\sigma}} \right)^{\frac{\sigma}{\sigma-1}},$$

where σ is the constant representing the given *elasticity of substitution.*

Leontief utilities are obtained in the limit as σ tends to zero, with zero elasticity of substitution, i.e., strict complementarity, while Cobb-Douglas utilities (obtained as σ tends to one) correspond to unitary elasticity of substitution.

Conversely, note that the case of linear utilities (obtained as σ tends to infinity) represents a situation where goods are *perfect substitutes.* (For more precise definitions and properties of the most popular utility functions, see [20], Chapters 1 and 7.)

3 Main Results

3.1 Preliminaries

We consider a market with a set $B = \{1, \ldots, m\}$ of buyers and a set $G = \{1, \ldots, n\}$ of goods. For each $1 \leq i \leq m$, let the real number $e_i > 0$ denote the initial endowment (or money) of buyer i. Associated with buyer i, there is also a vector $a_i = (a_{1i}, \ldots, a_{ni})$, with $a_{ji} > 0$, that describes her utility function. We will use the variable $x_{ji} \geq 0$ to denote the amount of the j'th good in buyer i's basket, and the vector $x_i = (x_{1i}, \ldots, x_{ni}) \in \mathbb{R}^n$ to denote a basket of buyer i. The utility function $u_i(x_i)$ of buyer i is given by[2]

$$u_i(x_i) = \min_{1 \leq j \leq n} \frac{x_{ji}}{a_{ji}}.$$

Let $q_j > 0$ denote the amount of good j in the market, for $1 \leq j \leq n$.

Given a market with a set B of buyers, with a vector a_i and endowment e_i for buyer i, and a set G of goods, with an amount q_j for each good j, an *equilibrium* is given by a *price vector* $\bar{\pi} = (\bar{\pi}_1, \ldots, \bar{\pi}_n) \in \mathbb{R}^n$, where $\bar{\pi}_j \geq 0$ is called the *price* of good j, and a basket $\bar{x}_i = (\bar{x}_{1i}, \ldots, \bar{x}_{ni}) \in \mathbb{R}^n$ for each buyer i, where each $\bar{x}_{ji}$ is nonnegative, satisfying the following conditions:

1. For each buyer i, the basket $\bar{x}_i$ maximizes her utility given the price vector $\bar{\pi}$ and her endowment e_i. That is, $\bar{x}_i$ is a vector that maximizes $u_i(x_i)$ subject to the constraint that $\bar{\pi}^T x_i \leq e_i$. Note that in our case this is equivalent to the requirement that $u_i(\bar{x}_i) = e_i / (\bar{\pi}^T a_i)$ and $\bar{\pi}^T \bar{x}_i = e_i$.
2. Each good j is cleared, that is, $\sum_{i=1}^{m} \bar{x}_{ji} = q_j$ for each $1 \leq j \leq n$.

3.2 Computing the Equilibrium

Let M be a market with a set $B = \{1, \ldots, m\}$ of buyers, with a vector $a_i \in \mathbb{R}^n$ and endowment $e_i > 0$ for each buyer i, and a set $G = \{1, \ldots, n\}$ of goods, with an amount $q_j > 0$ for each good j. In this section we give a new proof of the existence of an equilibrium for such a market. This proof immediately implies an efficient algorithm for computing an equilibrium.

Let A be the $n \times m$ matrix whose entry in the j-th row and i-th column is a_{ji}. That is, the i-th column of A is the vector a_i. Let $q \in \mathbb{R}^n$ be the vector

[2] This definition implies that buyer i has some interest in each good. Our approach readily generalizes to the scenario where there is a subset $G_i \subseteq G$ of goods and $u_i(x_i) = \min_{j \in G_i} x_{ji}/a_{ji}$.

$(q_1, \ldots, q_n)$. Let $\beta = (\beta_1, \ldots, \beta_m)$ stand for a variable in $\mathbb{R}^m$. Consider the following optimization problem, which we call CP:

$$\begin{aligned} &\text{maximize } (\beta_1^{e_1} * \beta_2^{e_2} * \cdots * \beta_m^{e_m})^{1/\sum_i e_i} \\ &\text{Subject to } \sum_{1 \le i \le m} a_{ji}\beta_i \le q_j, \text{ for } 1 \le j \le n \\ &\qquad \beta_i \ge 0, \text{ for } 1 \le i \le m. \end{aligned}$$

Since each $a_{ji} > 0$, the set of feasible solutions for this problem is bounded, hence compact, and so the continuous objective function attains its maximum. The objective function is concave, and so CP is a convex optimization problem. CP can be concisely stated as:

$$\begin{aligned} &\text{maximize } (\beta_1^{e_1} * \beta_2^{e_2} * \cdots * \beta_m^{e_m})^{1/\sum_i e_i} \\ &\text{Subject to } A\beta \le q \\ &\qquad \beta_i \ge 0, \text{ for } 1 \le i \le m. \end{aligned}$$

Let $\hat{\beta} = (\hat{\beta}_1, \ldots, \hat{\beta}_m)$ be an optimal solution to CP. We must have that $(\hat{\beta}_1^{e_1} * \hat{\beta}_2^{e_2} * \cdots * \hat{\beta}_m^{e_m})^{1/\sum_i e_i} > 0$. This is because for a sufficiently small $\delta > 0$, the point $(\delta, \ldots, \delta) \in \mathbb{R}^m$ is feasible for CP (since each $q_j > 0$) and the value of the objective function at this point is $\delta > 0$.

There is one constraint in CP corresponding to each good j, namely the constraint $\sum_{1 \le i \le m} a_{ji}\beta_i \le q_j$. Let G^t be the subset of goods for which the corresponding constraint is tight at $\hat{\beta}$. That is, $G^t = \{j | \sum_{1 \le i \le m} a_{ji}\hat{\beta}_i = q_j\}$. Let G^l denote the remaining goods. G^t *is* non-empty because otherwise for a sufficiently small $\delta > 0$ the solution $(1 + \delta)\hat{\beta}$ is feasible for CP and has an objective function value of

$$(1 + \delta)(\hat{\beta}_1^{e_1} * \hat{\beta}_2^{e_2} * \cdots * \hat{\beta}_m^{e_m})^{1/\sum_i e_i} > (\hat{\beta}_1^{e_1} * \hat{\beta}_2^{e_2} * \cdots * \hat{\beta}_m^{e_m})^{1/\sum_i e_i}.$$

Let D^* denote the $m \times m$ diagonal matrix, with diagonal entries $(\hat{\beta}_1, \ldots, \hat{\beta}_m)$. Then AD^* is the matrix whose (j, i)-th entry is $\hat{\beta}_i a_{ji}$. That is, the i-th column of AD^* is the vector $\hat{\beta}_i a_i$. We claim that the vector $(e_1, \ldots, e_m) \in \mathbb{R}^m$ is in the cone of the vectors $\{(\hat{\beta}_1 a_{j1}, \ldots, \hat{\beta}_m a_{jm}) | \ j \in G^t\}$ (the row vectors of AD^* that correspond to goods in G^t).

Claim 1. *For each good $j \in G^t$, there exists $\bar{\pi}_j \ge 0$ such that for each $1 \le i \le m$, we have $\sum_{j \in G^t} \bar{\pi}_j \hat{\beta}_i a_{ji} = e_i$.*

Proof. If the claim is false, then by Farkas Lemma [15] there is a $(t_1, \ldots, t_m) \in \mathbb{R}^m$ such that $e_1 t_1 + \cdots + e_m t_m > 0$ and for each $j \in G^t$, $\sum_{1 \le i \le m} t_i \hat{\beta}_i a_{ji} \le 0$. We will argue that, for a sufficiently small $\delta > 0$, the vector $\bar{\beta} = (\hat{\beta}_1(1 + \delta t_1), \ldots, \hat{\beta}_m(1 + \delta t_m))$ is a feasible solution for CP and the value of the objective

function at $\bar{\beta}$ is larger than at $\hat{\beta}$. This contradicts the fact that $\hat{\beta}$ is an optimal solution for CP.

For each $j \in G^t$, we have

$$\sum_{1\le i\le m} \bar{\beta}_i a_{ji} = \sum_{1\le i\le m} \hat{\beta}_i a_{ji} + \delta \sum_{1\le i\le m} t_i \hat{\beta}_i a_{ji} \le q_j + 0 = q_j,$$

so $\bar{\beta}$ satisfies the constraint in CP corresponding to j. For each $j \in G^l$, we have

$$\sum_{1\le i\le m} \bar{\beta}_i a_{ji} = \sum_{1\le i\le m} \hat{\beta}_i a_{ji} + \delta \sum_{1\le i\le m} t_i \hat{\beta}_i a_{ji} < q_j + \delta \sum_{1\le i\le m} t_i \hat{\beta}_i a_{ji},$$

so for a sufficiently small δ the point $\bar{\beta}$ satisfies the constraint in CP corresponding to j. Finally, we must have $\hat{\beta}_i > 0$ for otherwise the value of the objective function at $\hat{\beta}$ is 0. This implies that, for a sufficiently small δ, $\bar{\beta}_i = \hat{\beta}_i(1+\delta t_i) > 0$ as well. So the point $\bar{\beta}$ is a feasible solution for CP. Now, using the Taylor expansion,

$$\begin{aligned}
&(\bar{\beta}_1^{e_1} * \cdots * \bar{\beta}_m^{e_m}) - (\hat{\beta}_1^{e_1} * \cdots * \hat{\beta}_m^{e_m}) \\
&= (\hat{\beta}_1^{e_1} * \cdots * \hat{\beta}_m^{e_m})((1+\delta t_1)^{e_1} * \cdots * (1+\delta t_m)^{e_m} - 1) \\
&= (\hat{\beta}_1^{e_1} * \cdots * \hat{\beta}_m^{e_m})(\delta(e_1 t_1 + \cdots + e_m t_m) + \delta^2(\cdots) + O(\delta^3)) \\
&> 0,
\end{aligned}$$

for sufficiently small δ, since $e_1 t_1 + \cdots + e_m t_m > 0$ and $\hat{\beta}_1^{e_1} * \cdots * \hat{\beta}_m^{e_m} > 0$. But this means that the value of the objective function at $\bar{\beta}$ is greater than that at $\hat{\beta}$. This finishes the proof of the claim.[3] □

Theorem 2. *Let β^* be an optimal solution to CP, and assume that there are values $\bar{\pi}_j$ for each $j \in G^t$ as in Claim 1. Let $\bar{\pi}_j = 0$ for each $j \in G^l$. Set $\bar{x}_{ji} = \hat{\beta}_i a_{ji}$, for each $j \in G^t$ and each $1 \le i \le m$. Also set $\bar{x}_{ji} = \hat{\beta}_i a_{ji}$, for each $j \in G^l$ and each $2 \le i \le m$. Set*

$$\bar{x}_{j1} = q_j - \sum_{2\le i\le m} \hat{\beta}_i a_{ji} = q_j - \sum_{2\le i\le m} \bar{x}_{ji},$$

for each $j \in G^l$. Then the vector $\bar{\pi} \in \mathbb{R}^n$ and the vectors $\bar{x}_i = (\bar{x}_{1i}, \ldots, \bar{x}_{ni})$, for $1 \le i \le m$, are an equilibrium for the market M.

Proof. Note that $\bar{x}_{j1} \ge \hat{\beta}_1 a_{j1}$ because $\hat{\beta}$ is a feasible solution for CP. Each component of $\bar{\pi}$ is nonnegative by construction. We have $\bar{x}_{ji} \ge \hat{\beta}_i a_{ji} \ge 0$ for any $1 \le i \le m$ and $1 \le j \le n$, since each $\hat{\beta}_i \ge 0$ and each $a_{ji} > 0$. We now establish that the two conditions for equilibrium hold.

[3] The proof can also be established using the Kuhn-Tucker stationary-point necessary optimality theorem ([15], page 105).

1. Since $\bar{x}_{ji} \geq \hat{\beta}_i a_{ji}$, we have

$$u_i(\bar{x}_i) = \min_{1 \leq i \leq n} \frac{\bar{x}_{ji}}{a_{ji}} \geq \hat{\beta}_i,$$

for each good i. In fact, $u_i(\bar{x}_i) = \hat{\beta}_i$ since $\bar{x}_{ji} = \hat{\beta}_i a_{ji}$ for each good $j \in G^t$. Now the price of the basket $\bar{x}_i$ is

$$\sum_{j \in G^t} \bar{\pi}_j \bar{x}_{ji} + \sum_{j \in G^l} \bar{\pi}_j \bar{x}_{ji} = \sum_{j \in G^t} \bar{\pi}_j \hat{\beta}_i a_{ji} + 0 = e_i.$$

We have, for each $1 \leq i \leq m$,

$$\sum_{1 \leq j \leq n} \bar{\pi}_j \hat{\beta}_i a_{ji} = \sum_{j \in G^t} \bar{\pi}_j \hat{\beta}_i a_{ji} + \sum_{j \in G^l} \bar{\pi}_j \hat{\beta}_i a_{ji} = e_i + 0 = e_i.$$

Thus $u_i(\bar{x}_i) = \hat{\beta}_i = e_i / \left(\bar{\pi}^T a_i\right)$, and so we have established that $\bar{x}_i$ maximizes u_i for the given prices $\bar{\pi}$ and the endowment.

2. For each $j \in G^t$, we have

$$\sum_{1 \leq i \leq m} \bar{x}_{ji} = \sum_{1 \leq i \leq m} \hat{\beta}_1 a_{ji} = q_j$$

by definition of G^t. For each $j \in G^l$, we have $\sum_{1 \leq i \leq m} \bar{x}_{ji} = q_j$ by construction of the $\bar{x}_{ji}$. Thus all goods are cleared.

□

The Algorithm

The proof that an equilibrium exists yields the following algorithm for computing an equilibrium. We first solve the convex optimization problem CP to obtain $\hat{\beta}$. We then find the sets G^t and G^l by direct inspection. We then find values $\bar{\pi}_j \geq 0$ for each $j \in G^t$ such that, for each $1 \leq i \leq m$, $\sum_{j \in G^t} \bar{\pi}_j \hat{\beta}_i a_{ji} = e_i$. Note that this problem, which we denote by FP, involves finding a feasible solution for a system of linear inequalities and equalities. We set $\bar{\pi}_j = 0$ for each $j \in G^l$. We then use $\hat{\beta}$ to compute the baskets $\bar{x}_i$ for each buyer i as described in the proof. Since both CP and FP can be solved in polynomial time using the ellipsoid algorithm, we obtain a polynomial time algorithm to compute the equilibrium.

As we show below, the vector $\hat{\beta}$ can unfortunately consist of irrational numbers. This means that we have to settle for an approximation to $\hat{\beta}$, and therefore an approximate equilibrium, where the baskets and prices are such that the goods are almost cleared and the basket of each buyer almost optimizes her utility, given the prices and her endowment.

3.3 An Alternative Formulation

We can formulate the problem of computing an equilibrium in an alternative way as a feasibility problem with convex constraints. The problem FEAS is to find $\beta_1, \ldots, \beta_m$ and $\pi_1, \ldots, \pi_n$ satisfying

$$\sum_{1 \le i \le m} \beta_i a_{ji} \le q_j \text{ for } 1 \le j \le n$$

$$\beta_i \sum_{1 \le j \le n} \pi_j a_{ji} \ge e_i \text{ for } 1 \le i \le m$$

$$\sum_{1 \le i \le n} \pi_j q_j = \sum_i e_i$$

$$\pi_j \ge 0 \text{ for } 1 \le j \le n$$

$$\beta_i \ge 0 \text{ for } 1 \le i \le m.$$

It is not hard to show that any solution $\hat{\beta}_1, \ldots, \hat{\beta}_m$ and $\bar{\pi}_1, \ldots, \bar{\pi}_n$ of FEAS yields an equilibrium for our setting, and the prices and utilities at any equilibrium are a solution to FEAS. Note that this formulation does not guarantee the existence of such a solution. Also note that the second constraint of FEAS defines a convex set for nonnegative values of the β_i and the π_j, so this is indeed a convex feasibility problem.

3.4 Uniqueness and Irrationality

We now argue that the utilities at equilibrium are unique and can be irrational.

Theorem 3. *Let M be a market with a set $B = \{1, \ldots, m\}$ of buyers and a set $G = \{1, \ldots, n\}$ of goods, with a vector $a_i \in \mathbb{R}^n$ and endowment $e_i > 0$ for each buyer i, and an amount $q_j > 0$ for each good j. Let $\bar{\pi} \in \mathbb{R}^n$ be the vector of prices and $\bar{x}_i \in \mathbb{R}^n$ be a basket for buyer i, for each $1 \le i \le m$, so that the prices and the baskets constitute an equilibrium for the market M. Let $\bar{\beta}_i = \min_{1 \le j \le n} \bar{x}_{ji}/a_{ji}$ denote the utility of buyer i at equilibrium, and let $\bar{\beta} = (\bar{\beta}_1, \ldots, \beta_m) \in \mathbb{R}^m$. Then $\bar{\beta}$ is an optimal solution to CP.*

For lack of space we omit the proof of this theorem. The theorem says that the utilities at equilibrium must be an optimal solution to CP. It is easily verified that the objective function of CP is strictly quasi-concave and therefore CP has a unique optimal solution. Thus the utilities at equilibrium are unique. We now present an example, adapted from [8], of a market with two goods and three buyers for which the utilities at equilibrium are irrational. We have $q_1 = q_2 = 3$, $e_1 = e_2 = e_3 = 1$, and $a_1 = (1, 1/2)$, $a_2 = (1/2, 1)$, and $a_3 = (1/4, 1/5)$. The utilities at equilibrium are the solution to the program:

$$\begin{aligned} \text{maximize } & (\beta_1 * \beta_2 * \beta_3)^{1/3} \\ \text{Subject to } & \beta_1 + \frac{\beta_2}{2} + \frac{\beta_3}{4} \leq 3 \\ & \frac{\beta_1}{2} + \beta_2 + \frac{\beta_3}{5} \leq 3 \\ & \beta_i \geq 0, \text{ for } 1 \leq i \leq 3. \end{aligned}$$

The solution to this program is $\beta_1 = 2/\sqrt{3}$, $\beta_2 = 1 + 1/\sqrt{3}$, and $\beta_3 = 10 - 10/\sqrt{3}$. So the utilities at equilibrium are irrational for this market. This implies that both the equilibrium prices and the optimal baskets must contain irrational elements.

4 Generalizations

Let M be a market with a set $B = \{1, \ldots, m\}$ of buyers and a set $G = \{1, \ldots, n\}$ of goods, with a concave utility function $u_i : \mathbb{R}^n \to \mathbb{R}$ and endowment $e_i > 0$ for each buyer i, and an amount $q_j > 0$ for each good j. As before, we will denote by the variable x_{ji} the amount of good j in buyer i's basket, and by the vector $x_i = (x_{1i}, \ldots, x_{ni})$ the basket of buyer i.

Consider the following optimization problem CPG:

$$\begin{aligned} \text{maximize } & \Big(\prod_{1 \leq i \leq m} u_i(x_i)^{e_i}\Big)^{\frac{1}{e_1 + \cdots + e_m}} \\ \text{Subject to } & \sum_{1 \leq i \leq m} x_{ji} \leq q_j, \text{ for } 1 \leq j \leq n \\ & x_{ji} \geq 0, \text{ for } 1 \leq i \leq m, \ 1 \leq j \leq n. \end{aligned}$$

Since each u_i is a concave function, the objective function of CPG is concave, so this is a convex programming problem. Let $x \in \mathbb{R}^{mn}$ denote the vector $(x_{11}, \ldots, x_{n1}, x_{12}, \ldots, x_{n2}, \ldots, x_{1m}, \ldots, x_{nm})$. Let

$$\begin{aligned} \theta(x) &= -\Big(\prod_{1 \leq i \leq m} u_i(x_i)^{e_i}\Big)^{\frac{1}{e_1 + \cdots + e_m}} \\ g_j(x) &= \sum_{1 \leq i \leq m} x_{ji} - q_j, \text{ for } 1 \leq j \leq n \\ h_{ji}(x) &= -x_{ji}, \text{ for } 1 \leq i \leq m, \ 1 \leq j \leq n. \end{aligned}$$

Then CPG can be restated as a minimization problem:

$$\begin{aligned} \text{minimize } & \theta(x) \\ \text{Subject to } & g_j(x) \leq 0, \text{ for } 1 \leq j \leq n \\ & h_{ji}(x) \leq 0, \text{ for } 1 \leq i \leq m, \ 1 \leq j \leq n. \end{aligned}$$

Let us assume that the utility functions u_i satisfy some fairly general differentiability conditions. A vector $\bar{x} \in \mathbb{R}^{mn}$ and real numbers

$$\bar{\pi}_1, \ldots, \bar{\pi}_n, \bar{\lambda}_{11}, \ldots, \bar{\lambda}_{n1}, \bar{\lambda}_{12}, \ldots, \bar{\lambda}_{n2}, \ldots, \bar{\lambda}_{1m}, \ldots, \bar{\lambda}_{nm},$$

if they exist, are said to solve the Kuhn-Tucker stationary-point problem (KTP) if they satisfy the following four conditions ([15], page 94):

1. $\nabla\theta(\bar{x}) + \sum_j \bar{\pi}_j \nabla g_j(\bar{x}) + \sum_{j,i} \bar{\lambda}_{ji} \nabla h_{ji}(\bar{x}) = 0$.
2. $\bar{x}$ is a feasible solution to CPG.
3. $\sum_j \bar{\pi}_j g_j(\bar{x}) + \sum_{j,i} \bar{\lambda}_{ji} h_{ji}(\bar{x}) = 0$.
4. $\bar{\pi}_j \geq 0$, for $1 \leq j \leq n$, and $\bar{\lambda}_{ji} \geq 0$, for $1 \leq j \leq n$ and $1 \leq i \leq m$.

The Kuhn-Tucker stationary-point necessary optimality theorem ([15], page 105) states that if $\bar{x}$ is an *optimal* solution to CPG, then there exist $\bar{\pi}_j$ and $\bar{\lambda}_{ji}$ so that $\bar{x}$, the $\bar{\pi}_j$, and the $\bar{\lambda}_{ji}$ are a solution to KTP. The four requirements of KTP then imply that the baskets corresponding to $\bar{x}$ and prices obtained by multiplying the vector $(\bar{\pi}_1, \ldots, \bar{\pi}_n)$ by a suitable number yield a market equilibrium for *M, provided* each utility function u_i satisfies certain additional conditions. One sufficient condition is that

$$\sum_{1 \leq j \leq n} x_{ji} \frac{\partial u_i}{\partial x_{ji}} = u_i(x_i).$$

Observe that this sufficient condition is satisfied by linear utility functions, Cobb-Douglas utility functions, CES utility functions (and indeed all differentiable utility functions u that are homogeneous of degree one [20], that is, $u(\alpha x) = \alpha u(x)$ for every bundle x and real $\alpha > 0$). This yields a proof for the existence for an equilibrium when each utility function u_i is one of these types of functions. Also observe that the u_i are not required to be all of the same type.

This proof of existence also yields an efficient algorithm for computing an equilibrium: once we have solved the convex program CPG using the ellipsoid algorithm, we can immediately read off the prices from the four requirements of KTP. This becomes evident if we explicitly write out the four requirements in our case. (There is no need to solve a linear program as we did in the case of Leontief utilities.)

Leontief utilities, and in general utility functions defined as a minimum of a set of linear functions that have a value of 0 at the origin, are not differentiable and hence cannot be plugged directly into this paradigm but they can be dealt with using a slight variant.

Acknowledgment. We wish to acknowledge fruitful discussions with Sriram Pemmaraju on the topics of this paper. The first author would like to thank Varsha Dani, for several conversations on market equilibria, in particular on Cobb-Douglas utility functions.

References

1. K.J. Arrow and G. Debreu, Existence of an Equilibrium for a Competitive Economy, Econometrica 22 (3), pp. 265–290 (1954).
2. W.C. Brainard and H. Scarf, How to Compute Equilibrium Prices in 1891. Cowles Foundation Discussion Paper 1270 (2000).
3. B. Codenotti, K. Jain, K. Varadarajan, V. V. Vazirani. Market Equilibrium for Scalable Utilities and Production Models via Variational Calculus, submitted (2004).
4. X. Deng, C. H. Papadimitriou, M. Safra, On the Complexity of Equilibria, STOC 02.
5. N. R. Devanur, C. H. Papadimitriou, A. Saberi, V. V. Vazirani, Market Equilibrium via a Primal-Dual-Type Algorithm. FOCS 2002, pp. 389-395. (Full version with revisions available on line.)
6. N. R. Devanur, V. V. Vazirani, Extensions of the spending constraint-model: existence and uniqueness of equilibria (extended abstract). ACM Conference on Electronic Commerce 2003, pp. 202-203 (2003).
7. N. R. Devanur, V. V. Vazirani, An Improved Approximation Scheme for Computing Arrow-Debreu Prices for the Linear Case. FSTTCS 2003, pp. 149-155 (2003).
8. B. C. Eaves, A Finite Algorithm for the Linear Exchange Model, Journal of Mathematical Economics 3, 197-203 (1976).
9. B. C. Eaves, Finite Solution of Pure Trade Markets with Cobb-Douglas Utilities, Mathematical Programming Study 23, pp. 226-239 (1985).
10. B. C. Eaves and H. Scarf, The Solution of Systems of Piecewise Linear Equations, Mathematics of Operations Research, Vol. 1, No. 1, pp. 1-27 (1976).
11. D. Gale. The Theory of Linear Economic Models. McGraw Hill, N.Y. (1960).
12. T. Hansen and H. Scarf, The Computation of Economic Equilibrium, Cowles Foundation Monograph No. 24, New Haven: Yale University Press (1973).
13. K. Jain, M. Mahdian, and A. Saberi, Approximating Market Equilibria, Proc. APPROX 2003.
14. H.W. Kuhn, Simplicial Approximation of Fixed Points, Proc. National Academy of Sciences of the United States of America Vol. 61, n. 4, pp. 1238-1242 (1968).
15. O. L. Mangasarian. Nonlinear Programming, McGraw-Hill, 1969.
16. C. H. Papadimitriou, On the Complexity of the Parity Argument and other Inefficient Proofs of Existence, Journal of Computer and System Sciences 48, pp. 498-532 (1994).
17. C. H. Papadimitriou, Algorithms, Games, and the Internet, STOC 01, (2001).
18. H. Scarf, The Approximation of Fixed Points of a Continuous Mapping, SIAM J. Applied Math., 15, pp. 1328-1343 (1967).
19. H. Scarf, The Computation of Equilibrium Prices: An Exposition, in Arrow and Intriligator, editors, Handbook of Mathematical Economics, Volume II, pp. 1008-1061 (1982).
20. H. Varian, Microeconomic Analysis, New York: W.W. Norton, 1992.

Coloring Semirandom Graphs Optimally

Amin Coja-Oghlan*

Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
coja@informatik.hu-berlin.de

Abstract. We present heuristics and algorithms with polynomial expected running time for coloring semirandom k-colorable graphs made up as follows. Partition the vertex set $V = \{1, \ldots, n\}$ into k classes $V_1, \ldots, V_k$ randomly and include each V_i-V_j-edge $(i \neq j)$ with probability p independently. Then, an adversary adds further V_i-V_j-edges $(i \neq j)$. We show that if $np \geq \max\{(1+\varepsilon)k\ln(n), Ck^2\}$, an *optimal* coloring can be found in polynomial time with high probability. Furthermore, if $np \geq C\max\{k\ln(n), k^2\ln(k)\}$, an optimal coloring can be found in polynomial expected time. By contrast, it is NP-hard to find a k-coloring whp. if $np \leq (\frac{1}{2} - \varepsilon)k\ln(n/k)$.

1 Introduction and Results

The *graph coloring problem* - given a graph G, compute the chromatic number $\chi(G)$ - is of fundamental interest in theoretical computer science. At the same time, graph coloring is notoriously hard. Indeed, no polynomial time algorithm can approximate the chromatic number of graphs of order n within a factor of $n^{1-o(1)}$ (under a certain complexity theoretic assumption) [8]. These hardness results motivate the quest for *coloring heuristics* that always run in polynomial time and succeed on "most" instances, and for *algorithms with polynomial expected running time* that produce an optimal coloring on any input and whose "average" running time is polynomial (cf. the survey of Krivelevich [14]).

In order to evaluate heuristics rigorously, we need a stochastic model of the input instances. In the case of graph coloring, one could consider the $G_{n,p}$ model: construct a graph of order n by including every possible edge with probability ("w.p.") p independently. For instance if $p = 1/2$, the chromatic number of $G_{n,p}$ almost surely satisfies $\chi(G_{n,p}) \sim \frac{n}{2\log_2(n)}$, and the simple greedy algorithm for graph coloring uses $\sim \frac{n}{\log_2(n)}$ colors almost surely (cf. [13]). However, no heuristics is known that can color $G_{n,1/2}$ using $\leq (1-\varepsilon)\frac{n}{\log_2(n)}$ colors almost surely, where $\varepsilon > 0$ is an arbitrarily small constant (cf. [14]). As a consequence, $G_{n,p}$ has only limited relevance as a benchmark in distinguishing between "better" and "worse" coloring heuristics, because (for instance if $p = 1/2$) most known heuristics have about the same performance.

* Research supported by the Deutsche Forschungsgemeinschaft (grant DFG FOR 413/1-1).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 383–395, 2004.

In contrast to $G_{n,p}$, the $G_{n,p,k}$ model suggested by Kučera [15] allows to create random k-colorable graphs with a given density. The graph $G_{n,p,k}$ is obtained as follows. First, partition the vertex set $V = \{1, \ldots, n\}$ into k classes $V_1, \ldots, V_k$ of size n/k randomly (we assume that k divides n). Then, include every possible V_i-V_j-edge with probability $p = p(n)$ independently ($i \neq j$).

For what values of k and p we can k-color $G_{n,p,k}$ whp.? Kucera [16] has proved that for $k = \Theta(\sqrt{n/\ln(n)})$ and $p = 1/2$ a simple greedy heuristic succeeds. Concerning constant values of k, Alon and Kahale [1] have suggested a sophisticated heuristic based on spectral techniques that almost surely finds a k-coloring if $p \geq C_k/n$ for a sufficiently large constant $C_k > 0$. By contrast, the greedy algorithm almost surely fails to k-color $G_{n,p,k}$ in this range of k and p.

However, the $G_{n,p}$ and the $G_{n,p,k}$ model share a serious drawback: in both models the instances are purely random. As the theory of random graphs shows (cf. [13]), such instances have a very particular combinatorial structure. Therefore, designing heuristics for $G_{n,p}$ or $G_{n,p,k}$ yields heuristics for a *very special class* of graphs. Consequently, heuristics for purely random instances may lack "robustness", as even minor changes in the structure of the input may deteriorate the performance.

1.1 Semirandom Models

In order to figure out more robust heuristics and algorithmic techniques, we consider *semirandom models* where problem instances are made up of a random share and an adversarial part. In this paper, we consider semirandom graphs $G^*_{n,p,k}$ made up in two steps. First, choose a random k-colorable graph $G_0 = G_{n,p,k}$. Let $V_1, \ldots, V_k$ be its planted k-coloring. Then, an adversary may add further V_i-V_j-edges ($i \neq j$) to complete the instance $G = G^*_{n,p,k}$. We say that $G^*_{n,p,k}$ has some property *with high probability* ("whp.") if this property holds with probability $1 - o(1)$ as $n \to \infty$ regardless of the adversary's decisions. The $G^*_{n,p,k}$-model has been invented by Blum and Spencer [3].

Why is it natural to require that a "robust" heuristic should withstand the adversary's actions? The reason is that the adversary is just allowed to add more constraints (i.e. edges) that "point towards" the hidden coloring $V_1, \ldots, V_k$. Hence, the edges added by the adversary actually seem to *help*. However, neither the heuristic of Alon and Kahale [1] nor the one of Kučera [16] succeeds on $G^*_{n,p,k}$. For instance, to confuse the heuristic in [1], the adversary can jumble up the spectrum of the adjacency matrix by adding a few bipartite cliques of a suitable size between vertices in different color classes.

The first heuristic for coloring $G^*_{n,p,k}$ has been given by Blum and Spencer [3], who have shown that a k-coloring can be found in polynomial time whp. if k is constant and $np \geq n^{\alpha_k}$ for a certain $\alpha_k \geq 2/5$. Improving on this result, Feige and Kilian [9] have suggested a rather involved semidefinite programming ("SDP") based heuristic that finds a k-coloring whp. if k is constant and $np \geq (1+\varepsilon)k\ln(n)$. (Throughout, ε denotes an arbitrarily small constant > 0.)

We say that an algorithm has *polynomial expected running time* applied to $G^*_{n,p,k}$ if the expectation of the running time is bounded by a fixed polynomial

regardless of the behavior of the adversary. Clearly, coloring $G^*_{n,p,k}$ in polynomial expected time is a more demanding problem than coloring $G^*_{n,p,k}$ whp. For the case that k is constant and $np > n^{\alpha_k}$ for a certain $\alpha_k > 2/5$, Subramanian [18] has presented an algorithm that colors $G^*_{n,p,k}$ *optimally* in expected polynomial time. Furthermore, in [5] the author has given an algorithm that k-colors $G^*_{n,p,k}$ in expected polynomial time if k is a constant and $np \geq \omega \ln(n)$, where $\lim_{n\to\infty} \omega(n) = \infty$.

1.2 A Heuristic for Coloring $G^*_{n,p,k}$ Optimally

Observe that while $G^*_{n,p,k}$ is always k-colorable, it might happen that the chromatic number is actually smaller than k. Therefore, it makes sense to ask for heuristics that *color* $G^*_{n,p,k}$ *optimally whp.,* i.e. that output a coloring along with a certificate that this coloring uses precisely $\chi(G)$ colors whp.

Theorem 1. *Suppose that* $k = k(n)$ *and* $p = p(n)$ *are such that* $np \geq \max\{(1+\varepsilon)k\ln(n), C_0k^2\}$ *for a certain constant* C_0. *There is a polynomial time algorithm Color that colors* $G^*_{n,p,k}$ *optimally whp.*

Note that for $k = o(\ln n)$ – hence in particular for constant k – the assumption in Thm. 1 reads $np \geq (1+\varepsilon)k\ln(n)$. The algorithm `Color`, which we will present in Sec. 2, improves on the result of Feige and Kilian in several respects.

- In contrast to `Color`, the algorithm in [9] does not seem to be able to handle the case that k grows as a function of n (at least the analysis of the SDP rounding techniques breaks down). In contrast, choosing $p = 1/2$ we can make k as large as $\Omega(\sqrt{n})$ in Thm. 1.
- The algorithm `Color` is much simpler. For instance, it needs to solve an SDP only once, whereas [9] requires several SDP computations. However, the techniques of [9] apply to further problems that are not addressed in this paper (e.g. "maximum independent set").
- Instead of just producing a k-coloring of $G = G^*_{n,p,k}$ whp., `Color` also provides a proof that the output is indeed optimal.

The basic observation behind `Color` is that in $G = G^*_{n,p,k}$ whp. all optimal solutions to a certain SDP relaxation of the chromatic number are *integral,* i.e. encode colorings of G. Though it also uses SDP, the algorithm of Feige and Kilian relies on different techniques. The phenomenon that optimal fractional solutions are integral whp. has also been observed in the context of the minmum bisection and the maximum independent set problem [4,9,10].

Theorem 2. *Let* $3 \leq k \leq n^{99/100}$. *There is no polynomial time algorithm that for* $np \leq (1-\varepsilon)\frac{k}{2}\ln(n/k)$ k*-colors* $G^*_{n,p,k}$ *whp., unless NP⊂RP.*

Note that for $k = o(\ln n)$, Thm. 2 implies that the positive result Thm. 1 is essentially best possible (up to a factor of 2). The theorem improves by a factor of $\frac{k}{2}$ on a hardness result given in [9], where it is shown that it is NP-hard to k-color $G^*_{n,p,k}$ if $np \leq (1-\varepsilon)\ln(n)$. The proof of Thm. 2 is omitted.

1.3 Graph Coloring in Expected Polynomial Time

In addition to heuristics that always have a polynomial running time and perform well whp., we shall study coloring algorithms with polynomial expected running time. How does such an algorithm work? Imagine the quest of the algorithm for a solution as a search tree. Since the algorithm is supposed to work on *all* instances properly, this search tree can be of polynomial or exponential size, or anything in between. Hence, in order to ensure that the *average* size of the search tree is polynomial, on the one hand we need algorithmic techniques that are robust enough to result in a small search tree on the vast majority of instances. On the other hand, the analysis will trade the probability that the instance is "atypical" to a certain degree against the size of the resulting search tree (e.g. in [2] such an approach has been carried out for the Knapsack problem). With respect to graph coloring, we shall prove that the optimal solutions to a certain SDP relaxation of the chromatic number are extremely likely to be "close to" integral, and show how to extract a coloring from such fractional solutions. These methods lead to the following result.

Theorem 3. *Suppose that* $k = k(n)$ *and* $p = p(n)$ *are such that* $np \geq C_0 \max\{k \cdot \ln(n), k^2\}$ *for a certain constant* C_0. *There is an algorithm* `ExpColor` *that* k-*colors any* k-*colorable input graph and that applied to* $G^*_{n,p,k}$ *has polynomial expected running time.*

Thm. 3 improves on an algorithm suggested in [5], which breaks down in the case that the number of color classes $k = k(n)$ grows as a function of n. In fact, the expected running time in [5] is $n^{\Theta(k)}$, which is not polynomial if $k = k(n) \to \infty$. By contrast, the expected running time of `ExpColor` is polynomial in both n and k. Furthermore, the algorithm in [5] needs that $np \geq \omega \ln(n)$ where $\lim_{n\to\infty} \omega(n) = \infty$, so that `ExpColor` requires fewer random edges. In addition to these technical points, the algorithm for Thm. 3 gives new insight in why SDP is a good approach to color semirandom graphs.

The next theorem shows that for only slightly larger values of p than in Thm. 3, we can actually find an *optimal* coloring in polynomial expected time.

Theorem 4. *Suppose that* $k = k(n)$ *and* $p = p(n)$ *are such that* $np \geq C_0 \max\{k \cdot \ln(n), k^2 \ln(k)\}$ *for a certain constant* C_0. *There is an algorithm* `OptColor` *that colors any input graph optimally, and that applied to* $G^*_{n,p,k}$ *has polynomial expected running time.*

Thm. 4 improves on Subramanian's result [18] that $G^*_{n,p,k}$ can be colored optimally in polynomial expected time if $np \geq n^{\alpha_k}$ (cf. Sec. 1.1). Moreover, Thm. 4 also improves on Subramanian's result that *random* graphs $G_{n,p,k}$ can be colored optimally in polynomial expected time if k is constant and $np \geq n^\epsilon$ [19]. (The problem of extending this result to smaller values of p has also been posed by Krivelevich [14].)

Observe that Thm. 2 implies that Thms. 3 and 4 are best possible for $k \leq \ln(n)/\ln\ln(n)$, up to the precise value of the constant C_0. We will present `ExpColor` and `OptColor` in Sec. 3

1.4 Notation

Throughout, we let $V = \{1, \ldots, n\}$. If $G = (V, E)$ is a graph and $X \subset V$, then $N(X) = N_G(X)$ denotes the *neighborhood* of X (which may intersect X). Moreover, $\bar{N}(X) = \bar{N}_G(X) = V \setminus N(X)$. Furthermore, $G[X]$ signifies the subgraph of G induced on X. Often we let $V(G)$, $E(G)$ denote the vertex set and the edge set of G. If ξ, η are vectors, then $\langle \xi, \eta \rangle$ denotes their scalar product.

2 A Simple Heuristic for Finding an Optimal Coloring

The algorithm `Color` for Thm. 1 employs a SDP relaxation $\bar{\vartheta}_2$ of the chromatic number, which has been studied by Szegedy [20]. Let us recall the definition of $\bar{\vartheta}_2$. A *rigid vector* k*-coloring* of a graph $G = (V, E)$ is a family $(x_v)_{v \in V}$ of unit vectors in $\mathbb{R}^n$ such that $\langle x_v, x_w \rangle \geq -\frac{1}{k-1}$ for all $v, w \in V$, and $\langle x_v, x_w \rangle = -\frac{1}{k-1}$ for all $\{v, w\} \in E$. Let $\bar{\vartheta}_2(G) = \inf\{k > 1 |\ G$ admits a rigid vector k-coloring$\}$.

To recall a proof that $\bar{\vartheta}_2(G) \leq \chi(G)$, let $k = \chi(G)$, and let $(x_i^*)_{i=1,\ldots,k}$ be a family of unit vectors in $\mathbb{R}^k$ such that $\langle x_i^*, x_j^* \rangle = -\frac{1}{k-1}$ for $i \neq j$. Let $V_1, \ldots, V_k$ be a k-coloring of G. Set $x_v = x_i^*$ for all $v \in V_i$. Then, $(x_v)_{v \in V}$ is a rigid vector k-coloring of G, whence $\bar{\vartheta}_2(G) \leq k$.

Algorithm 5. `Color`(G)
Input: A graph $G = (V, E)$. *Output:* Either a $\chi(G)$-coloring of G or "fail".

1. Compute $\bar{\vartheta}_2(G)$ along with a rigid vector $\bar{\vartheta}_2(G)$-coloring $(x_v)_{v \in V}$.
2. Let $H = (V, F)$ be the graph with edge set $F = \{\{v, w\} |\ \langle x_v, x_w \rangle \leq 0.995\}$. Apply the greedy algorithm for graph coloring to H. Let $\mathcal{C}$ be the resulting coloring.
3. If $\mathcal{C}$ uses at most $\lceil \bar{\vartheta}_2(G) \rceil$ colors, then output $\mathcal{C}$ as a coloring of G. Otherwise, output "fail".

In summary, `Color`(G) computes the rigid vector coloring $(x_v)_{v \in V}$ (this can be done in polynomial time via SDP [12]) to construct an auxiliary graph in which two vertices v, w are adjacent iff their distance $\|x_v - x_w\|$ is at least 0.1, i.e. if v and w are "far apart". To this graph H, `Color` applies the simple greedy algorithm that goes through the vertices V in a fixed order and colors each vertex v with the least color among $\{1, \ldots, n\}$ not yet used by the neighbors of v.

To show that `Color` either finds an optimal coloring of the input graph G or outputs "fail", note that the graph H constructed in Step 2 contains G as a subgraph. For if $\{v, w\} \in E$, then $\langle x_v, x_w \rangle \leq 0$. Since $\chi(G) \geq \bar{\vartheta}_2(G)$, $\mathcal{C}$ is an optimal coloring of G if it uses at most $\lceil \bar{\vartheta}_2(G) \rceil$ colors.

To prove Thm. 1, it remains to show that `Color`$(G = G^*_{n,p,k})$ outputs an optimal coloring whp. Thus, let $V_1, \ldots, V_k$ be the k-coloring planted in G. Directed by the above proof that $\bar{\vartheta}_2(G) \leq \chi(G)$, we call a rigid vector k-coloring $(x_v)_{v \in V}$ *integral* if there are vectors $(x_i^*)_{i=1,\ldots,k}$ such that $x_v = x_i^*$ for all $v \in V_i$, and $\langle x_i^*, x_j^* \rangle = -\frac{1}{k-1}$ for $i \neq j$. If the rigid vector coloring $(x_v)_{v \in V}$ computed in Step 1 is integral, then the graph H constructed in Step 2 of `Color` is a complete k-partite graph with color classes $V_1, \ldots, V_k$. Consequently, the greedy algorithm

succeeds in k-coloring H. Hence, if also $\bar{\vartheta}_2(G) = k$, then `Color` finds an optimal coloring. Thus, the remaining task is to establish the following lemma. Throughout, we assume that $np \geq \max\{(1+\varepsilon)k \ln n, C_0 k^2\}$.

Lemma 6. *Let $G = G^*_{n,p,k}$. Whp. we have $\bar{\vartheta}_2(G) = k$, and every rigid vector k-coloring of G is integral (w.r.t. the planted k-coloring $V_1, \ldots, V_k$).*

To prove L. 6, we consider the following SDP from Frieze and Jerrum [11]:

$$\mathrm{SDP}_h(G) = \max \sum_{\{v,w\} \in E} \frac{h-1}{h}(1 - \langle x_v, x_w \rangle) \text{ s.t. } \langle x_v, x_w \rangle \geq -\frac{1}{h-1} \forall v, w \in V$$

where the max is taken over all families $(x_v)_{v \in V}$ of unit vectors in $\mathbb{R}^n$ and $2 \leq h \in \mathbb{R}$. Note that if G is k-colorable, then plugging a rigid vector k-coloring $(x_v)_{v \in V}$ into SDP_k shows that $\mathrm{SDP}_k(G) = \#E(G)$. Furthermore, SDP_h is *monotone*: if G' contains G as a subgraph, then $\mathrm{SDP}_h(G') \geq \mathrm{SDP}_h(G)$. The proof of the next lemma is based on SDP duality (details omitted).

Lemma 7. *Whp. the semirandom graph $G = G^*_{n,p,k}$ enjoys the following property. Let G' be a graph obtained by adding an edge $\{v, w\}$ to G, where $v, w \in V_i$ for some i. Let $2 < h \leq k$. Then $\mathrm{SDP}_h(G') \leq \#E(G) - \Omega\left(\frac{n^2 p}{hk}\right)(k-h)$.*

Proof of L. 6. To prove that $\bar{\vartheta}_2(G^*_{n,p,k}) = k$ whp., let $G = G^*_{n,p,k}$, and assume that $\bar{\vartheta}_2(G) = h < k$. Let $(x_v)_{v \in V}$ be a rigid vector h-coloring of G. Then $(x_v)_{v \in V}$ is a feasible solution to SDP_h, whence $\mathrm{SDP}_h(G) = \#E(G)$. However, by L. 7 we have $\mathrm{SDP}_h(G) < \#E(G)$ whp. Thus, $\bar{\vartheta}_2(G) = k$ whp.

Finally, to show that any rigid vector k-coloring $(x_v)_{v \in V}$ of $G = G^*_{n,p,k}$ is integral whp., suppose that G has the property stated in L.7. Let $s, t \in V_i^*$, and let G' be the graph obtained from G by adding the edge $\{s, t\}$. Then

$$\#E(G) = \mathrm{SDP}_k(G) = \sum_{\{v,w\} \in E(G)} \frac{h-1}{h}(1 - \langle x_v, x_w \rangle)$$

$$\leq \frac{h-1}{h}\left(1 - \langle x_s, x_t \rangle + \sum_{\{v,w\} \in E(G)} 1 - \langle x_v, x_w \rangle\right) \leq \mathrm{SDP}_k(G') \leq \#E(G)$$

implies that $\langle x_s, x_t \rangle = 1$, whence $x_s = x_t$. Consequently, there are unit vectors x_i^* such that $x_v = x_i^*$ for all $v \in V_i$, $i = 1, \ldots, k$. Furthermore, if $i \neq j$, then whp. there are vertices $v \in V_i$, $w \in V_j$ such that $\{v, w\} \in E(G)$. Therefore, $\langle x_i^*, x_j^* \rangle = \langle x_v, x_w \rangle = -\frac{1}{k-1}$, thereby proving that $(x_v)_{v \in V}$ is integral. □

3 Coloring $G^*_{n,p,k}$ in Polynomial Expected Time

First, we present the algorithm `ExpColor` for Thm. 3. Then, in Sec. 3.5, we indicate how to obtain the algorithm `OptColor` for Thm. 4. Throughout, we let $G = G^*_{n,p,k}$, and let $V_1, \ldots, V_k$ denote the planted coloring of G. Moreover, we assume that $np \geq C_0 \max\{k \ln n, k^2\}$. For $U \subset \{1, \ldots, k\}$, let $V_U = \bigcup_{i \in U} V_i$.

3.1 The Algorithm ExpColor: Outline

In order to k-color G, **ExpColor**(G,k) runs the procedure Classes, which proceeds recursively in k stages. In each stage, Classes tries to recover one of the color classes $V_1,\dots,V_k$, and then hands the graph without the recovered color class to the next stage. More precisely, if W_l is the set of vertices that have not yet been colored in the previous stages, then the l'th stage tries to exhibit a set $\mathcal{S}_l$ of large independent sets of $G[W_l]$. Then, for each $S_l \in \mathcal{S}_l$, Classes passes the graph $G[W_l \setminus S_l]$ to stage $l+1$, which tries to find a $(k-l)$-coloring of this graph. If G is "typical", which happens with high probability, then each $\mathcal{S}_l$ will consist precisely of one color class, so that a k-coloring will be found immediately.

However, since our goal is an algorithm that k-colors *all* k-colorable graphs, we also have to deal with "atypical" input instances G. To this end, ExpColor uses the variable T, which controls the size of the "search tree" that ExpColor is building, i.e. what amount of running time ExpColor spends in order to k-color G. This amount of time is distributed among the k stages of Classes via the variables $\eta_1,\dots,\eta_k$; i.e. stage l may spend time $(n\binom{n/k}{\eta_l})^{O(1)}$ to (try to) produce a set $\mathcal{S}_l$ that contains one of the hidden color classes.

Algorithm 8. **ExpColor**(G,k)
Input: A graph $G = (V, E)$, an integer $k \geq \chi(G)$. *Output:* A k-coloring of G.

1. For $T = 1,\dots,\lfloor \exp(n/\ln k)\rfloor$ do
2. For $\eta = 0,1,\dots,\eta_{\max} = \max\{\xi \in \mathbb{Z} \mid \exp(\xi), \binom{n/k}{\xi} \leq T\}$ do
3. For each decomposition $\eta = \eta_1 + \cdots + \eta_k$, where $0 \leq \eta_i \leq \frac{n}{2k}$ are integers, and $\prod_{i=1}^{k}\binom{n/k}{\eta_i} \leq T$ do
4. Run **Classes**$(G,V,k,\eta_1,\dots,\eta_k)$. If Classes k-colors G successfully, then output the coloring and halt.
5. Run **Exact**(G,k).

After exhibiting some properties of $G^*_{n,p,k}$ in Sec. 3.2, we describe Classes in Sec. 3.3. Finally, in Sec. 3.4, we deal with the procedure Exact.

3.2 Preliminaries

Let $G = G^*_{n,p,k}$. The following lemma is a consequence of estimates on the probable value of SDP_k on $G_{n,p}$ from [6].

Lemma 9. *Let $U \subset \{1,\dots,k\}$, $\#U = u$. With probability ("w.p.") $\geq 1 - \exp(-100nu/k)$ the graph G enjoys the following property.*

Let G' be a graph obtained from G by adding each edge inside the color classes V_i with probability p independently. Then for a certain constant $C_1 > 0$ we have

$$\mathrm{P}\left(\mathrm{SDP}_u(G'[V_U]) \leq \#E(G[V_U]) + C_1\frac{nu}{k}\sqrt{np}\right) \geq 2/3, \tag{1}$$

where probability is taken over the choice of the random edges inside the color classes.

Whp. the bipartite graph consisting of the edges joining a color class V_i with $V \setminus V_i$ is a good expanding graph. Indeed, we define the *defect* $\eta_i(G)$ of V_i as follows. If there is some $U \subset V_i$, $\#U \geq \frac{n}{2k}$, such that $\#\bar{N}_G(U) \setminus V_i > \frac{300}{p}$, then we let $\eta_i(G) = \frac{n}{2k}$. Otherwise,

$$\eta_i(G) = \min\{\max\{d\#T - \#N(T) \cap V_i| \; T \subset V \setminus V_i, \; \#T \leq \frac{n}{2kd}, \\ 6 \leq d \leq \lceil 50k \rceil\}, \frac{n}{2k}\}.$$

The smaller the defect is, the better the expansion.

Lemma 10. *Let* $\eta_i \geq 0$. *Then* $\mathrm{P}(\eta_i(G) \geq \eta_i \text{ for } i = 1, \ldots, k) \leq \prod_{i=1}^{k} \binom{n/k}{\eta_i}^{-100}$.

Lemma 11. *W.p.* $\geq 1 - \exp(-100n)$ *the following property holds.*

If U is an independent set in G, and $\#U \geq \frac{n}{100k}$, then there is some index i such that $\#U \cap V_i > \frac{199}{200}\#U$. (2)

Moreover, w.p. $\geq 1 - \exp(-100n/\ln k)$ *the following holds for all* $i \in \{1, \ldots, k\}$.

If $U \subset V_i$, $\#U \geq \frac{n}{2k\ln(k)}$, then $\#\bar{N}(U) \leq \frac{2n}{k}$. (3)

3.3 The Procedure `Classes`

The input of `Classes` consists of the graph G, a set $W \subset V(G)$, the number k, and integers $\eta_1, \ldots, \eta_l$. `Classes` is to find an l-coloring of $G[W]$. In Steps 1-4, `Classes` computes a set $\mathcal{S}_l$ of independent sets of $G_l = G[W]$, each of cardinality n/k. Then, in Steps 5–6, `Classes` tentatively colors each of the sets $S_l \in \mathcal{S}_l$ with the l'th color, and calls itself recursively on input $(G, W \setminus S_l, k, \eta_1, \ldots, \eta_{l-1})$ in an attempt to $(l-1)$-color $G[W \setminus S_l]$.

Algorithm 12. `Classes`$(G, W, k, \eta_1, \ldots, \eta_l)$
Input: A graph $G = (V, E)$, a set $W \subset V$, integers $k, \eta_1, \ldots, \eta_l$.
Output: Either an l-coloring of $G[W]$ or "fail".

1. Let $G_l = G[W]$. If $l = 1$ and G_l is an empty graph, then return a 1-coloring of G_l. If $\bar{\vartheta}_2(G_l) > l$, then return "fail". Otherwise, compute a rigid vector l-coloring $(x_v)_{v \in W}$ of G_l.
2. If $\eta < \frac{n}{2k}$
   If for all $w \in W$ the set $S_w = \{u \in W| \; \langle x_u, x_w \rangle \geq 0.99\}$ has cardinality $< \frac{199n}{200k}$, then return "fail". Otherwise, let $v = \min\{w \in W| \; \#S_w \geq \frac{199n}{200k}\}$. Let $\mathcal{S}_l = \mathtt{Purify}(G, S_v, \eta_l, k)$.
3. else
   Let $\mathcal{S}_l = \emptyset$. For each $U \subset W$, $\#U = \frac{n}{2k\ln(k)}$, do
   Let $T = \bar{N}_{G_l}(U)$. If $\#T \leq 2n/k$, then for all $I \subset T$, $\#I = n/k$, do
   If I is an independent set, then add I to $\mathcal{S}_l$.

4. For each $S_l \in \mathcal{S}_l$ do
5. Run `Classes`$(G, W \setminus S_l, k, \eta_1, \ldots, \eta_{l-1})$. If `Classes` succeeds in $l-1$-coloring G_{l-1}, return the l-coloring of G_l obtained by coloring S_l with an l'th color.
6. Return "fail".

Suppose that the input graph G is a semirandom graph $G^*_{n,p,k}$ with hidden coloring $V_1, \ldots, V_k$. Similarly as `Color`, `Classes` employs the relaxation $\bar{\vartheta}_2$ of the chromatic number, but in a more sophisticated way. If $\eta_l < \frac{n}{2k}$, then Step 2 of `Classes` tries to use the rigid vector coloring $(x_v)_{v \in W}$ to recover a large independent set S_v of the input graph, cf. L. 14 below. By L. 11, with extremely high probability S_v consists mainly of vertices of a certain color class V_i. Then, to recover V_i from S_v, `Classes` uses the procedure `Purify`, which we will describe below. On the other hand, if $\eta_l \geq \frac{n}{2k}$, then Step 3 of `Classes` tries to recover a color class in time $\exp(O(n/k))$.

Proposition 13. *To each semirandom graph $G = G^*_{n,p,k}$ that satisfies Properties (2) and (3) we can associate a sequence $\eta^* = (\eta_1^*, \ldots, \eta_k^*) \in \{0, 1, \ldots, \frac{n}{2k}\}^k$ such that the following two conditions hold.*

1. *`Classes`$(G, V, k, \eta_1^*, \ldots, \eta_k^*)$ outputs a k-coloring of G.*
2. *Let $\eta_1, \ldots, \eta_k \geq 0$. Then $\mathrm{P}(\eta_i^* \geq \eta_i \text{ for all } i) \leq \prod_{i=1}^k \binom{n/k}{\eta_i}^{-90}$.*

The running time of `Classes`$(G, V, k, \eta_1, \ldots, \eta_k)$ is at most $n^{O(1)} \prod_{i=1}^k \binom{n/k}{\eta_i}^{16}$.

The crucial observation behind `Classes` is that we can use the rigid vector coloring to recover a large independent set. The basic idea is as follows. Imagine that we would throw random edges into the color classes of $G = G^*_{n,p,k}$ by including the edges inside the color classes V_i with probability p independently. (Of course, the *algorithm* can't do this, because it does not know the color classes yet.) Let G' be the resulting graph. How do $\mathrm{SDP}_k(G)$ and $\mathrm{SDP}_k(G')$ compare? By L. 9, $\mathrm{SDP}_k(G')$ exceeds $\mathrm{SDP}_k(G)$ by at most $O(n^{3/2}p^{1/2})$ w.p. $\geq 2/3$, because $\mathrm{SDP}_k(G) = \#E(G)$. Hence, considering a rigid vector k-coloring $(x_v)_{v \in V}$ of G, there are only $O(n^{3/2}p^{1/2})$ random edges $\{v, w\}$ inside the color classes V_i such that $\{v, w\}$ contributes "much" to $\mathrm{SDP}_k(G')$ (say, $1 - \langle x_v, x_w \rangle \geq 1/200$). But then there must be at least one color class such that for almost all vertices v in this color class the vectors x_v are "close to each other". In fact, these vertices can be found easily by "guessing" one of them and considering all the vertices that are close to it. The following lemma makes this idea rigorous.

Lemma 14. *Let $G = G^*_{n,p,k}$. Assume that Property (1) holds for the set $U \subset \{1, \ldots, k\}$, $\#U = u > 1$, and let $(x_v)_{v \in V_U}$ be a rigid vector u-coloring of $G[V_U]$. Then there is a vertex $v \in V_U$ such that $S_v = \{w \in V_U \mid \langle x_v, x_w \rangle \geq 0.99\}$ is an independent set of cardinality $\geq \frac{199n}{200k}$ in G.*

Proof. Consider the graph $H = (V_U, F)$, where $F = \{\{v, w\} \mid \langle x_v, x_w \rangle < 0.99\}$. Then $G[V_U]$ is a subgraph of H. Let $\mathcal{B} = \bigcup_{i \in U} E(H[V_i])$ be the set of all edges

of H that join two vertices that belong to the same color class of G. Let $b = \#\mathcal{B}$. Furthermore, let G' be the random graph obtained from G by including each V_i-V_i-edge with probability p independently for all $i \in \{1, \ldots, k\}$. Note that $(x_v)_{v \in V}$ is a feasible solution to SDP_u. Hence, by Property (1), with probability $\geq 2/3$ taken over the choice of the random edges inside the color classes we have

$$\sum_{\{v,w\} \in E(G'[V_U])} \frac{u-1}{u} (1 - \langle x_v, x_w \rangle) \leq \mathrm{SDP}_u(G'[V_U]) \leq \#E(G[V_U]) + C_1 \frac{nu}{k} \sqrt{np}.$$

Observe that an edge $e = \{v, w\}$ of $G'[V_U]$ contributes 1 to the sum on the left hand side if $e \in E(G[V_U])$, and that e contributes $\geq \frac{1}{200}$ if $e \in \mathcal{B}$. Therefore,

$$\mathrm{P}\left(\#\mathcal{B} \cap E(G'[V_U]) \leq 200 C_1 \frac{nu}{k} \sqrt{np}\right) \geq \frac{2}{3}. \tag{4}$$

Since $\#\mathcal{B} \cap E(G'[V_U])$ is binomially distributed with mean bp, we have

$$\mathrm{P}\left(\#\mathcal{B} \cap E(G'[V_U]) \geq \frac{bp}{10}\right) \geq \frac{1}{2}.$$

Therefore, (4) yields that $bp \leq 2000 C_1 \frac{nu}{k} \sqrt{np}$. As $np \geq C_0 k^2$ for a large constant $C_0 > 0$, we conclude that $b \leq \frac{u}{401} \left(\frac{n}{k}\right)^2$. Thus, there is some $i \in U$ and some vertex $v \in V_i$ such that v has degree $< \frac{n}{200k}$ in $H[V_i]$. Let $S = \{w \in V_i \mid \{v, w\} \notin F\}$. Then for all $u, w \in S$ we have $\langle x_u, x_w \rangle \geq 0$. Consequently, S is an independent set of cardinality $\geq \frac{199n}{200k}$ in G. □

Step 2 of `Classes` employs a procedure `Purify` that uses network flow techniques from [7].

Algorithm 15. `Purify`(G, I, η, k)
Input: A graph $G = (V, E)$, integers η, k, $I \subset V$. *Output:* A set $\mathcal{S}$ of subsets of V.

1. Let $\mathcal{S} = \emptyset$. If $\#I > 2n/k$, then return $\emptyset$. Otherwise, for all $D \subset I$, $\#D \leq \eta$ do
2. Construct the following network N:
   - The vertices of N are s, t, s_v for $v \in I \setminus D$, and t_w for $w \in V$.
   - The arcs of N are (s, s_v) for $v \in I \setminus D$, (t_w, t) for $w \in V$, and (s_v, t_w) if $\{v, w\} \in E$.
   - The capacity c is given by $c(s, s_v) = \lceil 50k \rceil$, $c(t_w, t) = 1$, $c(s_v, t_w) = 1$ if $\{v, w\} \in E$.

   Compute a maximum integer flow f in N, let $L = \{v \in I \setminus D \mid f(s, s_v) = c(s, s_v)\}$, and set $I' = I \setminus (L \cup D)$.
3. If $\tilde{V} = V \setminus N(I')$ satisfies $\#\tilde{V} \leq \frac{2n}{k}$ then
4. For each set $Y \subset \tilde{V}$, $\#Y \leq 6\eta$, such that $I' \cup Y$ is an independent set of cardinality n/k add $I' \cup Y$ to $\mathcal{S}$.
5. For all $D' \subset \tilde{V}$, $\#D' \leq \eta$, do

Let $I'' = I'$. For $\tau = 0, 1, \ldots, \lceil \log_2(n) \rceil$ do
Let $V' = \tilde{V} \setminus D'$. Construct the following network N.
- The vertices of N are s, t, s_v for $v \in V' \setminus I''$, and t_w for $w \in V'$.
- The arcs of N are (s, s_v) for $v \in V' \setminus I''$, (t_w, t) for $w \in V'$, and (s_v, t_w) if $\{v, w\} \in E$.
- The capacities are $c(s, s_v) = 6$, $c(t_w, t) = 1$, $c(s_v, t_w) = 1$ if $\{v, w\} \in E$.

Compute a maximum integer flow f in N. Let $L = \{v \in V' \setminus I'' | f(s, s_v) = c(s, s_v)\}$ and $I'' = V' \setminus L$. If I'' is an independent set of cardinality n/k then add I'' to $\mathcal{S}$.

6. Output $\mathcal{S}$.

The proof of Prop. 13 relies on the following proposition, which summarizes the analysis of `Purify`.

Proposition 16. *Let $G = G^*_{n,p,k}$. Let $i \in \{1, \ldots, k\}$. Suppose that I is an independent set that satisfies $\#I \cap V_i \geq \frac{99n}{100k}$. Further, assume that $\eta_i(G) \leq \eta < \frac{n}{2k}$. Then the output $\mathcal{S}$ of* **Purify**(G, I, η, k) *contains V_i as an element.*

If the assumptions in Prop. 16 hold, then the set I' contains most vertices of some color class V_i and only few vertices not in V_i. `Purify`(G, I, η, k) proceeds in two phases. In the first phase (Steps 2–3), `Purify` tries to remove the vertices in $I \setminus V_i$, thereby obtaining I'. In the second phase (Steps 4–5), we enlarge $I' \subset V_i$ several times, in order to recover V_i. This general approach as well as the flow techniques in `Purify` build on ideas from [9]. The proof of Prop. 16 is based on the expansion of the bipartite graph consisting of the V_i-$V \setminus V_i$-edges.

3.4 The Procedure `Exact`

The idea behind the procedure `Exact` is to "guess" a certain part of the hidden coloring of $G = G^*_{n,p,k}$. Since `Exact` does not contribute essential ideas to `ExpColor`, we omit the details. The analysis of `Exact` entails the following result.

Proposition 17. *For every k-colorable graph G,* **Exact**(G, k) *finds a k-coloring. The probability that the running time of* **Exact**$(G^*_{n,p,k}, k)$ *exceeds $n^{O(1)}T$ for some $T \geq \exp(n/\ln(k))$ is at most T^{-3}.*

Combining Prop. 13 and 17 with L. 9 and 11, it is not hard to see that the expected running time of **ExpColor**$(G^*_{n,p,k}, k)$ is polynomial. Moreover, Prop. 17 shows that if G is a k-colorable graph, then `ExpColor`(G) will find a k-coloring.

3.5 Finding an Optimal Coloring

Since `ExpColor` does not provide a certificate that the coloring found is optimal, this algorithm does not yet satisfy the requirements of Thm. 4. In the case that $k \leq \ln(n)/\ln\ln(n)$, this can be corrected easily: Combining L. 7 with a

large deviation result on $\mathrm{SDP}_{k-\frac{1}{2}}(G^*_{n,p,k})$ given in [6] and invoking a similar argument as in the proof of L. 6, we obtain $\mathrm{P}(\bar{\vartheta}_2(G^*_{n,p,k}) \leq k-\frac{1}{2}) \leq \exp(-100n)$. Hence, we can first compute $\kappa = \lceil \bar{\vartheta}_2(G^*_{n,p,k}) \rceil$, and then apply $\mathtt{ExpColor}(G,\kappa)$. If $\mathtt{ExpColor}(G,\kappa)$ succeeds in κ-coloring G, we are done. Otherwise, we run Lawler's algorithm [17] to find an optimal coloring in time $O(\exp(n))$. In the case of general k, a similar approach works.

4 Conclusion

An interesting open problem might be to improve on the values of p and k for which coloring $G^*_{n,p,k}$ is easy/hard given in Thms. 1 and 2. While for $k = o(\ln n)$ the situation is rather clear (up to a constant factor), for larger values of k the gap between the upper bound in Thm. 1 and the lower bound in Thm. 2 diverges. For instance (cf. also [14]): is it possible to k-color $G^*_{n,\frac{1}{2},k}$ for $k \gg \sqrt{n}$?

The coloring algorithms in this paper are based on the fact that the optimal solutions to certain SDP relaxations are (almost) "integral" whp. Similar observations hold for SDP relaxations for various further problems such as coloring 2-colorable hypergraphs, MAX 3-Not-All-Equal-SAT, or Densest k-Subgraph.

References

1. Alon, N., Kahale, N.: A spectral technique for coloring random 3-colorable graphs. SIAM J. Comput. **26** (1997) 1733–1748
2. Beier, R., Vöcking, B.: Random Knapsack in expected polynomial time. Proc. 35th STOC (2003) 232–241
3. Blum, A., Spencer, J.: Coloring random and semirandom k-colorable graphs. J. of Algorithms **19** (1995) 203–234
4. Boppana, R.: Eigenvalues and graph bisection: An average-case analysis. Proc. 28th FOCS (1987) 280–285
5. Coja-Oghlan, A.: Finding sparse induced subgraphs of semirandom graphs. Proc. 6th RANDOM (2002) 139–148
6. Coja-Oghlan, A., Moore, C., Sanwalani, V.: MAX k-CUT and approximating the chromatic number of random graphs. Proc. 30th ICALP (2003) 200–211
7. Coja-Oghlan, A.: Finding large independent sets in polynomial expected time. Proc. 20th STACS (2003) 511–522
8. Engebretsen, L., Hohnerin, J.: Towards optimal lower bounds for clique and chromatic number. TCS **299** (2003) 537–584
9. Feige, U., Kilian, J.: Heuristics for semirandom graph problems. JCSS **63** (2001) 639–671
10. Feige, U., Krauthgamer, J.: Finding and certifying a large hidden clique in a semirandom graph. Random Structures & Algorithms **16** (2000) 195–208
11. Frieze, A., Jerrum, M.: Improved approximation algorithms for MAX k-CUT and MAX BISECTION. Algorithmica **18** (1997) 61–77.
12. Grötschel, M., Lovász, L., Schrijver, A.: Geometric algorithms and combinatorial optimization. Springer (1988)
13. Janson, S., Łuczak, T., Ruciński, A.: Random Graphs. Wiley (2000)

14. Krivelevich, M.: Coloring random graphs – an algorithmic perspective, Proc. 2nd MathInfo (2002) 175-195.
15. Kučera, L.: Expected behavior of graph coloring algorithms. Proc. 1st FCT (1977) 447–451
16. Kučera, L.: Graphs with small chromatic number are easy to color. Information Processing Letters **30** (1989) 233–236
17. Lawler, E.L.: A note on the complexity of the chromatic number problem, Information Processing Letters **5** (1976) 66–67
18. Subramanian, C.R.: Minimum coloring random and semirandom graphs in polynomial average time. J. of Algorithms **33** (1999) 112–123
19. Subramanian, C.R.: Coloring sparse random graphs in polynomial average time. Proc. 8th ESA (2000) 415–426
20. Szegedy, M.: A note on the θ number of Lovász and the generalized Delsarte bound. Proc. 35th FOCS (1994) 36–39

Sublinear-Time Approximation for Clustering Via Random Sampling*

Artur Czumaj[1] and Christian Sohler[2]

[1] Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA. czumaj@cis.njit.edu

[2] Heinz Nixdorf Institute and Department of Computer Science, University of Paderborn, D-33102 Paderborn, Germany, csohler@uni-paderborn.de

Abstract. In this paper we present a novel analysis of a random sampling approach for three clustering problems in metric spaces: ***k-median***, *min-sum* ***k-clustering***, and *balanced* ***k-median***. For all these problems we consider the following simple sampling scheme: select a small sample set of points uniformly at random from V and then run some approximation algorithm on this sample set to compute an approximation of the best possible clustering of this set. Our main technical contribution is a significantly strengthened analysis of the approximation guarantee by this scheme for the clustering problems.

The main motivation behind our analyses was to design *sublinear-time* algorithms for clustering problems. Our second contribution is the development of new approximation algorithms for the aforementioned clustering problems. Using our random sampling approach we obtain for the first time approximation algorithms that have the running time independent of the input size, and depending on k and the diameter of the metric space only.

1 Introduction

The problem of clustering large data sets into subsets (clusters) of similar characteristics has been extensively studied in computer science, operations research, and related fields. Clustering problems arise in various applications, for example, in data mining, data compression, bioinformatics, pattern recognition and pattern classification. In some of these applications massive datasets have to be processed, e.g., web pages, network flow statistics, or call-detail records in telecommunication industry. Processing such massive data sets in more than linear time is by far too expensive and often even linear time algorithms may be too slow. One reason for this phenomenon is that massive data sets do not fit into main memory and sometimes even secondary memory capacities are too low. Hence, there is the desire to develop algorithms whose running times are not only polynomial, but in fact are *sublinear* in n (for very recent survey expositions, see, e.g., [7,16]). In a typical sublinear-time algorithm a subset of the input is selected according to some random process and then processed by an algorithm. With high probability the outcome of this algorithm should be some approximation of the outcome of an exact

* Research partly supported by NSF ITR grant CCR-0313219, NSF grant CCR-0105701, and DFG grant Me 872-8/2.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 396–407, 2004.

algorithm running on the whole input. In many cases the randomized process that selects the sample is very simple, e.g., a uniformly random subset is selected.

In this paper we address the problem of designing *sublinear-time* approximation algorithms using *uniformly random sampling* for clustering problems in metric spaces. We consider three clustering problems: the ***k-median** problem,* the *min-sum **k-clustering** problem,* and the *balanced **k-median** problem.* Given a finite metric space (V, μ), the ***k-median** problem* is to find a set $C \subseteq V$ of k-centers that minimizes $\sum_{p \in V} \mu(p, C)$, where $\mu(p, C)$ denotes the distance from p to the nearest point in C. The *min-sum **k-clustering** problem* for a metric space (V, μ) is to find a partition of V into k subsets $C_1, \ldots, C_k$ such that $\sum_{1 \le i \le k} \sum_{p,q \in C_i} \mu(p, q)$ is minimized. The *balanced **k-median** problem* (which is perhaps less standard than the other two problems) for a metric space (V, μ) is to find a set $\{c_1, \ldots, c_k\} \subseteq V$ of k-centers and a partition of V into k subsets $C_1, \ldots, C_k$ that minimizes $\sum_{1 \le i \le k} |C_i| \cdot \sum_{p \in C_i} \mu(p, c_i)$.

For all these three clustering problems we study the following "simple sampling" algorithm: pick a random sample S of points, run an approximation algorithm for the sample, and return the clustering induced by the solution for the sample. The main goal of this paper is to design a generic method of analyzing this sampling scheme and to obtain a significantly stronger quantitative bounds for the performance of this method. Using our approach, for a large spectrum of input parameters we obtain *sublinear-time algorithms* for the three clustering problems above. These are the first approximation algorithms for these problems whose running time is *independent of the input size,* $|V|$.

1.1 Previous Research

***k*-median.** The k-median clustering problem is perhaps the most studied clustering problem in the literature, both, in theoretical research and in applications. It is well known that the k-median clustering in metric spaces is $\mathcal{NP}$-hard and it is even $\mathcal{NP}$-hard to approximate within a factor of $1 + \frac{2}{e}$ [13]. There exist polynomial time approximation algorithms with constant approximation ratios [2,4,5,11,14,17]. When the underlying space is the Euclidean plane, Arora et al. [1] obtained even a PTAS for k-median (extension to higher dimensions and improvements in the running time have been obtained in [15], and more recently in [10]). The k-median problem has been also extensively investigated in the data stream model, see e.g., recent works in [6,10].

There exist a few sublinear-time algorithms for the k-median problem, that is algorithms with the running time of $o(n^2)$ (if we consider an arbitrary metric space (V, μ) with $|V| = n$, then its description size is $\Theta(n^2)$), see, e.g., [11,17,18,19]. The algorithm of Indyk [11] computes in $O(n\,k)$ time a set of $O(k)$ centers whose cost approximates the value of the k-median by a constant factor. Mettu and Plaxton [17] gave a randomized $O(1)$-approximate k-median algorithm that runs in time $O(n(k + \log n))$ subject to the constraint $R = 2^{O(n/\log(n/k))}$, where R denotes the ratio between the maximum and the minimum distance between any pair of distinct points in the metric space. Very recently, Meyerson et al. [18] presented a sublinear-time for the k-median problem under an assumption that each cluster has size $\Omega(n\,k/\epsilon)$; their algorithm requires time $O((k^2/\epsilon)\,\log(k/\epsilon))$ and gives a $O(1)$-approximation guarantee with high probability.

Notice that all the sublinear-time ($o(n^2)$-time) algorithms mentioned above made some assumptions about the input. We follow this approach and in this paper we consider

a model with the diameter of the metric space Δ given, that is, with $\mu : V \times V \to [0, \Delta]$. Such a model has been investigated before by Mishra et al. [19], who studied the quality of k-median clusterings obtained by random sampling. Let $\mathbb{A}_\alpha$ be an arbitrary α-approximation algorithm for k-median. Using techniques from statistics and computational leanjinig theory, Mishra et al. [19] proved that if we sample a set S of $s = \widetilde{O}\left(\left(\frac{\alpha\Delta}{\epsilon}\right)^2 (k \ln n + \ln(1/\delta))\right)$ points from V i.u.r. *(independently and uniformly at random)* and run algorithm $\mathbb{A}_\alpha$ to find an approximation of k-median for S, then with probability at least $1 - \delta$ the outputted set of k centers has the *average distance* to the nearest center of at most $2\,\alpha\, med_{avg}(V,k) + \epsilon$, where $med_{avg}(V,k)$ denotes the *average distance* to the k-median C, that is, $med_{avg}(V,k) = \frac{\sum_{v \in V} \mu(v,C)}{n}$. Using this result, Mishra et al. [19] developed a generic sublinear-time approximation algorithm for k-median. If the algorithm $\mathbb{A}_\alpha$ has the running time of $T(s)$, then the resulting algorithm runs in $T(s)$ time for $s = \widetilde{O}\left(\left(\frac{\alpha\Delta}{\epsilon}\right)^2 \cdot (k \ln n + \ln(1/\delta))\right)$ and computes with probability at least $1 - \delta$ a set of k centers such that the *average distance* to the nearest center is at most $2\,\alpha\, med_{avg}(V,k) + \epsilon$. Notice that since there exist $O(1)$-approximation algorithms for k-median with $T(s) = O(s^2)$, this approach leads to an approximation algorithm for the k-median problem whose dependency on n is only $\widetilde{O}(\log^2 n)$, rather than $\Omega(n^2)$ or $\Omega(nk)$ as in the algorithms discussed above. On the other hand, the running time of this algorithm depends on Δ, and as discussed in [19] (see also [17,18]), such a dependency is necessary to obtain this kind of approximation guarantee.

Min-sum k-clustering. The min-sum k-clustering problem was first formulated (for general graphs) by Sahni and Gonzales [21]. There is a 2-approximation algorithm by Guttman-Beck and Hassin [9] with running time $n^{O(k)}$. Recently, Bartal et al. [3] presented an $O(\frac{1}{\epsilon}\log^{1+\epsilon} n)$-approximation algorithm with $O(n^{1/\epsilon})$ running time and then Fernandez de la Vega et al. [8] gave an $(1+\epsilon)$-approximation algorithm with the running time of $O(n^{3k} \cdot 2^{O((1/\epsilon)^{k^2})})$. For point sets in the $\mathbb{R}^d$, Schulman [20] introduced an algorithm for distance functions ℓ_2^2, ℓ_1 and ℓ_2 that computes a solution that is either within $(1+\epsilon)$ of the optimum or that disagrees with the optimum in at most an ϵ fraction of points. For the basic case of $k = 2$ (which is complement to the Max-Cut), Indyk [12] gave an $(1+\epsilon)$-approximation algorithm that runs in $O(n^{1+\gamma} \cdot (\log n)^{(1/\epsilon)^{O(1)}})$ time for any $\gamma > 0$, which is sublinear in the full input description size but superlinear in n.

Balanced k-median. It is known that in metric spaces the solution to balanced k-median is to within a factor of 2 of that of min-sum k-clustering, see, e.g. [3, Claim 1]. Therefore, balanced k-median has been usually considered in connection with the min-sum k-clustering problem discussed above. The problem was first studied by Guttman-Beck and Hassin [9] who gave an exact $O(n^{k+1})$-time algorithm and Bartal et al. [3] obtained an $O(\frac{1}{\epsilon}\log^{1+\epsilon} n)$-approximation in time $n^{O(1/\epsilon)}$ based on metric embeddings into HSTs. We are not aware of any sublinear-time algorithm for balanced k-median.

1.2 New Contribution

In this paper we investigate the quality of a simple *uniform sampling* approach to clustering problems and apply our analyzes to obtain new and improved bounds for the running time of clustering algorithms.

We first study the **k-median** problem. Our sampling is identical to the one by Mishra et al. [19], however our analysis is stronger and leads to significantly better bounds. Let $\alpha \geq 1$, $0 < \delta < 1$, and $\epsilon > 0$ be arbitrary parameters. We prove that if we pick a sample set of size $\widetilde{O}(\frac{\Delta \cdot \alpha}{\epsilon^2} \cdot (k + \alpha \ln(1/\delta)))$ i.u.r., then an **α-approximation** of the optimal solution for the sample set yields an approximation of the average distance to the nearest median to within $2(\alpha + \epsilon)\, med_{avg}(V, k) + \epsilon$ with probability at least $1 - \delta$; notice in particular, that this gives the sample size *independent of* n. As noted in [19], it is impossible to obtain a sample complexity independent of both Δ and n.

Comparing our result to the one from [19], we improve the sample complexity by a factor of $\Delta \cdot \log n$ while obtaining a slightly worse approximation ratio of $2(\alpha + \epsilon)\, med_{avg}(V, k) + \epsilon$, instead of $2\alpha\, med_{avg}(V, k) + \epsilon$ as in [19]. However, since the algorithm with the best known approximation guarantee has $\alpha = 3 + \frac{1}{c}$ for the running time of $O(n^c)$ time [2], we significantly improve the running time of [19] for all realistic choices of the input parameters while achieving the same approximation guarantee. As a highlight, we obtain an algorithm that in time $\widetilde{O}((\frac{\Delta k}{\epsilon^2} \cdot (k + \log(1/\delta)))^2)$ —*fully independent of* n — has the average distance to the nearest median at most $O(med_{avg}(V, k)) + \epsilon$ with probability at least $1 - \delta$.

Furthermore, our analysis can be significantly improved if we assume the input points are in Euclidean space $\mathbb{R}^d$. In this case we improve the approximation guarantee to $(\alpha + \epsilon)\, med_{avg}(V, k) + \epsilon$ in the cost of increasing the sample size to $\widetilde{O}(\frac{\Delta \cdot \alpha}{\epsilon^2} \cdot (k\,d + \log(1/\delta)))$. This bound also significantly improves an analysis from [19]. Due to space limitations we omit the corresponding proof in this extended abstract.

The **min-sum k-clustering** and the **balanced k-median** problems are combinatorially more complex than the **k-median** problem. For these two problems we give the *first* sublinear-time algorithms. Since in metric spaces the solution to the balanced **k-median** problem is within a factor of 2 of that of the min-sum **k-clustering** problem, we will consider the balanced **k-median** problem only.

We consider the problem of minimizing the average balanced **k-median** cost, that is, the cost of the balanced **k-median** normalized by the square of the number of input elements. We use the same approach as for the **k-median** problem. Let $\epsilon > 0$, $\alpha \geq 1$, $\beta > 0$, and $0 < \delta < 1$ be arbitrary parameters. We prove that if we pick a sample set of size $\widetilde{O}\left(\frac{\Delta}{\epsilon} \cdot \left(\frac{\sqrt{k}\,\alpha^2 \ln(1/\delta)}{\beta} + \frac{k + \ln(1/\delta)}{\epsilon}\right)\right)$ i.u.r., then an **α-approximation** of the optimal solution for the sample set approximates the average balanced **k-median** cost to within $(2\alpha + \beta)\, med^b_{avg}(V, k) + \epsilon$ with probability at least $1 - \delta$, where $med^b_{avg}(V, k)$ denotes the average cost of the optimal solution for balanced **k-median**. Notice that similarly as for the **k-median** problem, the sample size is independent of n.

Unlike in the **k-median** problem, the output of balanced **k-median** is supposed to consist of a set of k centers $c_1, \ldots, c_k$ and a partition (clustering) of the input V into $V_1 \cup \cdots \cup V_k$ that minimizes (or approximates the minimum) of $\sum_{i=1}^{k} |V_i| \sum_{v \in V_i} \mu(v, c_i)$. Our sampling algorithm leads to a randomized algorithm that in time independent of n

returns the set of k centers $c_1, \ldots, c_k$ for which the value of $\frac{\sum_{i=1}^{k} |V_i| \sum_{v \in V_i} \mu(v,c_i)}{|V|^2}$ is at most $O(med^b_{avg}(V,k)) + \epsilon$ with probability at least $1-\delta$. If one also knows the number of elements that are assigned to each cluster in an approximate solution, then one can compute in $O(n\,k) + \widetilde{O}(k^{2.5}\sqrt{n})$ time an optimal clustering [22]. Since our algorithm can be modified to provide the cluster sizes we can use this approach to compute a good solution quickly from the implicit representation as a balanced k-median.

1.3 High Level Description of Our Approach

Before we begin to analyze specific problems we first discuss our high level approach. We study the approximation guarantee of the following natural sampling scheme. Choose a multiset S of s elements i.u.r. from V, for some suitable chosen s. Then run an α-approximation algorithm $\mathbb{A}$ for the problem of interest on S. What is the quality of the solution computed by $\mathbb{A}$ on S?

Generic sampling scheme $(V, \mathbb{A}, s)$

choose a multiset $S \subseteq V$ of size s i.u.r.
run α-approximation algorithm $\mathbb{A}$ on input S to compute a solution C^* (set of k centers)
return set C^*

To analyze the approximation guarantee of this approach we proceed in two steps. First, we show that w.h.p. and after normalization $cost(S, C_{opt})$ is an approximation of $cost(V, C_{opt})$, where C_{opt} denotes an optimal solution for V. Since C_{opt} may not be a feasible solution for S (e.g., in the k-median problem C_{opt} may not be contained in S) we show that there is a *feasible* solution in S which has cost at most $\frac{c}{\alpha} \cdot cost(S, C_{opt})$ for some constant $c \geq \alpha$. Then we show that w.h.p. every possible solution for V with cost more than $c \cdot cost(V, C_{opt})$ is either not a feasible solution for S or has cost more than $c \cdot cost(S, C_{opt})$ for S. Since S contains a solution with cost at most $\frac{c}{\alpha} \cdot cost(S, C_{opt})$, $\mathbb{A}$ will compute a solution C^* with cost at most $c \cdot cost(S, C_{opt})$. Since every solution for V with cost more than $c \cdot cost(V, C_{opt})$ has cost more than $c \cdot cost(S, C_{opt})$ for S, we know that $\mathbb{A}$ computes a solution C^* with cost at most $c \cdot cost(V, C_{opt})$ for V. Hence, our sampling is a c-approximation algorithm.
We apply this approach to study sampling algorithms for three problems: the k-median problem, the balanced k-median problem, and the min-sum k-clustering problem.

2 Analysis of the k-Median Problem

We first consider the k-median problem. A *k-median of V* is a set C of k points *(centers)* in V that minimizes the value of $\sum_{v \in V} \min_{1 \leq i \leq k} \mu(v, c_i) \equiv \sum_{v \in V} \mu(v, C)$. The *$k$-median problem* is to compute a k-median for a given metric space (V, μ).

Let $med_{opt}(V,k) = \min_{C \subseteq V, |C|=k} \sum_{v \in V} \mu(v, C)$ denote the *cost of a k-median* of V. Let $med_{avg}(V,k) = \frac{1}{|V|} \cdot med_{opt}(V,k)$ denote the *average cost of a k-median* of V. In a similar manner, for a given $U \subseteq V$ and $C \subseteq V$, we define the *average cost* of solution C to be $cost_{avg}(U, C) = \frac{1}{|U|} \sum_{v \in U} \mu(v, C)$. The following theorem summarizes our analysis and it is the main result of this section.

Theorem 1. *Let (V, μ) be a metric space. Let $0 < \delta < 1$, $\alpha \geq 1$, and $\epsilon > 0$ be approximation parameters. Let $\mathbb{A}$ be an α-approximation algorithm for the k-median problem in metric spaces. If we choose a sample set $S \subseteq V$ of size s i.u.r., with*

$$s \geq c \cdot (1 + \alpha/\epsilon) \cdot \left(k + \frac{\Delta}{\epsilon} \cdot \left(\alpha \cdot \ln(1/\delta) + k \cdot \ln\left(\frac{k\,\Delta\,(1+\alpha/\epsilon)}{\epsilon} \right) \right) \right) ,$$

for some constant c and we run algorithm $\mathbb{A}$ with input S, then for the solution C^* obtained by $\mathbb{A}$, with probability at least $1 - \delta$ it holds the following

$$cost_{avg}(V, C^*) \leq (2\alpha + \epsilon) \cdot med_{avg}(V, k) + \epsilon .$$

To begin our analysis of the quality of the approximation of C^* and the proof of Theorem 1, let us introduce some basic notation. Let $\beta > 0, \alpha \geq 1$. A set of k centers C is a *β-bad α-approximation* of k-median of V if $cost_{avg}(V, C) > (\alpha+\beta) \cdot med_{avg}(V, k)$. If C is not a β-bad α-approximation then it is a *β-good α-approximation.*

For the k-median problem we want to prove for certain s that our algorithm is a $(2(\alpha+\beta))$-approximation algorithm. Following the approach described in the previous section, we have to show that our sample set S contains w.h.p. a solution with cost at most $2(1 + \beta/\alpha) \cdot med_{avg}(V, k)$, and hence, any α-approximation for S returns a $2(\alpha + \beta)$-approximation for V w.h.p. We prove the following lemma.

Lemma 1. *Let S be a multiset of size $s \geq \frac{3\Delta\alpha(1+\alpha/\beta)\ln(1/\delta)}{\beta \cdot med_{avg}(V,k)}$ chosen from V i.u.r. If an α-approximation algorithm for k-median $\mathbb{A}$ is run on input S, then for the solution C^* obtained by $\mathbb{A}$ holds $\mathbf{Pr}\left[cost_{avg}(S, C^*) \leq 2(\alpha + \beta) \cdot med_{avg}(V, k) \right] \geq 1 - \delta$.*

Proof. Let C_{opt} denote a k-median of V and let X_i denote the random variable for the distance of the ith point in S to the nearest center of C_{opt}. Then, $cost_{avg}(S, C_{opt}) = \frac{1}{s} \sum_{1 \leq i \leq s} X_i$. Furthermore, since $\mathbf{E}[X_i] = med_{avg}(V, k)$, we also have $med_{avg}(V, k) = \frac{1}{s} \cdot \mathbf{E}\left[\sum X_i \right]$. Therefore,

$$\mathbf{Pr}\left[cost_{avg}(S, C_{opt}) > (1 + \tfrac{\beta}{\alpha}) med_{avg}(V, k) \right] = \mathbf{Pr}\left[\sum_{1 \leq i \leq s} X_i > (1 + \tfrac{\beta}{\alpha}) \mathbf{E}\left[\sum_{1 \leq i \leq s} X_i \right] \right].$$

Observe that each X_i satisfies $0 \leq X_i \leq \Delta$. Therefore, we can apply a Hoeffding bound to obtain:

$$\mathbf{Pr}\left[\sum_{1 \leq i \leq s} X_i > (1 + \beta/\alpha) \cdot \mathbf{E}\left[\sum_{1 \leq i \leq s} X_i \right] \right] \leq e^{-\frac{s \cdot med_{avg}(V,k) \cdot \min\{(\beta/\alpha), (\beta/\alpha)^2\}}{3\Delta}} \leq \delta .$$

Let C be the set of k centers in S obtained by replacing each $c \in C_{opt}$ by its nearest neighbor in S. By the triangle inequality, we get $cost_{avg}(S, C) \leq 2 \cdot cost_{avg}(S, C_{opt})$. Hence, multiset S contains a set of k centers whose cost is at most $2 \cdot (1 + \beta/\alpha) \cdot med_{avg}(V, k)$ with probability at least $1 - \delta$. Therefore, the lemma follows because $\mathbb{A}$ returns an α-approximation C^* of the k-median for S. □

Next, we show that any solution $C_b \subseteq S$ that is a (6β)-bad (2α)-approximation of a k-median of V satisfies $cost_{avg}(S, C_b) > 2(\alpha+\beta) \cdot med_{avg}(V, k)$ with high probability.

Lemma 2. *Let S be a multiset of s points chosen i.u.r. from V with s such that*

$$s \geq c \cdot \left((1+\alpha/\beta)\, k + \frac{(\alpha+\beta)\cdot\Delta\cdot\left(\ln(1/\delta) + k\ln\left(\frac{k(\alpha+\beta)\Delta}{\beta^2\, med_{avg}(V,k)}\right)\right)}{\beta^2\, med_{avg}(V,k)} \right) ,$$

where c is a certain positive constant. Let $\mathbb{C}$ be the set of (6β)-bad (2α)-approximations C of a k-median of V. Then,

$$\mathbf{Pr}\left[\exists C_b \in \mathbb{C} : C_b \subseteq S \text{ and } cost_{avg}(S, C_b) \leq 2(\alpha+\beta)\, med_{avg}(V,k)\right] \leq \delta .$$

Proof. Let $s \geq \frac{2\alpha+3\beta}{\beta}\, k$. Let us consider an arbitrary solution C_b that is a (6β)-bad (2α)-approximation of a k-median of V and let S^* be a multiset of $s-k$ points chosen i.u.r from V. Then,

$$\begin{aligned}
&\mathbf{Pr}\left[C_b \subseteq S \text{ and } cost_{avg}(S, C_b) \leq 2(\alpha+\beta)\, med_{avg}(V,k)\right] \\
&= \mathbf{Pr}\left[cost_{avg}(S, C_b) \leq 2(\alpha+\beta)\, med_{avg}(V,k) \;\middle|\; C_b \subseteq S\right] \cdot \mathbf{Pr}\left[C_b \subseteq S\right] \\
&= \mathbf{Pr}\left[cost_{avg}(S^*, C_b) \leq 2\cdot\frac{s}{s-k}\left((\alpha+\beta)\, med_{avg}(V,k)\right)\right] \cdot \mathbf{Pr}\left[C_b \subseteq S\right] \quad (1) \\
&\leq \mathbf{Pr}\left[cost_{avg}(S^*, C_b) \leq 2\,(\alpha+1.5\beta)\, med_{avg}(V,k))\right] \cdot \mathbf{Pr}\left[C_b \subseteq S\right] , \quad (2)
\end{aligned}$$

where (1) holds because the elements are chosen with repetition and (2) follows from $s \geq \frac{2\alpha+3\beta}{\beta}\, k$. Furthermore, similarly as in the proof of Lemma 1, we can prove the following inequality

$$\mathbf{Pr}\left[cost_{avg}(S, C_b) \leq 2(\alpha+1.5\beta)\, med_{avg}(V,k) \;\middle|\; \right] \leq e^{\frac{-s\beta^2\, med_{avg}(V,k)}{(\alpha+3\beta)\Delta}} . \quad (3)$$

Therefore, we can plug inequality (3) and the identity $\mathbf{Pr}[C_b \subseteq S] = (s/n)^k$ into (2), and combine this with the upper bound $|\mathbb{C}| \leq n^k$, to conclude the proof. □

Proof of Theorem 1. Let s be chosen such that the prerequisites of Lemmas 1 and 2 hold, that is,

$$s \geq c(1+\alpha/\beta)\left(k + \frac{\Delta}{\beta\, med_{avg}(V,k)}\left(\alpha\ln(1/\delta) + k\ln\left(\frac{k(\alpha+\beta)\Delta}{\beta^2\, med_{avg}(V,k)}\right)\right)\right) \quad (4)$$

for certain constant c. Let S be a multiset of s points chosen i.u.r. from V. Then, by Lemma 2 with probability at least $1-\delta$, no set $C \subseteq S$ that is a (6β)-bad (2α)-approximation of a k-median of V satisfies the inequality

$$cost_{avg}(S, C) \leq 2(\alpha+\beta)\, med_{avg}(V,k) .$$

On the other hand, if we run algorithm $\mathbb{A}$ for set S, then the resulting set C^* of k centers with probability at least $1-\delta$ satisfies

$$cost_{avg}(S, C^*) \leq 2(\alpha+\beta)\, med_{avg}(V,k) .$$

This, together with the claim above implies that with probability at least $1 - 2\delta$ the set C^* is a (6β)-**good** (2α)-**approximation** of a k-**median** of V. Hence,

$$cost_{avg}(V, C^*) \leq (2\alpha + 6\beta) \cdot med_{avg}(V, k) \ .$$

This implies immediately the following bound:

$$\mathbf{Pr}\Big[cost_{avg}(V, C^*) \leq (2\alpha + 6\beta) \cdot med_{avg}(V, k)\Big] \ \geq \ 1 - 2\delta \ .$$

To complete the proof we only must remove the dependence of $med_{avg}(V, k)$ in the bound of s in (4) and relate β to ϵ. For $med_{avg}(V, k) \geq 1$, Theorem 1 follows directly from our discussion above by replacing 6β by ϵ. For $med_{avg}(V, k) < 1$, Theorem 1 follows by replacing β by $\epsilon / med_{avg}(V, k)$. For more details we refer to the full version of the paper. □

3 Min-sum k-Clustering and Balanced k-Median in Metric Spaces

As we mentioned in Introduction, we follow the approach from [3] and [9] and consider the balanced k-**median** problem instead of analyzing min-sum k-**clustering.**

Let (V, μ) be a metric space. A *balanced* k-*median of* V is a set $C = \{c_1, \ldots, c_k\}$ of k points (centers) in V that minimizes the value of

$$\min_{\text{partition of } V \text{ into } V_1 \cup \cdots \cup V_k} \ \sum_{i=1}^{k} |V_i| \cdot \sum_{u \in V_i} \mu(u, c_i) \ .$$

The *balanced* k-*median problem* is for a given (V, μ) to compute a balanced k-**median** of V and a partition of V into $V_1 \cup \cdots \cup V_k$ that minimizes the sum above.

Let

$$med^b_{opt}(V, k) \ = \ \min_{C = \{c_1, \ldots, c_k\} \subseteq V} \ \min_{\text{partition of } V \text{ into } V_1 \cup \cdots \cup V_k} \ \sum_{i=1}^{k} |V_i| \cdot \sum_{u \in V_i} \mu(u, c_i)$$

denote the *cost of a balanced* k-*median of* V, and let $med^b_{avg}(V, k) = \frac{1}{|V|^2} med^b_{opt}(V, k)$ denote the *average cost of a balanced* k-*median of* V. For a given set $U \subseteq V$ and a set of k centers $C = \{c_1, \ldots, c_k\} \subseteq V$, let us define

$$cost^b(U, C) = \min_{\substack{\text{partition of } U \\ \text{into } U_1 \cup \cdots \cup U_k}} \sum_{i=1}^{k} |U_i| \sum_{u \in U_i} \mu(u, c_i) \ \text{ and } \ cost^b_{avg}(U, C) = \frac{cost^b(U, C)}{|U|^2} \ .$$

A set of k centers C is called a (ϵ, β)-*bad* α-*approximation* of balanced k-**median** of V if $cost^b_{avg}(V, C) > (\alpha + \beta) \cdot med^b_{avg}(V, k) + \epsilon$. If C is not a (ϵ, β)-bad α-**approximation** then it is a (ϵ, β)-*good* α-*approximation.*

3.1 Sampling Algorithms for the Balanced k-Median Problem in Metric Spaces

Our high level approach of analyzing the balanced k-median problem is essentially the same as for the k-median problem. We investigate the generic sampling scheme described in Section 1.3, and in Section 3.2 we prove the following main theorem.

Theorem 2. *Let (V, μ) be a metric space. Let $\mathbb{A}$ be an α-approximation algorithm for balanced k-median in metric spaces and let $0 \leq \epsilon \leq 1/4$, $\beta \geq \frac{4\alpha\epsilon}{1-2\epsilon}$, $0 < \delta < 1$ be approximation parameters. If we choose a sample set $S \subseteq V$ of size s i.u.r., where*

$$s \geq \frac{c \cdot \Delta}{\epsilon} \cdot \left(\frac{\sqrt{k}\,\ln(k/\delta)\,\alpha^2}{\beta} + \frac{\ln(k/\delta) + k \cdot \ln(k\,\Delta/\epsilon)}{\epsilon} \right) ,$$

and we run algorithm $\mathbb{A}$ with input S, then for the solution C^* obtained by $\mathbb{A}$, with probability at least $1 - \delta$ it holds the following

$$cost^b_{avg}(V, C^*) \leq (2\alpha + \beta) \cdot med^b_{avg}(V, k) + \epsilon .$$

Furthermore, in time $O(n\,k) + \widetilde{O}(k^{2.5}\,n^{0.5})$ one can find a clustering of V that satisfies the above approximation guarantee.

Moreover, the solution C^* approximates an optimal solution for the min-sum k-clustering problem within a factor two times larger than claimed above.

The last claim in Theorem 2 follows from the fact that in metric spaces the solution to balanced k-median is within a factor of 2 of that of min-sum k-clustering.

3.2 Analysis of Generic Sampling Scheme for Balanced k-Median

Our analysis follows the path used in Section 2. The main difference is that we must explicitly use "outliers" in our analysis, what makes it significantly more complicated.

We begin with a result corresponding to Lemma 1 for k-median.

Lemma 3. *Let C_{opt} be a balanced k-median of V. Let $0 < \gamma, \delta < 1$, $\epsilon > 0$ be arbitrary parameters. If we choose a multiset $S \subseteq V$ of size $s \geq \frac{6\alpha \cdot \Delta \cdot \ln(3k/\delta)}{\gamma \cdot \epsilon}$ i.u.r., then*

$$\mathbf{Pr}\left[cost^b_{avg}(S, C_{opt}) \leq (1+\gamma)^3 med^b_{avg}(V, k) + \frac{6k\Delta\ln(3k/\delta)}{\gamma^2 s^2} + \epsilon/\alpha \right] \geq 1 - \delta .$$

Proof. To simplify the notation, let $\delta_1 = \frac{1}{3}\,\delta/k$. Let $C_{opt} = \{c_1, \ldots, c_k\}$. Let $V_1^* \cup \cdots \cup V_k^*$ be the optimal partition of V, i.e., $med^b_{opt}(V, k) = \sum_{i=1}^{k} |V_i^*| \cdot \sum_{u \in V_i^*} \mu(u, c_i)$.

Let us call set V_i^* *dense* if $|V_i^*| \geq \frac{3 \cdot \ln(1/\delta_1)}{\gamma^2} \cdot \frac{|V|}{s}$; V_i^* is *sparse* otherwise. Let S_i be the random variable that denotes the multiset $S \cap V_i^*$ (we assume S_i is a multiset, that is, an element can appear multiple times in S_i if it belongs to V_i^* and it appears multiple times in S). Our first observation (that can be easily proven using a Chernoff bound) is that if V_i^* is dense, then we have $\mathbf{Pr}\left[|S_i| \leq (1-\gamma) \cdot \frac{s \cdot |V_i^*|}{|V|} \right] \leq \delta_1$ and $\mathbf{Pr}\left[|S_i| \geq (1+\gamma) \cdot \frac{s \cdot |V_i^*|}{|V|} \right] \leq \delta_1$, and if V_i^* is sparse, then we have $\mathbf{Pr}\left[|S_i| \geq \frac{6 \cdot \ln(1/\delta_1)}{\gamma^2} \right] \leq \delta_1$.

Therefore, from now on, let us condition on the event that for dense sets V_i^* we have $(1-\gamma)\cdot\frac{s\cdot|V_i^*|}{|V|} < |S_i| < (1+\gamma)\cdot\frac{s\cdot|V_i^*|}{|V|}$ and for sparse sets V_i^* we have $|S_i| < \frac{6\cdot\ln(1/\delta_1)}{\gamma^2}$. This event holds with probability at least $1 - 2\cdot k\cdot\delta_1$.

For any set V_i^*, let X_i^j be the random variable that denotes the distance between the jth randomly selected element from S_i and the center c_i. Observe that for any set V_i^*, we have $\mathbf{E}[X_i^j] = \frac{1}{|V_i^*|}\cdot\sum_{u\in V_i^*}\mu(u,c_i)$. Let us fix i and let us first assume that

$$2\cdot\frac{|S_i|}{s^2}\cdot\gamma\cdot\frac{|S_i|}{|V_i^*|}\cdot\sum_{u\in V_i^*}\mu(u,c_i) \;\geq\; \epsilon/\alpha \;. \tag{5}$$

Since $0 \leq X_i^j \leq \Delta$, we use Hoeffding bound to prove

$$\mathbf{Pr}\Big[\sum_{j=1}^{|S_i|} X_i^j \;\geq\; (1+\gamma)\cdot|S_i|\cdot\frac{\sum_{u\in V_i^*}\mu(u,c_i)}{|V_i^*|}\Big] \leq \exp\Big(-\frac{\gamma}{3\cdot\Delta}\cdot s\cdot\epsilon/(2\alpha)\Big) \tag{6}$$

where the last inequality follows from (5). If (5) does not hold, then let γ^*, $\gamma^* > \gamma$, be such that

$$2\cdot\frac{|S_i|}{s^2}\cdot\gamma^*\cdot\frac{|S_i|}{|V_i^*|}\cdot\sum_{u\in V_i^*}\mu(u,c_i) \;=\; \epsilon/\alpha \;.$$

Notice that in that case,

$$\gamma^*\cdot\mathbf{E}\Big[\sum_{j=1}^{|S_i|} X_i^j\Big] \;=\; \gamma^*\cdot|S_i|\cdot\frac{\sum_{u\in V_i^*}\mu(u,c_i)}{|V_i^*|} \;=\; \frac{s^2\cdot\epsilon}{2\cdot\alpha\cdot|S_i|} \;\geq\; \frac{s\cdot\epsilon}{2\cdot\alpha} \;. \tag{7}$$

Observe that since (5) does not hold and since $\gamma \leq 1$, we have $\gamma \leq \min\{1,\gamma^*\}$. Therefore, we can use the Hoeffding bound to prove that

$$\begin{aligned}\mathbf{Pr}\Big[\sum_{j=1}^{|S_i|} X_i^j \geq (1+\gamma^*)\cdot\mathbf{E}\Big[\sum_{j=1}^{|S_i|} X_i^j\Big]\Big] &\leq \exp\Big(-\frac{\min\{\gamma^*,\gamma^{*2}\}\cdot|S_i|}{3\cdot\Delta}\cdot\frac{\sum_{u\in V_i^*}\mu(u,c_i)}{|V_i^*|}\Big)\\ &\leq \exp\Big(-\frac{\gamma\cdot s\cdot\epsilon}{6\cdot\Delta\cdot\alpha}\Big) \;.\end{aligned} \tag{8}$$

Notice that the inequalities (6) – (8) imply that if $s \geq \frac{6\alpha\cdot\Delta\cdot\ln(1/\delta_1)}{\gamma\cdot\epsilon}$, then

$$\mathbf{Pr}\Big[\sum_{j=1}^{|S_i|} X_i^j \;\geq\; (1+\gamma)\cdot\frac{|S_i|\cdot\sum_{u\in V_i^*}\mu(u,c_i)}{|V_i^*|} + \frac{s\cdot\epsilon}{2\cdot\alpha}\Big] \leq \delta_1 \;.$$

Therefore, from now on, let us condition on the event that for every i, we have

$$\sum_{u\in S_i}\mu(u,c_i) \;<\; (1+\gamma)\cdot\frac{|S_i|\cdot\sum_{u\in V_i^*}\mu(u,c_i)}{|V_i^*|} + \frac{s\cdot\epsilon}{2\cdot\alpha} \;,$$

what holds with probability at least $1 - k\,\delta_1$. Under the conditioning above, we can proceed to the final conclusion:

$$\begin{aligned} cost^b(S,C) &\le \sum_{i=1}^{k} |S_i| \cdot \sum_{u \in S_i} \mu(u,c_i) \le \sum_{i:V_i^* \text{ is sparse}} |S_i| \cdot \sum_{u \in S_i} \mu(u,c_i) + \sum_{i:V_i^* \text{ is dense}} |S_i| \cdot \sum_{u \in S_i} \mu(u,c_i) \\ &\le \frac{6k\Delta \ln(1/\delta_1)}{\gamma^2} + \sum_{i:V_i^* \text{ is dense}} \frac{(1+\gamma)s|V_i^*|}{|V|} \left(\frac{(1+\gamma)|S_i| \sum_{u \in V_i^*} \mu(u,c_i)}{|V_i^*|} + \frac{s\epsilon}{2\alpha} \right) \\ &\le \frac{6\,k\,\Delta\,\ln(1/\delta_1)}{\gamma^2} + \frac{\epsilon\, s^2}{\alpha} + \left(\frac{(1+\gamma)\,s}{|V|} \right)^2 (1+\gamma)\, med^b_{opt}(V,k) \ . \end{aligned}$$

This yields the following bound that holds with probability at least $1 - 3\,k\,\delta_1 = 1 - \delta$:

$$cost^b_{avg}(S,C) \ \le \ \frac{6 \cdot k \cdot \Delta \cdot \ln(3k/\delta)}{\gamma^2 \cdot s^2} \ + \ \frac{\epsilon}{\alpha} \ + \ (1+\gamma)^3 \cdot med^b_{avg}(V,k) \ ,$$

what concludes the proof of Lemma 3. □

Lemma 3 (with $\gamma \approx \alpha/\beta$) can be combined with arguments used in Lemma 1 to prove the following.

Corollary 1. *Let $0 < \beta < \alpha$ and $\epsilon > 0$. Let S be a multiset of size $s \ge \frac{c\sqrt{k}\Delta \ln(3k/\delta)\,\alpha^2}{\beta\,\epsilon}$ chosen from V i.u.r., where c is some constant. If an α-**approximation** algorithm for balanced k-**median** $\mathbb{A}$ is run with input S, then for the solution C^* obtained by $\mathbb{A}$ holds*

$$\mathbf{Pr}\left[cost^b_{avg}(S,C^*) \ \le \ 2\,(\alpha+\beta) \cdot med^b_{avg}(V,k) + \epsilon \right] \ \ge \ 1 - \delta \ . \qquad \square$$

The next step in our analysis is to consider bad approximations. Our analysis follows the approach used before in the proof of Lemma 2; the main difference is a larger number of parameters used in the analysis. Corollary 1 proves that typically there is a set of k centers in the sample S that has the average cost close to $med^b_{avg}(V,k)$. Now, we show that any $C_b \subseteq S$ that is a $(5\,\epsilon, 2\,\beta)$-**bad** $(2\,\alpha)$-**approximation** of a balanced k-**median** of V satisfies $cost_{avg}(S,C_b) > 2\,(\alpha+\beta) \cdot med^b_{avg}(V,k) + \epsilon$ with high probability. Details of the proof of the following lemma are deferred to the full version of the paper.

Lemma 4. *Let S be a multiset of s points chosen i.u.r. from V with s such that:*

$$s \ \ge \ c \cdot \left(\frac{\Delta}{\epsilon^2} \cdot (\ln(k/\delta) + k \cdot \ln(k\,\Delta/\epsilon)) + \frac{1}{\beta} \right) \ ,$$

*where c is a suitable positive constant. Let $\mathbb{C}$ be the set of $(5\epsilon, 2\,\beta)$-**bad** $(2\,\alpha)$-**approximations** C of a balanced k-**median** of V. Then,*

$$\mathbf{Pr}\left[\exists C_b \in \mathbb{C} : C_b \subseteq S \text{ and } cost_{avg}(S,C_b) \le (1-\epsilon)^2\,(2\,\alpha+\beta)\, med^b_{avg}(V,k) + \epsilon \right] \le \delta.$$

Now Theorem 2 follows from Corollary 1 and Lemma 4. To expand our implicit representation of the clustering, we can use the values v_i^* obtained from the optimum partition of our sample set S as cluster sizes and then use the algorithm from [22]. □

References

1. S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean **k-medians** and related problems. *30th STOC,* pp. 106–113, 1998.
2. V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for **k-median** and facility location problems. *33rd STOC,* pp. 21–30, 2001.
3. Y. Bartal, M. Charikar, and D. Raz. Approximating min-sum **k-clustering** in metric spaces. *33rd STOC,* pp. 11–20, 2001.
4. M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. *40th FOCS,* pp. 378–388, 1999.
5. M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the **k-median** problem. *31st STOC,* pp. 1–10, 1999.
   M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. *12th SODA,* pp. 642–651, 2001.
6. M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. *35th STOC,* pp. 30–39, 2003.
7. B. Chazelle. Who says you have to look at the input? The brave new world of sublinear computing? *15th SODA,* p. 134, 2004.
8. W. Fernandez de la Vega, M. Karpinski, C. Kenyon, and Y. Rabani. Polynomial time approximation schemes for metric min-sum clustering. *35th STOC,* pp. 50–58, 2003.
9. N. Gutmann-Beck and R. Hassin. Approximation algorithms for min-sum ***p*-clustering**. *Discrete Applied Mathematics,* 89: 125–142, 1998.
10. S. Har-Peled and S. Mazumdar. Coresets for **k-means** and **k-median** clustering and their applications. *36th STOC,* 2004.
11. P. Indyk. Sublinear time algorithms for metric space problems. *31st STOC,* pp. 428–434, 1999.
12. P. Indyk. A sublinear time approximation scheme for clustering in metric spaces. *40th FOCS,* pp. 154–159, 1999.
13. K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. *34th STOC,* pp. 731–740, 2002.
14. K. Jain and V. V. Vazirani. Primal-dual approximation algorithms for metric facility location and **k-median** problems. *40th FOCS,* pp. 2–13, 1999.
15. S. G. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the Euclidean **k-median** problems. *7th ESA,* pp. 378–389, 1999.
16. R. Kumar and R. Rubinfeld. Sublinear time algorithms. *SIGACT News,* 34(4):57–67, 2003.
17. R. R. Mettu and C. G. Plaxton. Optimal time bounds for approximate clustering. *18th Conference on Uncertainty in Artificial Intelligence (UAI),* pp. 344–351, August 2002.
18. A. Meyerson, L. O'Callaghan, and S. Plotkin. A **k-median** algorithm with running time independent of data size. *Journal ofMachine Learning,* 2004.
19. N. Mishra, D. Oblinger, and L. Pitt. Sublinear time approximate clustering. *12th SODA,* pp. 439–447, 2001.
20. L. J. Schulman. Clustering for edge-cost minimization. *32nd STOC,* pp. 547–555, 2000.
21. S. Sahni and T. Gonzalez. **P-complete** approximation problems. *JACM,* 23: 555-566, 1976.
22. T. Tokuyama, and J. Nakano. Geometric algorithms for the minimum cost assignment problem. *Random Structures and Algorithms,* 6(4): 393-406, 1995.

Solving Two-Variable Word Equations*
(Extended Abstract)

Robert Dąbrowski and Wojtek Plandowski

Institute of Informatics
University of Warsaw
Banacha 2, 02-097 Warszawa, Poland
{r.dabrowski,w.plandowski}@mimuw.edu.pl

Abstract. We present an algorithm that solves word equations in two variables. It computes a polynomial size description of the equation's solutions in time $O(n^5)$. This additionally improves the result by Ilie and Plandowski [8] by giving the currently fastest algorithm to decide solvability of two-variable word equations.

1 Introduction

One of the most famous and most complicated algorithms existing in literature is *Makanin's algorithm* [13]. The algorithm takes as an input a word equation and decides whether or not the equation has a solution. It has been improved several times. The algorithm's currently best version works in EXPSPACE [6] and occupies (including the proof of correctness) over forty pages [5].

Recently new algorithms to decide solvability of general word equations have been found [15,17]. The first one works nondeterministically in polynomial time with respect to the length of the input equation and the logarithm of the length of its minimal solution. Since the best upper bound for the length of the minimal solution is double exponential [16], then with this bound the algorithm in [15] works in NEXPTIME. The algorithm in [17] works in PSPACE.

Obviously the algorithms solving the problem of satisfiability of general word equations cannot be called efficient. We cannot even expect efficiency since the problem is NP-hard [1,11]. However, if we concentrate on selected classes of word equations, then there do exist polynomial time algorithms either to decide solvability, or to describe solutions of word equations.

For instance, an efficient algorithm that solves word equations in one variable is known [3]. It works in $O(\#_x \log n)$ time, where n is the length of the input equation and $\#_x$ is the number of variable occurrences. For two-variable word equations, there exist two polynomial time algorithms [2,8] that determine solvability. The best one works in $O(n^6)$ time. There is also an efficient $O(n \log^2 n)$ time algorithm for restricted equations with two variables [14].

There are two algorithms that solve general word equations [9,18]. The first one generates representation of solutions which is a set of unifiers. If this set

* Supported by KBN grant 4T11C04425.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 408–419, 2004.

is finite the algorithm terminates. The second algorithm generates a finite representation of all solutions in form of a finite graph. It works on equations in free groups and is based on Makanin's algorithm for free groups which is not primitive recursive [12]. Both algorithms cannot be called efficient. Existence of a polynomial time algorithm to solve a word equation in two variables has remained up to now an open problem. In this papers we present the first polynomial time algorithm that solves equations in two variables. By solving a word equation we mean an algorithm that finds a polynomial description of all of its solutions.

2 Notation

A *factorization of a word* w is a sequence of words $w_1, \ldots, w_l$ such that $w = w_1 \ldots w_l$. A *factorization* is a function $\mathcal{F}$ such that it takes a word and returns some factorization of this word.

Definition 1. *For a primitive word* $P \in \Sigma^*$ *we define P-factorization as follows. For any word* $w \in \Sigma^*$ *there exists a unique representation* $w = w_0 \cdot P^{k_1} \cdot w_1 \cdot \ldots \cdot P^{k_n} \cdot w_n$ *where* $n \geq 0$, $k_i \geq 0$ *for any* $1 \leq i \leq n$ *and (1)* w_i *does not contain* P^2 *as a factor for any* $0 \leq i \leq n$*; (2) P is both a proper prefix and suffix of* w_i *for any* $0 < i < n$*; (3) P is a proper suffix of* w_0 *or* $w_0 = 1$*; (4) P is a proper prefix of* w_n *or* $w_n = 1$*. Then the* P-factorization *of* w *is the ordered sequence* $w_0, P^{k_1}, w_1, \ldots, P^{k_n}, w_n$*. The* size *of the P-factorization is*

$$\sum_{i=0}^{n} |w_i| + |P| + \sum_{i=1}^{n} \log k_i.$$

Our next definition comes from [10] and is quite technical. The idea of it is to consider factorizations which have quite strong property. Let a word y occurs inside a word x. If we place y over its occurrence inside x and compare both factorization of y and the part of the factorization of x which is under y, then the factorizations are almost the same except the beginning and end. A formal definition follows.

Definition 2. *Let* $\mathcal{F}$ *be a factorization. Let* $\mathcal{F}(x) = x_1, \ldots, x_j$ *and* $\mathcal{F}(y) = y_1, \ldots, y_k$ *for some words* x *and* y*. The factorization* $\mathcal{F}$ *is* synchronizing *iff for some non-negative integer parameters* l *and* r *if* $k > l + r$ *then there exist* $l' \leq l$ *and* $r' \leq r$ *such that the following condition holds. Denote* $u = y_1 \ldots y_{l'}$ *and* $v = y_{k-r'+1} \ldots y_k$ (border *factors:* l' *starting and* r' *ending ones). If* y *occurs in* x *starting at position* i *then (1) positions* $i + |u|$ *and* $i + |y| - |v|$ *in* x *are starting positions of factors, say* x_p *and* x_q*, respectively; (2) the sequences of factors* $x_p, \ldots, x_q$ *and* $y_{l'+1}, \ldots, y_{k-r'}$ *are identical; (3) the occurrence of* u *at position* i *in* x *covers at most* $l-1$ *factors of* x*; (4) the occurrence of* v *at position* $i + |y| - |v|$ *in* x *covers at most* $r - 1$ *factors of* x*.*

Proposition 1 (Karhumäki, Mignosi, Plandowski [10]). *Given a primitive word P, the P-factorization is synchronizing with* $l = r = 2$.

Definition 3. *By a* k-ary word generator *we denote a function* $w : N^k \to \Sigma^*$ *that allows for a compact representation of a family of words* $\{w(n_1, \ldots) | n_1, \ldots \in N\}$. *In this paper* unary *or* binary generators *are used and they are typically represented by expressions, i.e.* unary generator $u^i v$ *representing set of words* $\{u^i v | i \geq 0\}$ *for certain* $u, v \in \Sigma^*$; *or* binary generator $(u^i v)^j u^i v$ *representing set of words* $\{(u^i v)^j u^i v | i, j \geq 0\}$ *We shall distinct a* constant *as a word generator with zero arity.*

Definition 4. *By a* rotation *we mean a mapping rot :* $\Sigma^* \to \Sigma^*$ *defined for any* $a \in \Sigma$, $w \in \Sigma^*$ *by* $rot(aw) = wa$. *A composition of* t *rotations is denoted by* rot^t, $t \geq 0$.

If $w = rot^t(u)$, for some t then we say that w and u *conjugate* or *are conjugates.*

Definition 5. *By a* primitive root *we mean a mapping root:* $\Sigma^* \to \Sigma^*$ *defined for any* $u \in \Sigma^*$ *by* $root(u) = v$ *iff* $v \in \Sigma^*$ *is of minimal length and such, that* $u = v^k$ *for some* $k \geq 1$.

Definition 6. *Given an equation (or a system of equations) E, by Sol(E) we denote the set of its solutions. In case of a* multiple-variable *word equations, by* $Sol_x(E)$, *for a variable* x *of E, we denote a language which is the set of the* x *components of the solutions of E.*

3 Systems of Equations

We introduce tools that let us solve some specific systems of word equations.

3.1 System S_1

Let S_1 be the following system of word equations in two variables $|u| < |x|$, where $A, B, C, D \in \Sigma^*$, CD is primitive.

$$S_1 : \begin{cases} u \cdot A \cdot x = x \cdot B \cdot u \\ CD \cdot u = u \cdot DC \end{cases}$$

We can prove the following lemma.

Lemma 1. *Given a system of equations* S_1 *of length* n, *it is posible to find in time* $O(n)$ *the following representation of* $Sol_x(S_1)$.

(α) *At most one binary generator of the form* $x_{j,k} = (P^j Q)^k P^{j+c} P'$ *for certain* $P, Q \in \Sigma^*$ *of length* $O(n)$, $|c| \leq n$, P *primitive,* P' *a prefix of P, P not a prefix of Q and any* $j, k \geq 0$, *or of the form* $x_{j,k} = (P^j Q)^k P'$ *for P primitive and P not a prefix of Q and any* $j, k \geq 0$.

(β) *A set of* $O(n)$ *unary generators* $x_j = P^j Q$ *for certain* $P, Q \in \Sigma^*$ *of length* $O(n)$, P *primitive, P not a prefix of Q,* $j \geq 0$.

3.2 System S_2

Let S_2 be the following system of distinct word equations in two variables $|u| < |x|$, where $A, B, C, D \in \Sigma^*$. We assume $|A| = |B| \leq |C| = |D|$ and $A \neq C$ or $B \neq D$.

$$S_2 : \begin{cases} u \cdot A \cdot x = x \cdot B \cdot u \\ u \cdot C \cdot x = x \cdot D \cdot u \end{cases}$$

We can prove the following lemma.

Lemma 2. *Given a system of equations S_2 of length n, it is possible to find in time $O(n^2)$ the following representation for $Sol_x(S_1)$*

(α) At most one binary generator $x_{j,k} = (P^jQ)^kP^{j+b}P'$ for some P, Q of length $O(n)$ and for some constant $|b| \leq n$.
(β) A set of $O(n)$ unary generators $x_j = P^jQ$ for some P, Q of length $O(n)$, P primitive and $j \geq 0$.
(γ) A set of $O(n)$ constants.

3.3 System S_3

We distinguish a system S_3 which consists of one equation, in two variables u, x, where $A, B \in \Sigma^*$.

$$S_3 : u \cdot A \cdot x = x \cdot B \cdot u$$

There is a a close connection between such equations and Sturmian words [8]. Moreover, as already noticed by Hmelevskii [7], S_3 has a non-empty set of solutions iff there exist $P, Q, R \in \Sigma^*$ such, that $A = PQR$ and $B = QRP$. The following lemma holds.

Proposition 2 (Ilie, Plandowski [8]). *Given an equation S_3 of length n, it is possible to find in time $O(n)$ a finite set of substitutions (computed on the basis of the graph induced by the equation) that represents $Sol(S_3)$.*

Therefore, if the problem reduces to a single equation of type S_3, then we can terminate the algorithm.

4 Interlaced Sequences

Denote $[C_i]_{i=1}^k = C_1 \ldots C_k$. Let $A(x) = [xA_i]_{i=1}^k$ and $B(x) = [B_jx]_{j=1}^k$ be two sequences of equal length over a variable x and coefficients $A_i, B_j \in \Sigma^*$. The equation

$$E : A(x) \cdot y = y \cdot B(x)$$

is called a *singleton equation* in variables x, y. The size of it denoted by n; we additionally assume that x is both a prefix and a suffix of y. We introduce the

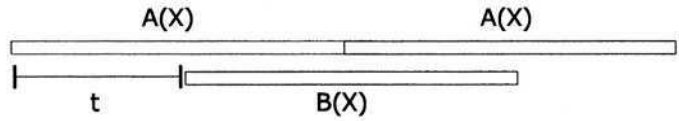


Fig. 1. Conjugation

technique of *interlaced sequences* that allows us to solve E. Fix $(x, y) \in Sol(E)$. Then $A(x)$ and $B(x)$ are conjugated (by y), that is

$$A(x) = rot^t(B(x))$$

for some $t \geq 0$. In other words $B(x)$ is a subword of $A(x)A(x)$, see Figure 1.

To find $Sol_x(E)$ we consider separately *simple* and *complex* solutions. Simple solutions correspond to the case when one of the ends of x in $B(x)$ or in $A(x)$ drops inside a constant A_i or B_j.

4.1 Simple Solutions

In the first case we consider *simple* x only. First, we take all $O(n^2)$ factors of A_i, B_j as possible *constants* x of length $O(n)$. Second, we consider all $O(n)$ prefixes and suffixes of A_i, B_j as periods. Each of them creates $O(n)$ *unary generators* $x = P^jQ$ where P is primitive, $j \geq 0$, and Q is a proper prefix of P. Totally we get $O(n^2)$ unary generators.

4.2 Complex Solution

In the remaining case we may assume that no occurrence of x either starts or ends within any of the coefficients. Therefore to solve the conjugation of $A(x)$ and $B(x)$ it suffices to consider k possible *interlaced sequences* of coefficients A_i and B_j. Fix the interlace to be $I : A_1, B_1, A_2, B_2, \ldots, A_k, B_k$; renumerate the coefficients if necessary.

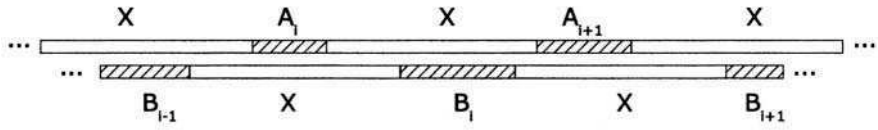


Fig. 2. Interlaced sequences

Case 1 If all coefficients A_i are equal and all coefficients B_i are equal and A_i and B_i are of equal length, then E degenerates to an equation of type S_3.

Case 2 If all coefficients are of equal length, then interlace results in a system S_2 of equations of length $O(n)$. Since Case 1 does not hold, we can find among the equations two which form a system S_2 where $|A| = |B| = |C| = |D|$ and $C \neq A$ or $B \neq D$. Then only cases (β) or (γ) may hold in Lemma 2. Hence, this case results in $O(n)$ constants and at most one unary generator.

Case 3 In the remaining case every interlace contains two consecutive coefficients $|A| \neq |B|$. It results in a system of equations as depicted in Figure 3. Let A be shorter and to the left to B (the other cases are symmetric).

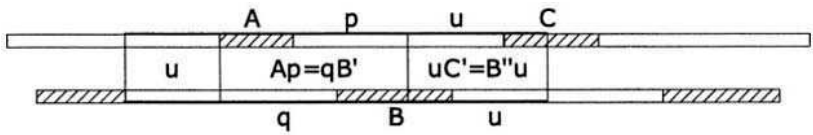


Fig. 3. System of equations

Since $pu = uq = x$ then the system is equivalent to the following one.

$$\begin{cases} u \cdot A \cdot x = x \cdot B' \cdot u \\ B'' \cdot u = u \cdot C' \end{cases}$$

It is possible to find the primitive roots of all coefficients in total time $O(n)$, hence all k systems can be reduced to systems of type S_1 in total time $O(n)$.

One remark should be done in the reasoning in this case when $|C| < |B'|$. In that case it is not possible to calculate directly the coefficient C' in the second equation. Note, however, that then, since $|u| \geq |B''| = C'$, C' is of the form CC'' where C'' is a prefix of u which is just to the right of C. We know that B'' is a prefix of u and we know the length of C'' which is $|B''| - |C|$ so we can compute C'' and therefore also C'. Hence, this case results in at most one binary generator and $O(n)$ unary generators.

Lemma 3. *Given a* singleton equation *E of length n, it either degenerates to a single equation of type S_3, or it is possible to find in time $O(n^2)$ the following representation of $Sol_x(E)$.*

- (α) *A set of $O(n)$ binary generators $x_{j,k} = (P^jQ)^kP^{j+b}P'$ for certain P, Q, R of lengths $O(n)$, $|b| \leq n$, P primitive, $j, k \geq 0$ or $x_{j,k} = (P^jQ)^kP'$ for P, Q, P' of lengths $O(n)$, P primitive $j, k \geq 0$.*
- (β) *A set of $O(n^2)$ unary generators $x_j = P^jQ$ for certain P, Q of lengths $O(n)$, P primitive, $j \geq 0$.*
- (γ) *A set of $O(n^2)$ constants x of lengths $O(n)$.*

5 Singleton Equations

Again we consider two sequences $A(x)$ and $B(x)$ as denned in the previous section and the singleton equation they induce, but this time we relax the condition $|A(x)| = |B(x)|$ and assume only that the number of x in $A(x)$ and $B(x)$ are the same. This leads to two *skew* types of singleton equations.

5.1 Singleton+ Equations

We assume $\sum_{i=1}^{k} |A_i| > \sum_{i=1}^{k} |B_i|$. By *singleton+* equations we denote equations of the form

$$E : A(x) \cdot y = y \cdot B(x) \cdot B'x'$$

where $B' \in \Sigma^*$, where x' is both a nontrivial prefix and suffix of x and $|B'x'| = |A(x)| - |B(x)|$ and x is a prefix and suffix of y. We can prove the following lemma.

Lemma 4. *Given a* singleton+ *equation E of length* n, *it is possible to find in time* $O(n^3)$ *the following representation of* $Sol_x(E)$.

(α) *A set of* $O(n)$ *binary generators* $x_{j,k} = (P^jQ)^k P^{j+b} P'$ *for certain P, Q, R of lengths* $O(n)$, $|b| \leq n$, *P primitive,* $j, k \geq 0$, *or* $x_{j,k} = (P^jQ)^k P'$ *for certain P, Q, P′ of lengths* $O(n)$ *and* $j, k \geq 0$.
(β) *A set of* $O(n^3)$ *unary generators* $x_j = P^jQ$, *for certain* $P, Q \in \Sigma^n$, *P primitive,* $j \geq 0$.
(γ) *A set of* $O(n^2)$ *constants* x *of lengths* $O(n)$.

5.2 Singleton- Equations

We assume $\sum_{i=1}^{k} |A_i| < \sum_{i=1}^{k} |B_i|$. By *singleton-* equations we denote equations of the form

$$E : A(x) \cdot y \cdot A'x'' = y \cdot B(x)$$

where $A' \in \Sigma^*$, x'' is both a nontrivial suffix and prefix of x and $|A'x''| = |B(x)| - |A(x)|$. We can prove the following lemma.

Lemma 5. *Given a* singleton- *equation E of length* n, *it is possible to find the following representation of* $Sol_x(E)$.

(α) *A set of* $O(n)$ *binary generators* $x_{j,k} = (P^jQ)^k P^{j+b} P'$ *for certain P, Q, R of lengths* $O(n)$, $|b| \leq n$, *P primitive,* $j, k \geq 0$.
(β) *A set of* $O(n^3)$ *unary generators* $x_j = P^jQ$, *for certain* $P, Q \in \Sigma^n$, *P primitive,* $j \geq 0$.
(γ) *A set of* $O(n^2)$ *constants* x *of lengths* $O(n)$.

6 Single-Periodic Solutions

Our goal is to solve equation E in two variables x, y for which x is known to be of the form $x = P^iQ$, for some P, Q of lengths $O(n)$, P primitive, P not a prefix of Q and any $i \geq 0$. First, we use the algorithm in [3] to solve E when the value of x is a fixed word, namely in cases $i = 0$ and $i = 1$, i.e. $x = Q$ and $x = PQ$. Then the algorithm works in $O(n^2 + \#_y \log n)$ time where $\#_y$ is the number of occurrences of the variable y in the equation. Since $\#_y = O(n)$ it totally works in $O(n^2)$ time. In the remaining part we may concentrate on the case $i \geq 2$.

Our considerations work on P-factorizations. We use two data structures which can be computed in linear time on the basis of P-factorizations of words. The first data structure is an *overlap array* for a word y. This is an array which says for each position i whether $y[i..|y|] = y[1..|y| - i + 1]$. The second data structure is a *prefix array* which for each position i of y says the length of the longest prefix of y which starts at position i in y. Both data structures are standard ones in text algorithms [4]. However, they are computed for explicite representation of y. We compute them for words which are given by a P-factorization of y. In our case it can happen that the size of P-factorization is of smaller order than the size of y (see the definition of the size of a P-factorization of a word). Both arrays can be, however, computed in linear time with respect to the size of the P-factorization of y. We can prove the following theorem.

Theorem 1. *Let P and Q be two words such that they are of length $O(n)$ and P is primitive and P is not a prefix of Q. Let i be an integer parameter. All solutions in which the variable x is of the form P^iQ can be found in $O(n^2)$ time.*

7 Double-Periodic Solutions

Our goal now is to solve equation E in two variables x, y for which x is known to be of the form $x = (P^iQ)^kP'$ or $x = (P^iQ)^kP^{i+c}P'$, for some $P, P', Q \in \Sigma^n$, P primitive, P not a prefix of Q, constant c and any $i \geq 0$. We split our considerations into n cases for $i = 0, 1, \ldots n-1$ and $i \geq n$. Starting from that point the proof follows the lines the proof of Theorem 1 where P^iQ plays the role of P, k plays the role of i and instead of P-factorization we work on P^iQ-factorizations. We can prove the following theorem.

Theorem 2. *Let P, P' and Q be three words such that they are of length $O(n)$, P is primitive and P is not a prefix of Q. Let i and k be be two integer parameters and c be an integer constant such that $|c| \leq n$. All solutions in which the variable x is of the form*

$$(P^iQ)^kP'$$

or of the form

$$(P^iQ)^kP^{i+c}P'$$

can be found in time $O(n^3)$.

8 Canonization

We revise a data structure that allows for efficient comparison of concatenated coefficients. Let Π be a set of words over an alphabet Σ of finite size and of total length $\sum_{u\in\Pi}|u| = n$. We consider two words $u = u_1 \cdot \ldots \cdot u_k$ and $v = v_1 \cdot \ldots \cdot v_l$ where $u_i, v_j \in \Pi$ and k, l are fixed. Our aim is to verify quickly whether u is a prefix of, or equal to, v. We follow the reasoning introduced originally in Section 4 of [3].

Proposition 3 (Dąbrowski, Plandowski [3]). *Given a finite set Π of words over an alphabet Σ and of total length $n = \sum_{u\in\Pi}|u|$, after an $O(n)$-time preprocessing it is possible to answer in time $O(1)$ if for given a, b being some prefixes of words in Π it is true, that a is a prefix of b.*

Definition 7 (Dąbrowski, Plandowski [3]). *For given set Π of words by a* prefix array *$Pref[u, j]$ for a word $u \in \Pi$ and $1 \le j \le |u|$ we mean the longest prefix of a word in Π which occurs at position j in u.*

Proposition 4 (Dąbrowski, Plandowski [3]). *Given a finite set Π of words of total length $n = \sum_{u\in\Pi}|u|$ and over an alphabet Σ, it is possible to construct the prefix array for Π in time $O(n)$.*

Remark 1. It clearly follows from the propositions, that after $O(n)$-time preprocessing it is possible to answer in constant time whether $u \in \Pi$ starts at position i in $v \in \Pi$.

We say that a word equation is in *canonical* form if its sides start with different variables and end with different variables. Now, we show how we transform an input word equation to its canonical form. If both parts of the equation start or end with the same symbol (constant or variable) then the symbol is reduced. Another case is when one side starts (ends) with a variable and the other starts (ends) with the same variable preceeded by a coefficient.

$$E : A \cdot x \cdot \ldots = x \ldots$$

In such case A is clearly a period of x and the case results in a set of $O(n)$ *unary generators* representing $Sol_x(E)$. The only difficult part is one when one side starts (ends) with a variable and the other starts (ends) with the other variable preceeded by a coefficient.

$$E : A \cdot y \cdot \ldots = x \ldots$$

In such case a set of $O(n)$ *constants* representing $Sol_x(E)$ which are prefices of A is considered first and then a substitution $x := Ax$ is executed. Now, both sides of the equation start with different variables. Similarly, as above we proceed with ends of sides of the equation. Now, the equation is in canonical

form. However its size can be quadratic with respect to the size of the original equation if the constant A in the substitution is large. This is why we do not apply the substitutions directly. Instead, we put before or after each occurrence of an appropriate variable an abbreviation which tells that it is the place for a constant A. Now, for such an equation, using the data structures we said about, we can, for instance, verify in linear time whether a pair of words x, y is a solution although the equation can represent an equation of quadratic size. Similarly, we can find a representation of a P-factorizations of all constants of the equation in linear time although the total size of the factorizations given explicitely can be larger than linear.

9 Main Result

Let E be a two-variable word equation in canonical form, namely it starts and ends with distinct variables. Fix $A(x) = [xA_i]_i$ and $B(x) = [B_j x]_j$ to be the longest respective sequences in one variable.

$$E : A(x) \cdot y \cdot \phi(x, y) = y \cdot B(x) \cdot B'y \cdot \psi(x, y)$$

The case of $|x| = |y|$ leads immediately to a one-variable word equation, which we can handle efficiently as described in [3]. Hence, due to the problem's symmetry, fix $(x, y) \in Sol(E)$ such that $|x| < |y|$. Since the equation is in canonical form then x is both a prefix and suffix of y.

The algorithm that solves E is iterated. A single iteration splits E into E' and E'' and either returns a representation of $Sol_x(E')$ by means of generators or reduces E' to an equation of system S_3. In the former case the algorithm we use the results of Section 6 and Section 7. In the latter case it follows to iterate with E''. Finally, the iterations either result in a system S_2 or a single equation S_3.

Denote by $|A(x)|_x$ the number of occurrences of the variable x in $A(x)$. Similarly, denote by $|B(x)|_x$ the number of occurrences of the variable x in $B(x)$. To perform a single iteration three cases need to be considered.

9.1 $|A(x)|_x \leq |B(x)|_x$

We consider the shortest $B'(x) \leq B(x) = B'(x)B''(x)$ such that $|B'(x)|_x = |A(x)|_x$ and $B'(x)$ ends with x.

If $|A(x)| = |B'(x)|$ then we reduce the problem to solving a singleton equation $E' : A(x)y = yB'(x)$. We either terminate with a representation of $Sol_x(E')$ by means of generators or reduce E' to a system S_3 and iterate with an equation $E'' : \phi(x, y) = B''(x)B'y\psi(x, y)$. In the latter case it either can be shortened to a canonical form or $O(n)$ constant candidates for x can be found or a period of x can be found and $Sol_x(E'')$ can be represented by $O(n)$ unary generators $x_j = P^j Q$, P, Q of lengths $O(n)$, P primitive, $j \geq 0$.

If $|B(x)| + |x| \geq |A(x)| > |B(x)|$, then we reduce the problem to solving a singleton+ equation $E' : A(x)y = yB(x)B'x'$ where $|A(x)| = |B(x)| + |B'x'|$.

We assume x' is nontrivial; otherwise a prefix of B' of length $|A(x)| - |B(x)|$ is a period of x.

If $|A(x)| > |B(x)| + |x|$, then $A(x)y = yC(x)x'$ for some prefix x' of x and the number of occurrences of x in $C(x)$ is bigger than this number in $A(x)$. In such case since the word $C(x)x'$ occurs in $A(x)A(x)$ it occurs only in a simple way, i.e. one of x in $C(x)x'$ touches a constant of $A(x)$. This means that in this case we have $O(n^2)$ unary generators for x.

If $|B(x)| > |A(x)| + |x|$, then $A(x)y = yC(x)x'$ for some prefix x' of x and the number of occurences of x in $C(x)$ is smaller than then this number in $A(x)$. In this case we have the same situation as in the previous one.

If $|A(x)| < |B(x)| \leq |A(x)| + |x|$ then we reduce the problem to solving a singleton- equation $E' : A(x)yA'x' = yB(x)$ where $|A(x)| + |A'x'| = |B(x)|$. We assume x' is nontrivial; otherwise a prefix of A' of length $|B(x)| - |A(x)|$ is a period of x.

9.2 $|A(x)|_x > |B(x)|_x$ and $|y| \geq |A(x)| - |B(x)|$

Since y is long enough, then we consider the shortest prefix $B'(x) < B(x)$ such that $|A(x)|_x = |B(x)|_x + |B'(x)|_x$. We follow the reasoning in the previous case to solve $E' : A(x) \cdot y = B(x) \cdot B' \cdot B'(x)$ or reduce to $E'' : B'(x)\phi(x, y) = y\psi(x, y)$ which is strictly shorter in terms of number of occurrences of y.

9.3 $|A(x)|_x > |B(x)|_x$ and $|y| < |A(x)| - |B(x)|$

Since $|y| < |A(x)|$ then either $y = A'(x)A'$ or $y = A'(x)Ax'$ for certain prefix $A'(x) < A(x)$. In the former case A' is a period of x; there is a total number of $O(n)$ possible periods and they yield $O(n^2)$ unary generators that represent $Sol_x(E)$. Therefore, we assume y ends with x'. There are $O(n)$ possible ways to choose $A'(x)$. Fix $A'(x)$.

We consider now the end of the equation. By symmetry $y = x''C'(x)$ for some $C'(x) = [C_i x]_{i=1}^{k}$. Starting from that point we follow the reasoning in [8]. It is proved there that either we end up with a unary or constants or system S_1 or system S_2 or one special case. The last case can be reduced to $xAy = yBx$ and $A = B$. Then the equation reduces to $(xA)^i y \ldots = y(Ax)^j Ay \ldots$ with $j < i$ and further to $(xA)^{i-j+1}x \ldots = y \ldots$, which is shorter and we consider it in the same manner as the input equation E.

Theorem 3. *Let E be an equations in two variables x, y in canonical form. Then it either reduces to an equation $xAy = yBx$ for some $A, B \in \Sigma^*$ or it is possible to establish, that $(x, y) \in Sol(E)$ only if x is of the following form.*

(α) *A set of $O(n)$ candidates of the form $x = (P^iQ)^j P^{i+b} P'$ or for some P, $P'Q$ of length $O(n)$, P primitive, P not a prefix of Q, $b \leq n$ and $i, j \geq 0$ or of the form $x = (P^iQ)^j P'$ for some P, Q, P' of length $O(n)$ and i, $j \geq 0$.*

(β) *A set of $O(n^3)$ candidates of the form $x = P^iQ$ for some P, Q of lengths $O(n)$, P primitive, P not a prefix of Q and any $i \geq 0$.*

(γ) *A set of* $O(n^2 \log n)$ *candidates* x *of length* $O(n)$.

Therefore combining the theorems presented in the paper, we can find the representation for any equation in two variables. Namely, we reduce the equation to canonized form, establish candidates for one of the solution component, and then solve the original equation by substitution the periodical candidates. The total time to solve the equation is $O(n^5)$.

Theorem 4. *Given an equation E in two variables* x, y, *it is possible to find in time* $O(n^5)$ *a polynomial representation of its solutions.*

References

1. Angluin D., Finding pattern common to a set of string, *in Proc. STOC'79,* 130-141, 1979.
2. Charatonik W., Pacholski L., Word equations in two variables, *Proc. IWWERT'91,* LNCS 677, 43-57, 1991.
3. Dąbrowski R., Plandowski W., On word equations in one variable, *Proc. MFCS'02,* LNCS 2420, 212-221, 2002.
4. Crochemore M., Rytter W., *Text Algorithms,* Oxford University Press, 1994.
5. Diekert V., Makanin's algorithm, Chapter 13 in M. Lothaire, *Algebraic Combinatorics on Words,* Cambridge University Press, 2002.
6. Gutierrez C., Satisfiability of word equations with constants is in exponential space, in: *Proc. FOCS'98,* IEEE Computer Society Press, Palo Alto, California.
7. Hmielevskii Yu.I., Equations in free semigroups, *Proc. Steklov Institute of Mathematics,* Amer. Math. So., 107, 1976.
8. Ilie L., Plandowski W., Two-variable word equations, *RAIRO Theoretical Informatics and Applications* **34,** 467-501, 2000.
9. Jaffar J., Minimal and complete word unification, *Journal of the ACM* **37**(1), 47-85, 1990.
10. Karhumäki J., Mignosi G., Plandowski W., The expressibility of languages and relations by word equations, *Journal of the ACM,* Vol. **47,** No 5, May 2000, pp. 483-505.
11. Koscielski A., Pacholski L., Complexity of Makanin's Algorithm, *Journal of the ACM* **43**(4), 670-684, 1996.
12. Koscielski A., Pacholski L., Makanin's algorithm is not primitive recursive, *Theoretical Computer Science* **191**(1-2):145-156, 1998.
13. Makanin G. S., The problem of solvability of equations in a free semigroup, *Mat. Sb.,* **103**(2), 147-236. In Russian; English translation in: *Math. USSR Sbornik* **32,** 129-198, 1977.
14. Neraud J., Equations in words: an algorithmic contribution, *Bull. Belg. Math. Soc.* **1**, 253-283, 1994.
15. Plandowski W., Rytter W., Application of Lempel-Ziv encodings to the solution of word equations, in: *Proc. ICALP'98,* LNCS 1443, 731-742, 1998.
16. Plandowski W., Satisfiability of word equations with constants is in NEXPTIME, *Proc. STOC'99,* ACM Press, 721-725, 1999.
17. Plandowski W., Satisfiability of word equations with constants is in PSPACE, *Proc. FOCS'99,* IEEE Computer Society Press, 495-500, 1999.
18. Razborov A. A., On systems of equations in a free group, *Izv. Akad. Nauk SSSR,* Ser. Mat. 48:779-832, 1984. In Russian; English translation in: Math. USSR Izvestija, 25, 115-162, 1985.

Backtracking Games and Inflationary Fixed Points

Anuj Dawar[1], Erich Grädel[2], and Stephan Kreutzer[3]

[1] University of Cambridge Computer Laboratory, Cambridge CB3 0FD, UK,
anuj.dawar@cl.cam.ac.uk
[2] Mathematische Grundlagen der Informatik, Aachen-University,
graedel@informatik.rwth-aachen.de
[3] Logik in der Informatik, Humboldt-University, Berlin,
kreutzer@informatik.hu-berlin.de

Abstract. We define a new class of games, called *backtracking games.* Backtracking games are essentially parity games with an additional rule allowing players, under certain conditions, to return to an earlier position in the play and revise a choice.
This new feature makes backtracking games more powerful than parity games. As a consequence, winning strategies become more complex objects and computationally harder. The corresponding increase in expressiveness allows us to use backtracking games as model checking games for inflationary fixed-point logics such as IFP or MIC. We identify a natural subclass of backtracking games, the *simple games,* and show that these are the "right" model checking games for IFP by a) giving a translation of formulae φ and structures $\mathfrak{A}$ into simple games such that $\mathfrak{A} \models \varphi$ if, and only if, Player 0 wins the corresponding game and b) showing that the winner of simple backtracking games can again be defined in IFP.

1 Introduction

The view of logic as a dialectic game, a set of rules by which a proponent attempts to convince an opponent of the truth of a proposition, has deep roots going back to Aristotle. One of the modern manifestations of this view is the presentation of the semantics of logical operators as moves in a two-player game. A paradigmatic example is the Hintikka semantics of first-order logic, which is just one instance of what are now commonly called *model-checking games.* These are two-player games played on an arena which is formed as the product of a structure $\mathfrak{A}$ and a formula φ where one player attempts to prove that φ is satisfied in $\mathfrak{A}$ while the other player attempts to refute this.

Model-checking games have proved an especially fruitful area of study in connection with logics for the specification of concurrent systems. The modal μ-calculus L_μ is widely used to express properties of such systems and, in terms of expressive power it subsumes a variety of common modal and temporal logics. The most effective algorithms for model checking properties specified in L_μ are based on *parity games.* Formally, a parity game is played on an arena $\mathcal{G} := (V, E, V_0, V_1, \Omega)$, where *(V, E)* is a directed graph, $V_0, V_1 \subseteq V$ form a partition

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 420–432, 2004.

of V, and $\Omega : V \to \{0, \ldots, k-1\}$ assigns to each node a priority. The two players move a token around the graph, with Player 0 moving when the token is on a node in V_0 and Player 1 when it is on V_1. The edges E determine the possible moves. To determine the winner, we look at the sequence of priorities $\Omega(v_i)$ occurring in an infinite play $v_0 v_1 \ldots$. Player 0 wins if the smallest priority occurring infinitely often is even and Player 1 wins if it is odd.

Parity games are the model-checking games not just for L_μ but also of LFP—the extension of first-order logic with an operator for forming relational least fixed points. That is, for any formula φ of LFP and any structure $\mathfrak{A}$ one can easily construct a game $\mathcal{G}(\mathfrak{A}, \varphi)$ where Player 0 has a winning strategy if, and only if, the formula φ is satisfied in $\mathfrak{A}$. The game arena is essentially obtained as the product of $\mathfrak{A}^w$ and φ, where w is the width of the formula—the maximal arity of a relation defined by a subformula of φ. Furthermore, for any fixed number k, the class of parity games with k priorities in which Player 0 has a winning strategy is itself definable in L_μ and therefore by an LFP formula of width 2. This tight correspondence between games and the fixed-point logic leads us to describe parity games as the "right" model-checking games for LFP.

LFP is not the only logic that extends first-order logic with a means of forming fixed points. In the context of finite model theory, a rich variety of fixed-point operators has been studied due to the close connection that the resulting logics have with complexity classes. Here we are mainly concerned with IFP, the logic of *inflationary fixed points* (see Section 3 for a definition). In the context of finite model theory the logics IFP and LFP have often been used interchangeably as it has long been known that they have equivalent expressive power on finite structures. More recently, it has been shown that the two logics are equally expressive even without the restriction to finite structures [6]. However, it has also recently been shown that the extension of propositional modal logic is vastly more expressive than L_μ [1] and that LFP and IFP have very different structural properties even when they have the same expressive power [6]. This exploration of the different nature of the fixed-point operators leads naturally to the question of what an appropriate model-checking game for IFP might look like.

The correspondence between parity games and logics with least and greatest fixed point operators rests on the structural property of *well-foundedness.* A proponent in a game who is trying to prove that a certain element x belongs to a least fixed point X, needs to present a well-founded justification for its inclusion. That is, the inclusion of x in X may be based on the inclusion of other elements in X whose inclusion in turn needs to be justified but the entire process must be well-founded. On the other hand, justification for including an element in a greatest fixed point may well be circular. This interaction between sequences that are required to be finite and those that are required to be infinite provides the structural correspondence with parity games.

A key difference that arises when we consider inflationary fixed points (and, dually, deflationary fixed points) is that the stage at which an element x enters the construction of the fixed point X may be an important part of the justification for its inclusion. In the case of least and greatest fixed points, the operators

involved are monotone. Thus, if the inclusion of x can be justified at some stage, it can be justified at all later stages. In contrast, in constructing an inflationary fixed point, if x is included in the set, it is on the basis of the immediately preceding stage of the iteration. It may be possible to reflect this fact in the game setting by including the iteration stage as an explicit component of the game position. However, our aim is to leave the notion of the game arena unchanged as the product of the structure and the formula. We wish only to change the rules of the game to capture the nature of the inflationary fixed point operator.

The change we introduce to parity games is that either player is allowed to *backtrack* to an earlier position in the game, effectively to force a *countback* of the number of stages. That is, when a backtracking move is played, the number of positions of a given priority that are backtracked are counted and this count plays an important role in the succeeding play. The precise definition is given in Section 3 below. The backtracking games we define are far more complex than parity games. We prove that winning strategies are necessarily more complicated, requiring unbounded memory, in contrast to the memoryless strategies that work for parity games. Furthermore, deciding the winner is PSPACE-hard and remains hard for both NP and Co-NP with only two priorites. In contrast, parity games are known to be decidable in $\mathrm{NP} \cap \mathrm{Co\text{-}NP}$ and in PTIME when the number of priorities is fixed. In Section 3 we show that the model-checking problem for IFP can be represented in the form of backtracking games. The construction allows us to observe that a simpler form of backtracking game suffices which we call *simple* backtracking games. In Section 4 we show that in IFP we can define the class of simple backtracking games that are won by Player 0. Thus, we obtain a tight correspondence between the game and the logic, as exists between LFP and parity games.

2 Games with Backtracking

Backtracking games are essentially parity games with the addition that, under certain conditions, players can jump back to an earlier position in the play. This kind of move is called backtracking.

A backtracking move from position v to an earlier position u is only possible if v belongs to a given set B of backtrack positions, if u and v have the same priority and if no position of smaller priority has occurred between u and v. With such a move, the player who backtracks not only resets the play back to u, he also commits herself to a backtracking distance d, which is the number of positions of priority $\Omega(v)$ that have been seen between u and v. After this move, the play ends when d further positions of priority $\Omega(v)$ have been seen, unless this priority is "released" by a lower priority.

For finite plays we have the winning condition that a player wins if her opponent cannot move. For infinite plays, the winner is determined according to the parity condition, i.e., Player 0 wins a play π if the least priority seen infinitely often in π is even, otherwise Player 1 wins.

Definition 2.1. *The arena* $\mathcal{G} := (V, E, V_0, V_1, B, \Omega)$ *of a backtracking game is a directed graph (V,E), with a partition* $V = V_0 \cup V_1$ *of V into positions of Player 0 and positions of Player 1, a subset* $B \subseteq V$ *of backtrack positions and a map* $\Omega : V \to \{0, \dots, k-1\}$ *that assigns to each node a priority.*

In case $(v, w) \in E$ we call w a successor of v and we denote the set of all successors of v by vE. A *play* of $\mathcal{G}$ from initial position v_0 is formed as follows. If, after n steps the play has gone through positions $v_0 v_1 \dots v_n$ and reached a position $v_n \in V_\sigma$, then Player σ can select a successor $v_{n+1} \in v_n E$; this is called an ordinary move. But if $v_n \in B$ is a backtrack position, of priority $\Omega(v_n) = q$, say, then Player σ may also choose to backtrack; in that case she selects a number $i < n$ subject to the conditions that $\Omega(v_i) = q$ and $\Omega(v_j) \geq q$ for all j with $i < j < n$. The play then proceeds to position $v_{n+1} = v_i$ and we set $d(q) = |\{k : i \leq k < n \wedge \Omega(v_k) = q\}|$. This number $d(q)$ is relevant for the rest of the game, because the play ends when $d(q)$ further positions of priority q have been seen without any occurrence of a priority $< q$. Therefore, a play is not completely described by the sequence $v_0 v_1 \dots$ of the positions that have been visited. For instance, if a player backtracks from v_n in $v_0 \dots v_i \dots v_j \dots v_n$, it matters whether she backtracks to i or j, even if $v_i = v_j$ because the associated numbers $d(p)$ are different.

We now proceed to a more formal description of how backtracking games are played. We distinguish therefore between the notion of a *(partial) play,* which is a word $\pi \in (V \cup \mathbb{N})^{\leq \omega}$ and the sequence $\mathrm{path}(\pi)$ of nodes visited by π. Further, we associate with every partial play π a function $d_\pi : \{0, \dots, k-1\} \to \mathbb{N} \cup \{\infty\}$ associating with every priority p the distance $d_\pi(p)$. Here $d(p) = \infty$ means that p is not active; either there never has been a backtracking move of priority p, or the priority p has since been released by a smaller priority. Every occurrence of a node with priority p decrements $d_\pi(p)$, with the convention that $\infty - 1 = \infty$. A play π cannot be extended if $d_\pi(p) = 0$ for some p.

Definition 2.2 (Playing backtracking games). *Let* $\mathcal{G} = (V, E, V_0, V_1, B, \Omega)$ *be a backtracking game with priorities* $\{0, \dots, k-1\}$, *and* $v_0 \in V$. *The set of* partial plays π *from position* v_0, *together with the associated sequence* $\mathrm{path}(\pi)$ *of the visited positions and the distance function* $d_\pi : \{0, \dots, k-1\} \to \mathbb{N} \cup \{\infty\}$, *are inductively defined as follows.*

start: v_0 *is a partial play, with* $\mathrm{path}(v_0) = v_0$, *and* $d_{v_0}(p) = \infty$ *for all* p.

ordinary move: *If* π *is a partial play with* $d_\pi(p) > 0$ *for all* p, $\mathrm{path}(\pi) = v_0 \dots v_n$ *and* $v_n \in V_\sigma$, *then Player* σ *can extend* π *to* πv *for each* $v \in v_n E$; *Further,* $\mathrm{path}(\pi v) = \mathrm{path}(\pi) v$ *and* $d_{\pi v}(p) := d_\pi(p)$ *for* $p < \Omega(v)$, $d_{\pi v}(p) := d_\pi(p) - 1$ *for* $p = \Omega(v)$, *and* $d_{\pi v}(p) := \infty$ *for* $p > \Omega(v)$.

backtracking move: *Suppose that* π *is a partial play with* $d_\pi(p) > 0$ *for all* p *and that* $\mathrm{path}(\pi) = v_0 \dots v_n$ *with* $v_n \in V_\sigma \cap B$, $\Omega(v_n) = q$, *and* $d_\pi(q) = \infty$. *Then Player* σ *can extend* π *to* πi *for any number* $i < n$ *such that* $\Omega(v_i) = q$ *and* $\Omega(v_k) \geq q$ *for all* k *with* $i < k < n$. *Further* $\mathrm{path}(\pi i) = \mathrm{path}(\pi) v_i$ *and* $d_{\pi i}(p) := d_\pi(p)$ *for* $p < q$, $d_{\pi i}(p) := |\{k : i \leq k < n : \Omega(v_k) = q\}|$ *for* $p = q$, *and* $d_{\pi i}(p) := \infty$ *for* $p > q$.

Definition 2.3 (Winning condition). *A partial play* π *with* $\mathrm{path}(\pi) = v_0 \dots v_n$ *is won by Player* σ*, if* $v_n \in V_{1-\sigma}$ *and no move is possible. This is the case if either* $d_\pi(p) = 0$ *for some* p*, or if* $v_n E$ *is empty and no backtracking move is possible from* π*. An infinite play* π *is won by Player* 0 *if the smallest priority occurring infinitely often on* $\mathrm{path}(\pi)$ *is even; otherwise* π *is won by Player 1.*

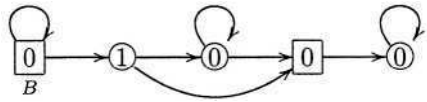
A game is *determined* if from each position one of the two players has a winning strategy. Determinacy of backtracking games follows from general facts on infinite games. Indeed, by Martin's Theorem [7] all Borel games are determined, and it is easy to see that backtracking games are Borel games.

Proposition 2.4. *Backtracking games are determined.*

Backtracking games generalise parity games. Indeed a parity game is a backtracking game without backtrack positions. Since parity games are determined via positional (i.e. memoryless) winning strategies, the question arises whether this also holds for backtracking games. We present a simple example to show that this is not the case. In fact, no fixed amount of finite memory suffices. For background on positional and finite-memory strategies we refer to [5].

Theorem 2.5. *Backtracking games in general do not admit finite-memory winning strategies.*

Proof. Consider the following game (where circles are positions of Player 0 and boxes are positions of Player 1).



We claim that Player 0 wins from the leftmost position, but needs infinite memory to do so. Clearly, if Player 1 never leaves the leftmost position, or if she leaves it before doing a backtracking move, then Player 0 wins seeing priority 0 infinitely often. If Player 1 at some point backtracks at the leftmost position and then moves on, the strategy of Player 0 depends on the value of $d(0)$ to make sure that the fourth node is hit at the point when $d(0) = 0$. But Player 1 can make $d(0)$ arbitrarily large, no finite-memory strategy suffices for Player 0. □

This result establishes that winning strategies for backtracking games are more complex than the strategies needed for parity games. It is also the case that the computational complexity of deciding which player has a winning strategy is also higher for backtracking games than for parity games. While it is known that winning regions of parity games can be decided in NP ∩ Co-NP (and it is conjectured by many, that this problem is actually solvable in polynomial time), the corresponding problem for backtracking games is PSPACE-hard. Further, for any fixed number of priorities, parity games can be decided in PTIME, but there are examples of backtracking games with just two priorities that are NP-hard. The proof is by reduction from the language equivalence problem for

finite automata over a unary alphabet, which is known to be Co-NP-hard [2]. As the problem of deciding the winner of a backtracking game is closed under complementation, it is also NP-hard.

Theorem 2.6. *Deciding the winner of backtracking games is* Co-NP *and* NP-*hard, even for games with only two priorities.*

3 Model Checking Games for Inflationary Fixed Point Logic

In this section we want to show that backtracking games can be used as model checking games for inflationary fixed point logics. We will present the games in terms of IFP, the extension of first-order logic by inflationary and deflationary fixed points, but the construction applies, with the obvious modifications, also to the modal iteration calculus MIC [1].

Inflationary fixed point logic. A formula $\varphi(R, \boldsymbol{x})$ with a free k-ary second-order variable and a free k-tuple of first-order variables $\boldsymbol{x}$ defines, on every structure $\mathfrak{A}$, a relational operator $F_\varphi : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ taking $R \subseteq A^k$ to the set $\{\boldsymbol{a} : (\mathfrak{A}, R) \models \varphi(\boldsymbol{a})\}$. Fixed point extensions of first-order logic are obtained by adding to FO explicit constructs to form fixed points of definable operators. The type of fixed points that are used determines the expressive power and also the algorithmic complexity of the resulting logics. The most important of these extensions are least fixed point logic (LFP) and inflationary fixed point logic (IFP).

The inflationary fixed point of any operator $F : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ is defined as the limit of the increasing sequence of sets $(R^\alpha)_{\alpha \in \mathrm{Ord}}$ defined as $R^0 := \varnothing$, $R^{\alpha+1} := R^\alpha \cup F(R^\alpha)$, and $R^\lambda := \bigcup_{\alpha<\lambda} R^\alpha$ for limit ordinals λ. The *deflationary fixed point* of F is constructed in the dual way starting with A^k as the initial stage and taking intersections at successor and limit ordinals.

Definition 3.1. Inflationary fixed-point logic (IFP) *is obtained from* FO *by allowing formulae of the form* $[\mathbf{ifp}\, R\boldsymbol{x} \,.\, \varphi(R, \boldsymbol{x})](\boldsymbol{x})$ *and* $[\mathbf{dfp}\, R\boldsymbol{x} \,.\, \varphi(R, \boldsymbol{x})](\boldsymbol{x})$*, for arbitrary* φ*, defining the inflationary and deflationary fixed point of the operator induced by* φ*.*

Model checking games for LFP. Let us recall the definitions of model checking games for least fixed-point logic LFP (the games for the modal μ-calculus are analogous). Consider a sentence $\psi \in \mathrm{LFP}$ which we assume is in negation normal form and *well-named,* i.e. every fixed-point variable is bound only once.

The game $\mathcal{G}(\mathfrak{A}, \psi)$ is a parity game whose positions are subformulae of ψ instantiated by elements of $\mathfrak{A}$, i.e. expressions $\varphi(\boldsymbol{a})$ such that $\varphi(\boldsymbol{x})$ is a subformula of ψ, and $\boldsymbol{a}$ a tuple of elements of $\mathfrak{A}$. Player 0 (Verifier) moves at positions associated with disjunctions and formulae $\exists y \varphi(\boldsymbol{a}, y)$. From a position $(\varphi \vee \vartheta)(\boldsymbol{a})$ she moves to either $\varphi(\boldsymbol{a})$ or $\vartheta(\boldsymbol{a})$ and from a position $\exists y \varphi(\boldsymbol{a}, y)$ she can move to any position $\varphi(\boldsymbol{a}, b)$ such that $b \in \mathfrak{A}$. In addition, Verifier is supposed to move

at atomic false positions, i.e., at positions Ra where $a \notin R^{\mathfrak{A}}$ and $\neg Ra$ where $a \in R^{\mathfrak{A}}$. However, these positions do not have successors, so Verifier loses at atomic false positions. Dually, Player 1 (Falsifier) moves at conjunctions and formulae $\forall y \varphi(a, y)$, and loses at atomic true positions. The rules described so far determine the model checking game for FO-formulae ψ and it is easily seen that Verifier has a winning strategy in this game $\mathcal{G}(\mathfrak{A}, \psi)$ starting at a position $\varphi(a)$ if, and only if, $\mathfrak{A} \models \varphi(a)$.

For formulae in LFP, we also have positions $[\mathbf{fp}\, Tx.\varphi](a)$ (where **fp** stands for either **lfp** or **gfp**) and Ta, for fixed-point variables T. At these positions there is a unique move (by Falsifier, say) to $\varphi(a)$, i.e. to the formula defining the fixed point. The priority labelling assigns even priorities to **gfp**-atoms Ta and odd priorities to **lfp**-atoms Ta. Further, if T, T' are fixed-point variables of different kind with T' depending on T (which means that T occurs free in the formula defining T'), then T-positions get lower priority than T'-positions. The remaining positions, not associated with fixed-point variables, do not have a priority (or have the maximal one). As a result, the number of priorities in the model checking game equals the alternation depth of the fixed-point formula plus one. For more details and explanations, and for the proof that the construction is correct, see e.g. [3,8].

Theorem 3.2. $\mathfrak{A} \models \psi$ *if, and only if, Verifier has a winning strategy for the parity game* $\mathcal{G}(\mathfrak{A}, \psi)$ *from position* ψ.

Games for IFP. We restrict attention to finite structures. The model checking game for an IFP-formula ψ on a finite structure $\mathfrak{A}$ is a backtracking game $\mathcal{G}(\mathfrak{A}, \psi) = (V, E, V_0, V_1, B, \Omega)$. As in the games for LFP, the positions are subformulae of ψ, instantiated by elements of $\mathfrak{A}$. We only describe the modifications.

We always assume that formulae are in negation normal form, and write $\overline{\vartheta}$ for the negation normal form of $\neg\vartheta$. Consider any **ifp**-formula $\varphi^*(x) := [\mathbf{ifp}\, Tx.\varphi(T, x)](x)$ in ψ. In general, φ can have positive or negative occurrences of the fixed point variable T. We use the notation $\varphi(T, \overline{T})$ to separate positive and negative occurrences of T. To define the set of positions we include also all subformulae of $Tx \vee \varphi(T, x)$ and $\overline{T}x \wedge \overline{\varphi}(T, x)$. From a position $\varphi^*(a)$ the play proceeds to $Ta \vee \varphi(T, a)$. When a play reaches a position Tc or $\overline{T}c$ the play proceeds back to the formula defining the fixed point by a regeneration move. More precisely, the regeneration of an **ifp**-atom Tc is $Tc \vee \varphi(T, c)$, the regeneration of $\overline{T}c$ is $\overline{T}c \wedge \overline{\varphi}(T, c)$. Verifier can move from Tc to its regeneration, Falsifier from $\overline{T}c$. For **dfp**-subformulae $\vartheta^*(x) := [\mathbf{dfp}\, Rx.\vartheta(R, x)](x)$, dual definitions apply. Verifier moves from $\overline{R}c$ to its regeneration $\overline{R}c \vee \overline{\vartheta}(R, c)$, and Falsifier can make regeneration moves from Rc to $Rc \wedge \vartheta(R, c)$. The priority assignment associates with each **ifp**-variable T an odd priority $\Omega(T)$ and with each **dfp**-variable R an even priority $\Omega(R)$, such that for any two distinct fixed point variables S, S', we have $\Omega(S) \neq \Omega(S')$, and whenever S' depends on S, then $\Omega(S) < \Omega(S')$. Positions of the form Sc and $\overline{S}c$ are called S-positions. All S-positions get priority $\Omega(S)$, all other formulae get a higher priority. The set B of backtrack positions is the set of S-positions, where S is any fixed-point variable.

For simplicity we focus on IFP-formulae with a single fixed point, $\psi := [\mathbf{ifp}\, T\boldsymbol{x}\,.\,\varphi](\boldsymbol{a})$ where $\varphi(T, \boldsymbol{x})$ is a first-order formula. When the play reaches a position $T\boldsymbol{c}$ Verifier can make a regeneration move to $T\boldsymbol{c} \vee \varphi(T, \boldsymbol{c})$ or backtrack. Dually, Falsifier can regenerate from positions $\overline{T}\boldsymbol{c}$ or backtrack. However, since we have only one fixed point, all backtrack positions have the same priority and only one backtrack move can occur in a play.

In this simple case, the rules of the backtracking game ensure that infinite plays (which are plays without backtracking moves) are won by Falsifier, since **ifp**-atoms have odd priority. However, if one of the players backtracks after the play has gone through α T-positions, then the play ends when α further T-positions have been visited. Falsifier has won, if the last of these is of form $T\boldsymbol{c}$, and Verifier has won if it is of form $\overline{T}\boldsymbol{c}$.

The differences between IFP model checking and LFP model checking are in fact best illustrated with this simple case. For this reason, and for lack of space, we prove the correctness of the model checking game only for this case, and defer the general case to the full version of this paper.

We claim that Verifier has a winning strategy for the game $\mathcal{G}(\mathfrak{A}, \psi)$ if $\mathfrak{A} \models \psi$ and Falsifier has a winning strategy if $\mathfrak{A} \not\models \psi$.

To prove our claim, we look at the first-order formulae φ^α defining the stages of the induction. Let $\varphi^0(\boldsymbol{a}) = \mathit{false}$ and $\varphi^{\alpha+1}(\boldsymbol{a}) = \varphi^\alpha(\boldsymbol{a}) \vee \varphi[T/\varphi^\alpha, \overline{T}/\overline{\varphi}^\alpha](\boldsymbol{x})$. On finite structures $\psi(\boldsymbol{a}) \equiv \bigvee_{\alpha<\omega} \varphi^\alpha(\boldsymbol{a})$. Consider the situation after a backtracking move prior to which β T-positions have been visited and suppose that $\mathfrak{A} \models \varphi^\beta(\boldsymbol{a})$. A winning strategy for Verifier in the first-order game $\mathcal{G}(\mathfrak{A}, \varphi^\beta(\boldsymbol{a}))$ (from position $\varphi^\beta(\boldsymbol{a})$) translates in the obvious way into a (non-positional) strategy for the game $\mathcal{G}(\mathfrak{A}, \psi)$ from position $\psi(\boldsymbol{a})$ with the following properties: Any play that is consistent with this strategy will either be winning for Verifier before β T-positions have been seen, or the β-th T-position will be negative.

Similarly, if $\mathfrak{A} \not\models \varphi^\beta(\boldsymbol{a})$ then Falsifier has a winning strategy for $\mathcal{G}(\mathfrak{A}, \varphi^\beta(\boldsymbol{a}))$, and this strategy translates into a strategy for the game $\mathcal{G}(\mathfrak{A}, \psi)$ by which Falsifier forces the play (after backtracking) from position $\psi(\boldsymbol{a})$ to a positive β-th T-position, unless she wins before β T-positions have been seen.

Lemma 3.3. *Suppose that a play on $\mathcal{G}(\mathfrak{A}, \psi)$ has been backtracked to the initial position $\psi(\boldsymbol{a})$ after β T-positions have been visited. Verifier has a winning strategy for the remaining game if, and only if, $\mathfrak{A} \models \varphi^\beta(\boldsymbol{a})$.*

From this we obtain the desired result.

Proposition 3.4. *// $\mathfrak{A} \models \psi(\boldsymbol{a})$, then Verifier wins the game $\mathcal{G}(\mathfrak{A}, \psi(\boldsymbol{a}))$ from position $\psi(\boldsymbol{a})$. If $\mathfrak{A} \not\models \psi(\boldsymbol{a})$, then Falsifier wins the game $\mathcal{G}(\mathfrak{A}, \psi(\boldsymbol{a}))$ from position $\psi(\boldsymbol{a})$.*

Proof. Suppose first that $\mathfrak{A} \models \psi(\boldsymbol{a})$. Then there is some ordinal $\alpha < \omega$ such that $\mathfrak{A} \models \varphi^\alpha(\boldsymbol{a})$. We construct a winning strategy for Verifier in the game $\mathcal{G}(\mathfrak{A}, \psi(\boldsymbol{a}))$ starting at position $\psi(\boldsymbol{a})$.

From $\psi(\boldsymbol{a})$ the game proceeds to $(T\boldsymbol{a} \vee \varphi(\boldsymbol{a}))$. At this position, Verifier repeatedly chooses the node $T\boldsymbol{a}$ until this node has been visited α-times. After

that, she backtracks and moves to $\varphi(a)$. By Lemma 3.3 and since $\mathfrak{A} \models \varphi^\alpha(a)$, Verifier has a strategy to win the remaining play.

Now suppose that $\mathfrak{A} \not\models \psi(a)$. If, after α T-positions, one of the players backtracks, then Falsifier has a winning strategy for the remaining game, since $\mathfrak{A} \not\models \varphi^\alpha(a)$. Hence, the only possibility for Verifier to win the game in a finite number of moves is to avoid positions $\overline{T}b$ where Falsifier can backtrack.

Consider the formulae φ_f^α, with $\varphi_f^0 = \mathit{false}$ and $\varphi_f^{\alpha+1}(x) = \varphi[\varphi_f^\alpha, \mathit{false}](x)$. They define the stages of $[\mathbf{ifp}\, Tx.\varphi[T, \mathit{false}](x)]$, obtained from ψ by replacing negative occurrences of T by *false*. If Verifier could force a finite winning play, with $\alpha - 1$ positions of the form Tc and without positions $\overline{T}c$, then she would in fact have a winning strategy for the model checking game $\mathcal{G}(\mathfrak{A}, \varphi_f^\alpha(a))$. Since ψ_f^α implies φ^α, it would follow that $\mathfrak{A} \models \varphi^\alpha(a)$. But this is impossible. □

4 Definability of Backtracking Games

In the previous section we demonstrated that backtracking games can be used as model-checking games for IFP. The aim of this section is to show that they are, in some sense, the "right" model-checking games for inflationary fixed-point logics. For this, we identify a natural sub-class of backtracking games, which we call *simple,* such that for every formula $\varphi \in \mathbf{IFP}$ and finite structure $\mathfrak{A}$, the game $\mathcal{G}(\mathfrak{A}, \varphi)$ can be trivially modified to fall within this class and, on the other hand, for every $k \in \mathbb{N}$ there is a formula $\varphi \in \mathbf{IFP}$ defining the winning region for Player 0 in any simple game with at most k priorities. In this sense, simple backtracking games precisely capture IFP model-checking.

Consider again the proof given in Section 3 for winning strategies in a game $\mathcal{G}(\mathfrak{A}, \varphi)$ and the way backtracking was used there: If Player 0 wanted to backtrack it was always after opening a fixed point, say $[\mathbf{ifp}\, Rx.Rx \vee \varphi]$. She then looped α times through the Rx sub-formula and backtracked. With choosing the α she essentially picked a stage of the fixed-point induction on φ and claimed that $x \in \varphi^\alpha$. From this observation we can derive two important consequences. As every inflationary fixed-point induction must close after polynomially many steps in the size of the structure $\mathfrak{A}$ and therefore in linearly many steps in terms of the game graph, there is no need for Player 0 to backtrack more than n steps, where n is the size of the game graph. Further, the game can easily be modified such that instead of having the nodes for the disjunction $Rx \vee \varphi$ and the sub-formula Rx, we simply have a node for φ with a self-loop. In this modified game graph, not only is it sufficient for Player 0 to backtrack no more than n steps, we can, in addition, require that whenever she backtracks from a node v, it must be to v again, i.e. when she decides to backtrack from a node corresponding to the formula φ, she loops α times through φ and then backtracks α steps to φ again. The same is true for Player 1 and her backtracking.

Definition 4.1. *A strategy in a backtracking game $\mathcal{G}$ is* local *if, for any backtracking node v, all backtracking moves from v are to a previous occurrence of v. Given a function $f : \mathbb{N} \to \mathbb{N}$, we call a strategy* f-backtracking *if all backtracking*

moves made by the strategy have distance at most $f(|\mathcal{G}|)$. *The strategy is called* linear *in case* $f(n) = n$ *and* polynomial *if* f *is a polynomial in* n.

As explained above, we can easily modify the construction of the game graph $\mathcal{G}(\mathfrak{A}, \varphi)$ for a formula φ and structure $\mathfrak{A}$ such that every node in B has a self loop. We call such game graphs *inflationary*.

Definition 4.2. *A backtracking game* $\mathcal{G} := (V, E, V_0, V_1, B, \Omega)$ *is* inflationary, *if every node in* B *has a self-loop. An inflationary game* $\mathcal{G}$ *is called* simple *if both players have local linear winning strategies on their winning regions.*

Proposition 4.3. *For any* IFP*-formula* ψ *and every finite structure* $\mathfrak{A}$, *the model-checking game* $\mathcal{G}(\mathfrak{A}, \varphi)$, *as defined in Section 3, is simple.*

We will construct IFP-formulae defining the winning regions of simple backtracking games. Since backtracking games are extensions of parity games we start with the formula defining winning regions in parity games (see [9]). Let $\mathcal{G}$ be a parity game with $k+1$ priorities and consider the formula $\varphi(x) := [\mathbf{gfp}\, R_0 x\,.\, \mathbf{lfp}\, R_1 x\,. \ldots \mathbf{fp}\, R_k x\,.\, \vartheta(x, R_0, \ldots, R_k)](x)$, where

$$\vartheta(x, R_0, \ldots, R_k) := \bigwedge_{i=0}^{k}(V_0 x \wedge \Omega(y) = i \to \exists y\,(Exy \wedge R_i y)) \wedge \bigwedge_{i=0}^{k}(V_1 x \wedge \Omega(y) = i \to \forall y\,(Exy \to R_i y)).$$

For every node $v \in V$, we have that $\mathcal{G} \models \varphi(v)$ if, and only if, Player 0 has a winning strategy for the game $\mathcal{G}$ from v. A simple way to see this is to analyse the model checking game for $\varphi(v)$ on $\mathcal{G}$. If we remove the edges which would force a player to lose immediately, we obtain $\mathcal{G}$ itself (from position v).

We take this formula as a starting point for defining an IFP-formula deciding the winner of backtracking games. To define strategies involving backtracking, we first need some preparation. In particular, in order to measure distances we need an ordering on the arenas.

It is easily seen that backtracking games are invariant under bisimulation. Thus, it suffices to consider arenas where no two distinct nodes are bisimilar (we refer to such arenas as *bisimulation minimal*). The next step is to define an ordering on the nodes in an arena. This is done by ordering the bisimulation types realised in it.

Lemma 4.4. *There is a formula* $\varphi_{ord}(x, y) \in$ IFP *defining on every bisimulation minimal arena a linear order.*

This is well-known in finite model theory. For an explicit construction, see e.g. [4]. As a result, we can assume that the backtracking games are ordered and that we are given an arithmetical predicate for addition with respect to the order defined above.

In Theorem 2.5 we exhibited a backtracking game that requires infinite memory strategies. All strategies in this game are necessarily local strategies. Thus Theorem 2.5 also applies to games with local strategies. The reason for the increased memory consumption is that when the decision to backtrack is made, it is necessary to know which nodes have been seen in the past, i.e. to which

node a backtracking move is possible. However, since we here consider strategies with local backtracking only, it suffices to know the backtracking moves that are still active, i.e. have not yet been released. Thus we can capture all the relevant information about a partial play π ending in position v by the tuple $(v, d_\pi(0), \ldots, d_\pi(k))$. This is formalised in the notion of a *configuration.*

Definition 4.5. *Let $\mathcal{G}$ be a backtracking game with $k+1$ priorities. A* configuration *is a pair $(v, \boldsymbol{d})$ consisting of a node v and a tuple $\boldsymbol{d} \in \{0, \ldots, k, \infty\}^{k+1}$. Let π be a (partial) play ending in node v. The* configuration of π *is the tuple $(v, d_0, \ldots, d_k)$ such that $d_i := d_\pi(i)$ for all $i \leq k$.*

We are now ready to present a formula defining the winning region for Player 0 in a simple backtracking game with priorities $0, \ldots, k$. The structure of the formula is similar to the structure of $\varphi(x)$ for parity games, in the sense that for games with $k+1$ priorities we have $k+1$ nested fixed points of the form $\mathbf{gfp}\, R_0 x\boldsymbol{d}.\ \mathbf{lfp}\, R_1 x\boldsymbol{d}. \ldots \mathbf{fp}\, R_k x\boldsymbol{d}$ and a ψ which is first-order, up to the IFP-subformula defining the order of the bisimulation types. In its various nested fixed points the formula builds up sets of configurations $(x, d_0, \ldots, d_k)$ such that if $(x, d_0, \ldots, d_k) \in R_{\Omega(x)}$, then Player 0 can extend any partial play π, ending in node x with $d_\pi(j) = d_j$ for all $0 \leq j \leq k$, to a winning play. As the d_i range over 0 to $n := |\mathcal{G}|$ and also may take the value ∞, we would, strictly speaking, need to encode each d_i by a pair of elements. However, to simplify notation, we only use one variable for each d_i and allow it to take all possible values. We also use a constant ∞ and variables $\mathtt{i}, \mathtt{j}, \ldots$ for constants between 0 and k. Finally, in the case distinctions below we write $d_i = m$ for $\exists m \in \{0, \ldots, n\} \wedge d_i = m$.

The inner formula ψ is split in two parts $\psi_0 \vee \psi_1$ taking care of positions where Player 0 moves and positions where Player 1 moves. We first present the formula $\psi_0(x, R_0, \ldots, R_k)$ defining positions in V_0 from which Player 0 can win.

$$
\begin{aligned}
\psi_0(x, \boldsymbol{d}) := \; & V_0 x \wedge \bigvee_{\mathtt{i}} \Omega(x) = \mathtt{i} \wedge \textstyle\bigwedge_{l=i+1}^{k} d_l = \infty \wedge \\
& \exists y \exists \boldsymbol{d}' E x y \wedge \bigvee_{\mathtt{j}} \Omega(y) = \mathtt{j} \wedge R_{\mathtt{j}} y \boldsymbol{d}' \wedge \\
& d_i = \infty \wedge \mathtt{j} \geq \mathtt{i} \wedge \boldsymbol{d} = \boldsymbol{d}' \vee \\
& \qquad \mathtt{j} < \mathtt{i} \wedge \boldsymbol{d}' = (d_0, \ldots, d_j, \infty, \ldots, \infty) \vee \\
& \qquad Bx \wedge \exists m \neq \infty\, R_i(x, d_0, \ldots, d_{i-1}, m, \infty, \ldots, \infty) \vee \\
& d_i = m \wedge \; \mathtt{j} > \mathtt{i} \wedge \boldsymbol{d}' = \boldsymbol{d} \vee \\
& \qquad \mathtt{j} = \mathtt{i} \wedge \boldsymbol{d}' = (d_0, \ldots, d_{i-1}, d_i - 1, \infty, \ldots, \infty) \vee \\
& \qquad \mathtt{j} < \mathtt{i} \wedge \boldsymbol{d}' = (d_0, \ldots, d_j, \infty, \ldots, \infty)
\end{aligned}
$$

The first line of the formula states that x has to be in V_0, the priority of x is i, for some i, and the tuple $(d_0, \ldots, d_k)$ has ∞ at all positions greater than i. This corresponds to the fact that a node of priority i releases all backtracking moves on higher priorities. Now, Player 0 can win from configuration $(x, \boldsymbol{d})$ if she can move to a successor y of x from which she wins the play. That she can win from y means that if $(y, \boldsymbol{d}')$ is the configuration reached when she moves from $(x, \boldsymbol{d})$ to y, then $(y, \boldsymbol{d}') \in R_{\Omega(y)}$. The second row of the formula states the existence of such a successor y and the rest of the formula defines what it means for $(y, \boldsymbol{d}')$ to be the configuration reached from x when moving to y.

The next formula ψ_1 takes care of nodes $x \in V_1$.

$$\begin{aligned}\psi_1(x,\boldsymbol{d}) := \; & V_1 x \wedge \bigvee_{\mathtt{i}} \Omega(x) = \mathtt{i} \wedge \bigwedge_{l=i+1}^{k} d_l = \infty \wedge \\ & (Bx \to \forall m < \infty \, R_i(x, d_0, \ldots, d_{i-1}, m, \infty, \ldots, \infty)) \wedge \\ & \forall y (Exy \to \bigvee_{\mathtt{j}} \Omega(y) = \mathtt{j} \wedge \exists \boldsymbol{d}' \, R_{\mathtt{j}} y \boldsymbol{d}' \wedge \\ & d_i = \infty \wedge \mathtt{j} \geq \mathtt{i} \wedge \boldsymbol{d}' = (d_0, \ldots, d_i, \infty, \ldots, \infty) \vee \\ & \qquad \mathtt{j} < \mathtt{i} \wedge \boldsymbol{d}' = (d_0, \ldots, d_{\mathtt{j}}, \infty, \ldots, \infty) \vee \\ & d_i = m \wedge \; \mathtt{j} > \mathtt{i} \wedge \; \boldsymbol{d}' = \boldsymbol{d} \vee \\ & \qquad \mathtt{j} = \mathtt{i} \wedge \; \boldsymbol{d}' = (d_0, \ldots, d_{i-1}, d_i - 1, \infty, \ldots, \infty) \vee \\ & \qquad \mathtt{j} < \mathtt{i} \wedge \; \boldsymbol{d}' = (d_0, \ldots, d_{\mathtt{j}}, \infty, \ldots, \infty) \vee \\ & \qquad m = 0 \wedge Bx)\end{aligned}$$

A node $x \in V_1$ with configuration $(x, \boldsymbol{d})$ is good for Player 0 if Player 1 has no choice but to move to a node from which Player 0 wins. The formula is defined similarly to ψ_0 only that in the second line we ensure that if $x \in B$ then Player 0 must win the m-step game from x for all m, as otherwise Player 1 could make a backtracking move and win, and further Player 0 now also wins the m-step game from x where $m = 0$.

With ψ_0 and ψ_1 defined we can present the formula $\varphi_0(x)$ true for a node x in a simple backtracking game with $k+1$ priorities if, and only if, Player 0 has a linear winning strategy from x with local backtracking.

$$\varphi_0(x) := [\mathbf{gfp}\, R_0 x \boldsymbol{d} \,.\, \mathbf{lfp}\, R_2 x \boldsymbol{d} \,. \ldots \mathbf{fp}\, R_k x \boldsymbol{d} \,.\, (\psi_0 \vee \psi_1)](x, \infty, \ldots, \infty)$$

The next step is to show that the formula indeed defines the winning region for Player 0. This is done by showing that whenever for a node x the tuple $(x, \infty, \ldots, \infty)$ satisfies φ_0 then Player 0 has a winning strategy for the game starting at x.

It is a simple observation that the formula φ_1 defining the winning positions for Player 1 analogous to φ_0 is equivalent to the dual formula of φ_0. Thus, all nodes x either satisfy φ_0 or φ_1 and therefore φ_0 defines the winning region for Player 0 and analogously φ_1 defines the winning region for Player 1. This establishes the next theorem.

Theorem 4.6. *Winning regions of simple backtracking games are definable in* IFP.

Note that the definition of simple games involves semantic conditions, i.e. the players having linear strategies. It is open whether there is a purely syntactic criterion on game graphs allowing for the same kind of results.

References

1. A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logic. *ACM Transactions on Computational Logic (TOCL),* 2003. Accepted for publication.
2. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and company, New York, 1979. ISBN 0-7167-1044-7.

3. E. Grädel. Finite model theory and descriptive complexity. In *Finite Model Theory and Its Applications.* Springer-Verlag, 2003. To appear. See http://www-mgi.informatik.rwth-aachen.de/Publications/pub/graedel/Gr-FMTbook.ps.
4. E. Grädel and S. Kreutzer. Will deflation lead to depletion? On non-monotone fixed-point inductions. In *IEEE Symp. of Logic in Computer Science (LICS),* 2003.
5. E. Grädel, W. Thomas, and T. Wilke (eds). *Automata, Logics, and Infinite Games. A Guide to Current Research.* Lecture Notes in Computer Science Nr. 2500, Springer, 2002.
6. S. Kreutzer. Expressive equivalence of least and inflationary fixed-point logic. In *17th Symp. on Logic in Computer Science (LICS)*, pages 403 – 413, 2002.
7. D. Martin. Borel determinacy, *Annals of Mathematics* 102 (1975), pp. 336–371.
8. C. Stirling. Bisimulation, model checking and other games. Notes for the Mathfit instructional meeting on games and computation. Edinburgh, 1997.
9. I. Walukiewicz. Monadic second order logic on tree-like structures. In *STACS'96,* volume 1046 of *Lecture Notes in Computer Science (LNCS)*, pages 401 – 414. Springer Verlag, 1996.

A PTAS for Embedding Hypergraph in a Cycle
(Extended Abstract)

Xiaotie Deng[1]* and Guojun Li[2]**

[1] City University of Hong Kong, Hong Kong SAR, P. R. China
[2] Institute of Software, Chinese academy of Sciences, Beijing 100080, P. R. China; and School of Mathematics and System Sciences, Shandong University, Jinan 250100, P. R. China.

Abstract. We consider the problem of embedding hyperedges of a hypergraph as paths in a cycle such that the maximum congestion–the maximum number of paths that use any single edge in a cycle–is minimized. We settle the problem with a polynomial-time approximation scheme.

Keywords: Minimum congestion embedding hypergraph in a cycle; computer application; polynomial-time approximation scheme

1 Introduction

Embedding hyperedges of a hypergraph as paths in a cycle is a challenging problem with applications to various areas such as computer networks, communication, parallel computation, electronic design automation. The objective is to minimize the maximum congestion, where the congestion of an edge in the cycle is the number of paths that use the edge, and is called *Minimum Congestion Hypergraph Embedding in a Cycle.*

The special case of graph embedding in a cycle, MCGEC, models communication on a cyclic network with a set of routing requests where each request is defined by a pair of network nodes, a source and a destination, to be connected. The optimal solution for MCGEC can be solved in polynomial time by Frank [3], Frank, Nishizeki, Saito, Suzuki and Tardos [4], applying a deep graph theoretical approach by Okamura and Seymour [12]. The weighted version has a polynomial time approximation scheme by Shrijver, Seymour and Winkler [13, 14], and Khanna [8].

For more general communication applications, such as multicast, a request is sent to more than two nodes of the network. In execution of such a communication application, we set up a virtual routing path in the network to connect the nodes in each hyperedge. We are to minimize the congestion on the edges

* The results reported in this work is fully supported by a joint research grant of NSFC of China and Hong Kong RGC (N_CityU 102/01).

** This author was supported by the funds from NSFC under fund numbers 10271065 and 60373025 (gjli@sdu.edu.cn).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 433–444, 2004.

in the network. The hypergraph model, MCHEC, deals with such general cases. An optimal solution for the MCHEC problem corresponds to the solution, to the communication application problem of the minimum congestion value.

For general hypergraphs, Ganley and Cohoon [5] proved that the MCHEC problem is NP-hard and gave a 3-approximation algorithm for the problem. They also gave an algorithm which determines if an instance of the MCHEC problem has a solution with maximum congestion l in $O((mn)^{l+1})$ time for hypergraphs with m hyperedges and n nodes [5]. The result immediately implies that the MCHEC problem can be solved in polynomial time if the maximum congestion is bounded by a constant. In general, the maximum congestion may not be bounded by a constant. There have been several approximation algorithms that are based on different approaches, by Ganley and Cohoon [5], Gonzalez [6], and Lee and Ho [9]. All their algorithms have the same approximation ratio two. Recently, Gu and Wang present an algorithm to solve the MCHEC problem with the performance ratio 1.8 by a re-embedding technique [7].

In this paper, we present a polynomial-time approximation scheme to settle the problem. The approach is quite different from all previous (approximate) solutions. The main idea is a combinatorial approach presented in Section 3 for a special case of the problem, and its non-trivial combination with the standard randomization (and its de-randomization) approach using the optimal linear relaxation solution to an integer programming formulation presented in Section 5.

In Section 2, we will introduce the formal definition of the problem and the necessary notations with a standard integer linear program formulation of the problem. Then we handle the problem with several techniques, each works for a set of different parameters. Our solution heavily relies on a combinatorial approach, first presented in Section 3 for a special case, i.e., where the number m of hyperedges is small (bounded by $m \leq C \log n$ for any constant $C > 0$), where n is the number of nodes in the cycle. Notice that the case is trivial by an exponential size enumeration if the size of hyperedges is bounded by a constant. In comparison, we note that Ganley and Cohoon's solution results in a polynomial time solution when the optimal congestion is bounded by a constant. Our solution for this case is quite non-trivial and depends on deep insight in a neighborhood structure of the optimal solution.

In Section 4, we present the standard linear relaxation method for the case where the optimal congestion is large (greater than or equal to cm, $c > 0$ is a constant and $m \geq C \log n$ is the number of hyperedges). The final solution for the general case is presented in Section 5, where we deal with the intermediate cases using a nontrivial hybrid version of the above two methods.

The combination of the combinatorial approach for problems of small size and with linear programming problem is motivated by the work of Li, Ma and Wang, originally designed for a string problem [10], with applications to various related problems [2,1] In comparison, the exact solution for the graph case has relied on a deep graph theoretical tool [12,3,4]. Our PTAS for the hypergraph case is a result of the combinatorial insight introduced in Section 3, and its novel combination with the linear programming relaxation method.

We conclude the paper with remarks and discussion in Section 6.

2 Preliminaries

A cycle C of n nodes is an undirected graph $G = (V, E_G)$ with node set $V = \{i | 1 \leq i \leq n\}$ and edge set $E_G = \{e_i | 1 \leq i \leq n\}$, here each edge e_i connects the nodes i and $i+1$ for $i = 1, 2, \ldots, n$, where and in what follows, when appropriate, the arithmetic operations involving in nodes are performed implicitly using modulo n operations (using the convention of denoting 0 by n). Without loss of generality, we consider the numbers on the nodes ordered in the clockwise direction. Let $H = (V, E_H)$ be a hypergraph with the same node set $V = \{i | 1 \leq i \leq n\}$ and with a hyperedge set $E_H = \{h_1, h_2, \ldots, h_m\}$, where each hyperedge h_j is a subset of V with two or more nodes.

For each j $(1 \leq j \leq m)$, a *connecting path* (or c-path) in C for hyperedge h_j is a minimal path P_j in C such that all nodes in h_j are in P_j. That is, the two end nodes of P_j must be in h_j. Therefore, there are exactly $|h_j|$ possible c-paths for each hyperedge h_j. Choosing one connecting path for each hyperedge of H, we have an embedding of hypergraph H in a cycle is a set of connecting paths in C. Given an embedding of a hypergraph, the congestion of each edge of C is the number of c-paths that contain the edge. For a given hypergraph and a cycle on the same node set, the MCHEC problem requires to find an embedding of the hypergraph such that the maximum congestion of any edge in the cycle is minimized.

More formally, we introduce the following notations. For each $j (1 \leq j \leq m)$, let the hyperedge $h_j = \{i_1^{(j)}, i_2^{(j)}, \cdots, i_{k_j}^{(j)}\}$, such that its nodes $\{i_1^{(j)}, i_2^{(j)}, \cdots, i_{k_j}^{(j)}\}$, are ordered in the clockwise order along the cycle C. Then h_j partitions edges on the cycle C into k_j segments: $E_l^{(j)}$, $l = 1, 2, \cdots, j$, where $E_l^{(j)}$ is the set of edges in the segment $[i_l^{(j)}, i_{l+1}^{(j)}]$. Thus,

$$E_l^{(j)} = \{e_{i_l}^{(j)}, e_{i_l+1}^{(j)}, \ldots, e_{i_{l+1}-1}^{(j)}\}.$$

Note that the arithmetic operations involving in subscripts of the indices are performed by modulo k_j (with the convention of denoting 0 by k_j).

An embedding of the hyperedge $h_j = \{i_1^{(j)}, i_2^{(j)}, \cdots, i_{k_j}^{(j)}\}$ is an $E_l^{(j)}$-embedding if the c-path, that embeds the hyperedge h_j, starts from the node $i_{l+1}^{(j)}$ in clockwise and ends at the node $i_l^{(j)}$. That is, the c-path, $E_G - E_l^{(j)}$, consisting of all edges in the cycle but just missing the edges in $E_l^{(j)}$. An embedding of the hypergraph consists of a set of connecting paths that embed the m hyperedges. For each hyperedge $h_j = \{i_1^{(j)}, i_2^{(j)}, \cdots, i_{k_j}^{(j)}\}, j = 1, 2, \ldots, m$, there are k_j different ways to embed it in the cycle C, and thus the total number of feasible solutions to the MCHEC problem is therefore $k_1 k_1 \cdots k_m$.

Let $x = (x_1, x_2, \ldots, x_m)$ be a vector of dimension m, where x_j is a subset of edges in C that forms an embedding of jth hyperedge h_j. That is, $x_j = E_G - E_{l_j}^{(j)}$, for some $l_j : 1 \leq l_j \leq k_j$, represents the fact that the c-path, that embeds h_j, excludes $E_{l_j}^{(j)}$, $1 \leq j \leq m$. Such an embedding of H determined by x is then called an x-embedding. We also call the x a feasible solution to the MCHEC

problem. Let e_i be an edge of the cycle C, we use $e_i(x)$ to denote the congestion of edge e_i for the feasible solution x-embedding. The MCHEC problem can be modeled as the following optimization problem.

$$\begin{cases} \min \ z; \\ e_i(x) \le z, \ \ i = 1, 2, \ldots, n. \end{cases} \tag{1}$$

Since the problem is known to be NP-complete, we should be interested in establishing a polynomial time approximation scheme (PTAS). That is, we want to find an algorithm A which has the following performance ratio

$$R_A(I, \epsilon) = \frac{A(I)}{OPT(I)} \le 1 + \epsilon,$$

where $A(I)$ is the cost of the solution given by A and $OPT(I)$ is the cost of an optimal solution, and has complexity polynomial on the input size if ϵ is considered a constant.

3 The Special Case with $O(\log n)$ Hyperedges

In this section, we consider a hypergraph with a small number of edges, i.e., we assume that $m \le C(\log n)$ for any fixed constant $C > 0$. Let $x = (x_1, x_2, \ldots, x_m)$ be an embedding (not necessarily an optimum) of H that we wish to approximate, here $x_j = E_G - E_{l_j}^{(j)}, 1 \le l_j \le k_j$, is an embedding of hyperedge h_j for $j = 1, 2, \ldots, m$. We examine a restricted type of embeddings (enumerable by brute force) and show that for any given embedding, there is an embedding of the restricted type that is a good approximation to it with regard to the congestion.

Let $1 \le i_1, i_2, \ldots, i_r \le n$ be r distinct indices of edges on C, where r is a constant to be determined later. Let $\Omega_{i_1,i_2,\ldots,i_r}$ denote a set of indices of hyperedges such that j is a member in this set iff $E_{l_j}^{(j)}$ contains at least one of $e_{i_1}, e_{i_2}, \ldots, e_{i_r}$, i.e.,

$$\Omega_{i_1,i_2,\ldots,i_r} = \{1 \le j \le m | E_{l_j}^{(j)} \cap \{e_{i_1}, e_{i_2}, \ldots, e_{i_r}\} \ne \emptyset\}.$$

Intuitively, we may regard an edge e in C as a representation of a hyperedge h in H if it is contained in the segment of C that is left out in the x-embedding of the hyperedge h. Then, $\Omega_{i_1,i_2,\ldots,i_r}$ is the union of hyperedges represented by $e_{i_1}, e_{i_2}, \ldots, e_{i_r}$. Let Ω be one of the maximum size for the fixed parameter r, that is, $\Omega = \Omega_{m_1,m_2,\ldots,m_r}$ such that

$$|\Omega| = \max_{1 \le i_1, i_2, \ldots, i_r \le n} \{|\Omega_{i_1,i_2,\ldots,i_r}|\}.$$

Let x' be any embedding of H such that $x'_j = x_j$ if $j \in \Omega$, i. e., the two embeddings x and x' of H have the same embedding for the hyperedges with their indices in Ω. Note that the jth component x'_j (x_j) of x' (x) stands for a segment (or c-path) in the cycle C to embed the jth hyperedge h_j. The following lemma shows that any such x' is a good approximation to x.

Lemma 1. *Let x be any (not necessarily an optimal) embedding of H and x' be any embedding such that $x'_j = x_j$ if $j \in \Omega$. Then, for each $e_i \in E_G$, we have:*

$$e_i(x') - e_i(x) \leq \frac{1}{r} e_i(x).$$

In other words, the difference of the congestions of x and x' on any edge in the cycle C ($e_i \in E_G$) is bounded by a factor $1/r$, if they have the same embedding for edge in Ω, which is defined by x and r.

Proof. Let $\Omega = \Omega_{m_1,m_2,\ldots,m_r}$ be defined as above for the fixed $x = \{x_1, x_2, \ldots, x_m\}$, where $x_j = E_G - E_{l_j}^{(j)}$ is an embedding of hyperedge h_j. For any $e_i \in E_G$, define a subset $\Omega(i)$ of indices j of hyperedges of H such that e_i is in $E_{l_j}^{(j)}$, but none of e_{m_h} is in $E_{l_j}^{(j)}, h = 1, 2, \ldots, r$. That is,

$$\Omega(i) = \{1 \leq j \leq m | e_i \in E_{l_j}^{(j)} \text{ and } e_{m_h} \notin E_{l_j}^{(j)}, h = 1, 2, \ldots, r\}.$$

Therefore,

$$\Omega(i) = \Omega_{m_1,m_2,\ldots,m_r,i} - \Omega_{m_1,m_2,\ldots,m_r} \tag{2}$$

For $1 \leq t \leq r$, define

$$\Omega(m_t, i) = \{1 \leq j \leq m | e_{m_t} \in E_{l_j}^{(j)} \text{ and } e_i, e_{m_h} \notin E_{l_j}^{(j)}, h \in \{1, 2, \ldots, r\} - \{t\}\},$$

and thus,

$$\Omega(m_t, i) = \Omega_{m_1,m_2,\ldots,m_r,i} - \Omega_{m_1,\ldots,m_{t-1}m_{t+1},\ldots,m_r,i} \tag{3}$$

By the choice of Ω, we have

$$\Omega_{m_1,m_2,\ldots,m_r} = |\Omega| \geq |\Omega_{m_1,\ldots,m_{t-1}m_{t+1},\ldots,m_r,i}| \tag{4}$$

Combining the formulas (2),(3) and (4), we get $|\Omega(m_t, i)| \geq |\Omega(i)|$.

Consider two distinct integers p and q ($1 \leq p, q \leq r$). If $j \in \Omega(m_p, i)$, then

$$e_{m_p} \in E_{l_j}^{(j)} \text{ and } e_i, e_{m_h} \notin E_{l_j}^{(j)}, h \in \{1, 2, \ldots, r\} - \{p\}.$$

In particular, $e_{m_p} \in E_{l_j}^{(j)}$ and $e_{m_q} \notin E_{l_j}^{(j)}$. Similarly, $j \in \Omega(m_q, i)$ implies $e_{m_q} \in E_{l_j}^{(j)}$ and $e_{m_p} \notin E_{l_j}^{(j)}$. Therefore, $\Omega(m_p, i) \cap \Omega(m_q, i) = \emptyset$

In addition, the embedding segment of a hyperedge j (according to the fixed embedding x) contains e_i if and only if $e_i \notin E_{l_j}^{(j)}$. Therefore, every hyperedge in $\Omega(m_t, i)$ is embedded (according to x) on a segment containing e_i.

Summing up the above statements, we have

$$e_i(x) \geq \sum_{t=1}^{r} |\Omega(m_t, i)| \geq r|\Omega(i)|. \tag{5}$$

Now consider an x' such that $x'_j = x_j$ if $j \in \Omega$. Let l'_j be the index such that $x'_j = E_G - E^{(j)}_{l'_j}$. Therefore,

$$e_i(x') - e_i(x) \le |\{1 \le j \le m | e_i \notin E^{(j)}_{l'_j} \text{ and } e_i \in E^{(j)}_{l_j}\}|.$$

Since $e_i \notin E^{(j)}_{l'_j}$ and $e_i \in E^{(j)}_{l_j}$, $x'_j \neq x_j$. By the condition that $x'_j = x_j$ if $j \in \Omega$, it follows that $j \notin \Omega$ in such case. Therefore, we conclude

$$\begin{aligned} e_i(x') - e_i(x) &\le |\{1 \le j \le m | e_i \notin E^{(j)}_{l'_j} \text{ and } e_i \in E^{(j)}_{l_j}\}| \\ &= |\{j \notin \Omega | e_i \notin E^{(j)}_{l'_j} \text{ and } e_i \in E^{(j)}_{l_j}\}| \\ &\le |\{j \notin \Omega | e_i \in E^{(j)}_{l_j}\}| \\ &= |\{1 \le j \le m | e_i \in E^{(j)}_{l_j} \text{ and } e_{m_h} \notin E^{(j)}_{l_j}, h = 1, 2, \ldots, r\}| \\ &= |\Omega(i)| \le \frac{1}{r} e_i(x). \end{aligned}$$

where the last inequality is from (5).

Note that consider $\Omega_{i_1, i_2, \ldots, i_r}$ derived from the assumed optimal solution x^*. We may enumerate through all the subscripts $m_1, m_2, \ldots, m_r$ to choose e_{m_j}, $j = 1, 2, \cdots, r$. in time $O(n^r)$ iterations.

For each hyperedge $h_j, j = 1, 2, \ldots, m$, let $E^{(j)}_{j_1}, E^{(j)}_{j_2}, \ldots, E^{(j)}_{j_{l_j}}$ be all the segments such that

$$E^{(j)}_{j_i} \cap \{e_{m_1}, e_{m_2}, \ldots, e_{m_r}\} \neq \emptyset, i = 1, 2, \ldots, l_j$$

and then $|l_j| \le r, j = 1, 2, \ldots, m$. Let

$$X = \{x = \{x_1, x_2, \ldots, x_m\} \,|\, x_j \in \{E_G - E^{(j)}_{j_i}, i = 1, 2, \ldots, l_j\}, j = 1, 2, \ldots, m\}$$

be a set of embeddings. Then it follows that

$$|X| \le l_1 l_2 \cdots l_m \le r^m \le r^{C \log n} = n^{C \log r} \quad (\text{note that } m \le C \log n).$$

By Lemma 1, we can enumerate all embeddings over X in $O(n^{C \log r})$ times to find one x' that is a good approximation of the optimum solution x^* (the one with the same component as x^* at all $j \in \Omega$ should be our desired).

Recall that for an optimal solution x^* we have in mind, we can enumerate the subscripts $m_1, m_2, \ldots, m_r$ such that $\Omega = \Omega_{m_1, m_2, \ldots, m_r}$ with respect to x^* by choosing all possible r elements from the edge set E_G. Summing up the above discussion, we know that the total number of solutions generated by our brute enumeration method can be up bounded by $O(n^{r + C \log r})$. Choosing one with minimum congestion among the enumerated solutions, we get the desired approximation. The algorithm is given in Figure 1.

Setting $r = \frac{1}{\epsilon}$ we have the following theorem.

Algorithm specialEmbedding

Input: $G = (V, E_G)$ and $H = (V, E_H)$.
Output: an x-embedding of H.
1. **for** each r-element subset $\{e_{i_1}, e_{i_2}, \ldots, e_{i_r}\}$ of the n input edges in E_G **do**
   enumerate all the possible solutions in the above method.
2. Output the best solution obtained in Step 1.

Fig. 1. Algorithm for MCHEC with small number of hyperedges.

Theorem 1. *The MCHEC problem can be solved with a PTAS when* $m \leq C \log n$ *for any constant* $C > 0$. *In particular, for any given* $\epsilon > 0$, *a solution with* $1 + \epsilon$ *factor of the optimum can be found in time* $O(n^{(C+1)/\epsilon})$.

4 The Case with Large Optimal Solution

We define a variable, $x_{j,l}$, to be one if $x_j = E_G - E_l^{(j)}$, and to be zero otherwise, where $1 \leq j \leq m$ and $1 \leq l \leq k_j$. We also introduce a set of index functions, $\chi_j(e_i, l) = 0$ if $e_i \in E_l^{(j)}$ and 1 if $e_i \notin E_l^{(j)}$. Then, (1) is equivalent to the following 0-1 optimization problem:

$$\begin{cases} \min \ z; \\ \sum_{l=1}^{k_j} x_{j,l} = 1, \ \ j = 1, 2, \ldots, m, \\ \sum_{j=1}^{m} \sum_{l=1}^{k_j} \chi_j(e_i, l) x_{j,l} \leq z, \ \ i = 1, 2, \ldots, n. \end{cases} \tag{6}$$

Here c_{opt} denotes the objective value of optimum solution of ILP (6). In this section, we only need to consider the case $m \geq C \log n$ for any constant C. In addition, we restrict ourselves to the MCHEC problem with a large value c_{opt} ($c_{opt} \geq cm$, where $c > 0$ is a constant.

We apply the randomized rounding strategy to round a fractional optimal solution $\bar{x}_{j,l}, j = 1, 2, \ldots, m; l = 1, 2, \ldots, k_j$ for (6). For each $j = 1, 2, \ldots, m$, independently, with probability $\bar{x}_{j,l}$, set $x'_{j,l} = 1$ and $x'_{j,h} = 0$ for any $h \in \{1, 2, \ldots, k_j\} - \{l\}$. Then we get a solution $x'_{j,l}$ for $j = 1, 2, \ldots, m; l = 1, 2, \ldots, k_j$ for the 0-1 problem (6), hence a solution for (1).

The following lemma will be useful here.

Lemma 2. *[11] Let* $X_1, X_2, \ldots, X_n$ *be* n *independent random 0-1 variables, where* X_i *takes* 1 *with probability* p_i, $0 < p_i < 1$. *Let* $X = \sum_{i=1}^{n} X_i$, *and* $\mu = E[X]$. *Then for any* $\delta > 0$, $\mathbf{Pr}(X > \mu + \delta n) < \exp\left(-\frac{1}{3} n \delta^2\right)$,

We then establish a key lemma:

Lemma 3. *Let* $\epsilon > 0$, $m \geq C \log n$, *and* $c_{opt} \geq cm (0 < c \leq 1)$. *Let* $x'_{j,l}$ *be a 0-1 solution of (6) after the randomized rounding procedure. Then, with probability at least* $1 - n^{1 - \frac{1}{3}\epsilon^2 c^2 C}$, *for each* $e_i \in E_G$,

$$e_i(x') \leq (1 + \epsilon) c_{opt}.$$

Proof. Note that, fixing j, $x_{j,l}$ is rounded to 1 only for one index l: $1 \le l \le k_j$. Therefore, the variable $\sum_{l=1}^{k_j} \chi_j(e_i, l)x_{j,l}$ also rounds to the value of either 1 or 0, and is independently for different j's. So $e_i(x) = \sum_{j=1}^{m} \sum_{l=1}^{k_j} \chi_j(e_i, l)x_{j,l}$ is a sum of m independent 0-1 random variables. Moreover,

$$E[e_i(x)] = \sum_{j=1}^{m} \sum_{l=1}^{k_j} \chi_j(e_i, l)E[x_{j,l}] = \sum_{j=1}^{m} \sum_{l=1}^{k_j} \chi_j(e_i, l)\bar{x}_{j,l} = \tau_i \le c_{opt}. \quad (7)$$

So, for any fixed $\delta > 0$, using Lemma 2, we have

$$\mathbf{Pr}\,(e_i(x) > \tau_i + \delta m) < \exp\left(-\frac{1}{3}\delta^2 m\right).$$

Consider all edges respectively, we have

$$\mathbf{Pr}\,(e_i(x) > \tau_i + \delta m \text{ for at least one } e_i \in E_G) < n \times \exp\left(-\frac{1}{3}\delta^2 m\right),$$

Since $m \ge C \log n$, we get $n \times \exp(-\frac{1}{3}\delta^2 m) \le n^{1-\delta^2 C/3}$. So we get a randomized algorithm to find a solution x for (6) and so for (1) with probability at least $1 - n^{1-\delta^2 C/3}$ such that for $i = 1, 2, \cdots, n$, $e_i(x) \le \tau_i + \delta m \le c_{opt} + \frac{\delta}{c} c_{opt} = (1 + \epsilon)c_{opt}$, where $\epsilon = \frac{\delta}{c}$. The lemma follows by setting $\delta = c\epsilon$.

Applying the standard derandomization method [10], we have the following result.

Theorem 2. *The MCHEC problem can be solved with a PTAS when $c_{opt} \ge cm$ and m is sufficient large (by choosing sufficiently large constant C such $m \ge C \log n$).*

5 The Ultimate PTAS

The straightforward LP relaxation technique does not work when the optimal congestion c_{opt} is small relative to m, the number of hyperedges, because the randomized rounding procedure will introduce large errors. We use the idea from a string problem [10] that applies the LP relaxation to a specified subset of variables.

Let x be the optimal solution with minimum congestion c_{opt}. For any $e_l \in E_G$, let $x^{(l)}$ be such an embedding such that $e_l \notin x_j^{(l)}$ for $j = 1, 2, \ldots, m$, i. e., c-path (embedding segment) of each hyperedge in the $x^{(l)}$-embedding misses the edge e_l. Then $e_i(x^{(l)}) \le 2c_{opt}$ for any $e_i \in E_G$. Thus, this *same edge-missing* algorithm has performance ratio 2. Now, we generalize the ratio 2 algorithm by considering k edges $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ in E_G at a time. Recall that the edges in the same segment, say in $E_l^{(j)}$, derived by h_j are called to be relative with respect

to j-th hyperedge. Let $R_{i_1,i_2,\ldots,i_k}$ be the set of indices of hyperedges such that $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ are all relative with respect to those hyperedges, i. e.,

$$R_{i_1,i_2,\ldots,i_k} = \left\{1 \le j \le m \,|\, \exists\, l_j \in \{1,2,\ldots,k_j\} \text{ such that } e_{i_1}, e_{i_2}, \ldots, e_{i_k} \in E_{l_j}^{(j)}\right\}$$

For a y-embedding, we use $y|_{R_{i_1,i_2,\ldots,i_k}}$ to denote a partial embedding of y restricted on $R_{i_1,i_2,\ldots,i_k}$. Let $U_{i_1,i_2,\ldots,i_k} = \{1,2,\ldots,m\} - R_{i_1,i_2,\ldots,i_k}$. The following observation ensures that the techniques developed in last two sections can be applied to $U_{i_1,i_2,\ldots,i_k}$.

Lemma 4. *[10]* $|U_{i_1,i_2,\ldots,i_k}| \le kc_{opt}$ *and* $|R_{i_1,i_2,\ldots,i_k}| \ge m - kc_{opt}$.

Proof. Let $j \in U_{i_1,i_2,\ldots,i_k}$. Then $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ do not belong to a same segment of j-th hyperedge, and thus there exists some e_{i_h} such that $e_{i_h} \in x_j$, i. e., e_{i_h} gets one congestion from the optimal embedding x. Since $e_{i_h}(x) \le c_{opt}$, each e_{i_h} contributes at most c_{opt} indices in $U_{i_1,i_2,\ldots,i_k}$. Therefore, $|U_{i_1,i_2,\ldots,i_k}| \le kc_{opt}$. By definition, $|R_{i_1,i_2,\ldots,i_k}| \ge m - kc_{opt}$.

Our main idea to approximate the optimal embedding x is to attack the two sets of hyperedges with their indices in $R_{i_1,i_2,\ldots,i_r}$ and $U_{i_1,i_2,\ldots,i_k}$ respectively. We first show that there exist indices $i_1, i_2, \ldots, i_r$ such that the $x^{(i_1)}|_{R_{i_1,i_2,\ldots,i_r}}$-embedding forms a good approximation to the optimal embedding x for all hyperedges $j \in R_{i_1,i_2,\ldots,i_r}$. For this purpose, we need some notations.

For any $2 \le k < r$, and $1 \le i_1, i_2, \ldots, i_k \le n$, let

$$p_{i_1,i_2,\ldots,i_k} = |\{j \in R_{i_1,i_2,\ldots,i_k} | x_j^{(i_1)} \ne x_j\}|,$$

and

$$\rho_k = \min_{1 \le i_1,i_2,\ldots,i_k \le n} \frac{p_{i_1,i_2,\ldots,i_k}}{c_{opt}}.$$

We would need the following lemma. It ensures that there exist indices $i_1, i_2, \ldots, i_r$ such that the $x^{(i_1)}|_{R_{i_1,i_2,\ldots,i_r}}$-embedding is indeed a good approximation to the optimal embedding x for all hyperedges $j \in R_{i_1,i_2,\ldots,i_r}$.

Lemma 5. *[10] For any constant* $r, 2 \le r < n$, *there are indices* $1 \le i_1, i_2, \ldots, i_r \le n$ *such that for any* $e_i \in E_G$,

$$e_i(x^{(i_1)}|_{R_{i_1,i_2,\ldots,i_r}}) - e_i(x|_{R_{i_1,i_2,\ldots,i_r}}) \le \frac{1}{r-1} c_{opt}. \tag{8}$$

Proof. Note that ρ_k decreases when k increasing. Consider the sum of $r-1$ terms,

$$(\rho_2 - \rho_3) + (\rho_3 - \rho_4) + \ldots + (\rho_r - \rho_{r+1}) = \rho_2 - \rho_{r+1} \le \rho_2 \le 1.$$

Thus, there is $k (2 \le k \le r)$ such that $\rho_k - \rho_{k+1} \le \frac{1}{r-1}$.

Let $R(i) = \{j \in R_{i_1,i_2,\ldots,i_r} | x_j^{(i_1)} \ne x_j \text{ and} x_j^{(i_1)} \ne x_j^{(i)}\}$ for all $e_i \in E_G$. Then $e_i(x^{(i_1)}|_{R_{i_1,i_2,\ldots,i_r}}) - e_i(x|_{R_{i_1,i_2,\ldots,i_r}}) \le |R(i)|$. We only need to show that $|R(i)| \le \frac{1}{r-1} c_{opt}$.

Consider the indices $1 \le i_1, i_2, \ldots, i_k \le n$ such that $p_{i_1,i_2,\ldots,i_k} = \rho_k c_{opt}$. Then, for any $k < r \le n$ and $1 \le i \le n$, we have

$$\begin{aligned}
&|R(i)| = |\{j \in R_{i_1,i_2,\ldots,i_r} | x_j^{(i_1)} \ne x_j \text{ and } x_j^{(i_1)} \ne x_j^{(i)}\}| \\
&\le |\{j \in R_{i_1,i_2,\ldots,i_k} | x_j^{(i_1)} \ne x_j \text{ and } x_j^{(i_1)} \ne x_j^{(i)}\}| \\
&= |\{j \in R_{i_1,i_2,\ldots,i_k} | x_j^{(i_1)} \ne x_j\} - \{j \in R_{i_1,i_2,\ldots,i_k} | x_j^{(i_1)} = x_j^{(i)} \text{ and } x_j^{(i_1)} \ne x_j\}| \\
&= |\{j \in R_{i_1,i_2,\ldots,i_k} | x_j^{(i_1)} \ne x_j\} - \{j \in R_{i_1,i_2,\ldots,i_k,i} | x_j^{(i_1)} \ne x_j\}| \\
&= |\{j \in R_{i_1,i_2,\ldots,i_k} | x_j^{(i_1)} \ne x_j\}| - |\{j \in R_{i_1,i_2,\ldots,i_k,i} | x_j^{(i_1)} \ne x_j\}| \\
&= p_{i_1,i_2,\ldots,i_k} - p_{i_1,i_2,\ldots,i_k,i} \\
&\le (\rho_k - \rho_{k+1}) c_{opt} \le \frac{1}{r-1} c_{opt}.
\end{aligned}$$

For the hyperedges with indices in $U_{i_1,i_2,\ldots,i_r} = \{1, 2, \ldots, m\} - R_{i_1,i_2,\ldots,i_r}$, we use ideas developed in the last two sections. Without loss of generality, we assume that $U = U_{i_1,i_2,\ldots,i_r} = \{1, 2, \ldots, |U|\}$, and $R = \{1, 2, \ldots, m\} - U$. We consider two cases:

Case 1. $|U_{i_1,i_2,\ldots,i_r}| \le C \log n$. Using the technique developed in section 3, we can find a partial embedding $x''|_U$ for the hyperedges with their indices in $U_{i_1,i_2,\ldots,i_r}$ in polynomial time such that for any $e_i \in E_G$,

$$e_i(x''|_U) - e_i(x|_U) \le \frac{1}{r} c_{opt}.$$

Together with Lemma 5, we define $x'_j = x_j^{(i_1)}$ if $j \in R_{i_1,i_2,\ldots,i_r}$ and $x'_j = x''_j$ if $j \in U_{i_1,i_2,\ldots,i_r}$, then we get

$$e_i(x') - e_i(x) \le (\frac{1}{r} + \frac{1}{r-1}) c_{opt},$$

and thus

$$e_i(x') \le (1 + \frac{1}{r} + \frac{1}{r-1}) c_{opt}.$$

Case 2. $|U_{i_1,i_2,\ldots,i_r}| \ge C \log n$. We use LP relaxation to approximate the optimal embedding x for the hyperedges with their indices in $U_{i_1,i_2,\ldots,i_r}$, since $c_{opt} \ge \frac{1}{r}|U_{i_1,i_2,\ldots,i_r}|$ (by Lemma 4) and thus the conditions for applying the method are satisfied. From Lemma 5, the following optimization problem

$$\begin{cases} \min \ z; \\ \sum_{l=1}^{k_j} x_{j,l} = 1, \ \ j = 1, 2, \ldots, |U|, \\ \sum_{j=1}^{|U|} \sum_{l=1}^{k_j} \chi_j(e_i, l) x_{j,l} \le z - e_i(x^{(i_1)}|_R), \ \ i = 1, 2, \ldots, n. \end{cases} \tag{9}$$

has a fractional solution $\bar{x}_{j,l} (1 \le j \le |U|, 1 \le l \le k_j)$ with cost $\bar{d} \le (1 + \frac{1}{r-1}) c_{opt}$.

From the proof of Lemma 3, we have:

Lemma 6. *Let $x'|_U$ be a 0-1 solution of (9) after randomized rounding. Then, for any $\delta > 0$, with high probability, for each $e_i \in E_G$,*

$$e_i(x'|_U) \leq (1 + \frac{1}{r-1})c_{opt} - e_i(x^{(i_1)}|_R) + \delta|U|.$$

Using standard derandomization procedure as in the last section, we can find an approximate solution x'' in polynomial time such that

$$e_i(x''|_U) \leq (1+\frac{1}{r-1})c_{opt} - e_i(x^{(i_1)}|_R) + 2\delta|U| \leq (1+\frac{1}{r-1}+2r\delta)c_{opt} - e_i(x^{(i_1)}|_R.$$

Let x' be a concatenation of $x^{(i_1)}|_R$ and $x''|_U$. Then x' is our desired approximation of the optimal embedding x such that

$$e_i(x') = e_i(x^{(i_1)}|_R) + e_i(x'|_U) \leq (1 + \frac{1}{r-1} + 2r\delta)c_{opt}.$$

The algorithm for the general MCHEC problem depicted in Figure 2.

Algorithm generalEmbedding

Input: $G = (V, E_G)$ and $H = (V, E_H)$.
Output: an x-embedding of H.
1. **for** each r-element subset $\{e_{i_1}, e_{i_2}, \ldots, e_{i_r}\}$ of the n input edges in E_G **do**
   (a) $R = \{1 \leq j \leq m \mid e_{i_1}, e_{i_2}, \ldots, e_{i_r}$ are in the same segment of j-th hyperedge$\}$, $U = \{1, 2, \ldots, m\} - R$.
   (b) For the hyperedges with their indices in $R_{i_1,i_2,\ldots,i_r}$, take $x^{(i_1)}|_R$ as an approximation of optimal embedding x.
   (c) For the hyperedges with their indices in $U_{i_1,i_2,\ldots,i_r}$, find a partial embedding $x'|_U$ using the techniques developed in section 5.
   (d) Get an approximation x' of x by concatenating $x^{(i_1)}|_R$ and $x'|_U$.
2. Output the best solution obtained in Step 1.

Fig. 2. Algorithm for the general MCHEC problem.

Theorem 3. *There is a PTAS for the MCHEC problem.*

6 Discussions

The MCHEC problem can solve in polynomial time when each hyperedge contains exactly two nodes [3,4]. However, the weighted MCHEC problem is NP-complete even if each hyperedge contains exactly two nodes [9]. In this work, we establish a polynomial time approximation scheme for the MCHEC problem. An

immediate open problem is whether there is a polynomial time approximation scheme for its weighted version.

Our work extends the techniques started in [10] for the string problems in bioinformatics to a completely different application area. It would be interesting to find other applications.

References

1. Xiaotie Deng, Guojun Li, Zimao Li, Bin Ma, Lusheng Wang: Genetic Design of Drugs Without Side-Effects. SIAM J. Comput. 32(4): 1073-1090 (2003)
2. Xiaotie Deng, Guojun Li, Lusheng Wang, *Center and Distinguisher for Strings with Unbounded Alphabet,* Journal of Combinatorial Optimization, 6: 383-400, 2002.
3. A. Frank, Edge-disjoint paths in planar graphs, *J. Combin. Theory Ser. B,* Vol. 38 (1985), pp. 164-178.
4. A. Frank, T. Nishizeki, N. Saito, H. Suzuki, E. Tardos, *Algorithms for routing around a rectangle,* Discrete Applied Mathematics, 40: 363-378, 1992.
5. J. L. Ganley and J. P. Cohoon. *Minimum-congestion hypergraph embedding on a cycle.* IEEE Trans. on Computers, Vol.46, No.5, 1997, pp. 600-602.
6. T. Gonzalez, *Improved approximation algorithm for embedding hyperedges in a cycle,* Information Processing Letters, 67: 267-271, 1998.
7. Q. P. Gu and Y. Wang, *Efficient algorithm for embedding hypergraph in a cycle,* Proceedings of the 10th International Conference On High Performance Computing, pp.85-94, December 2003, Hyderabad, India.
8. Sanjeev Khanna, A Polynomial Time Approximation Scheme for the SONET Ring Loading Problem. Bell Labs Tech. J. 2 (1997), pp.36-41.
9. S. L. Lee, H. J. Ho, *Algorithms and complexity for weighted hypergraph embedding in a cycle,* In proc. of the 1st International Symposium on Cyber World (CW2002), 2002.
10. Ming Li, Bin Ma, Lusheng Wang: On the closest string and substring problems. JACM 49(2): 157-171 (2002)
11. R. Motwani and P. Raghavan, Randomized algorithms, Cambridge Univ. Press.
12. Haruko Okamura, and P.D. Seymour, Multicommodity Flows in Planar Graph. Journal of Combinatorial Theory, Series B, Vol. 31, pp.75–81, 1981.
13. A. Schrijver, P. Seymour, P. Winkler, The Ring Loading Problem. SIAM Discrete Mathematics, Vol 11, No. 1, pp.1-14, 1998.
14. A. Schrijver, P. Seymour, P. Winkler, The Ring Loading Problem. SIAM Review, Vol 41, No. 4, pp.777-791, 1999.

Towards an Algebraic Theory of Typed Mobile Processes*

Yuxin Deng[1] and Davide Sangiorgi[2]

[1] INRIA and Université Paris 7, France
[2] Università di Bologna, Italy

1 Introduction

The π-calculus is the paradigmatic calculus for process mobility. Its theory has been studied in depth [8,12]. Relevant parts of it are the algebraic theory and the type systems. Most of the algebraic theory has been developed on the untyped calculus; the results include proof systems or axiomatisations that are sound and complete on finite processes for the main behavioral equivalences: late and early bisimilarity, late and early congruence [9,6,7], open bisimilarity [11], testing equivalence [1]. Much of the research on types has focused on their behavioral effects. For instance, modifications of the standard behavioral equivalences have been proposed so as to take types into account [10,12].

In this paper, we study the impact of types on the algebraic theory of the π-calculus. Precisely, we study axiomatisations of the typed π-calculus. Although algebraic laws for typed calculi for mobility have been considered in the literature [12], we are not aware of any axiomatisation or proof system.

The type system that we consider has *capability types* (sometimes called I/O types) [10,5]. These types allow us to distinguish, for instance, the capability of using a channel in input from that of using the channel in output. A capability type shows the capability of a channel and, recursively, of the channels carried by that channel. For instance, a type $a : \mathtt{iob}T$ (for an appropriate type expression T) says that channel a can be used only in input; moreover, any channel received at a may only be used in output — to send channels which can be used both in input and in output. Thus, process $a(x).\bar{x}b.b(y).\bar{b}y.\mathbf{0}$ (sometimes the trailing $\mathbf{0}$ is omitted) is well-typed under the type assignment $a : \mathtt{iob}T, b : \mathtt{b}T$. We recall that $\bar{a}b.P$ is the output at a of channel b with continuation P, and that $a(x).P$ is an input at a with x a placeholder for channels received in the input whose continuation is P.

On calculi for mobility, capability types have emerged as one of the most useful forms of types, and one whose behavioral effects are most prominent. Capabilities are useful for protecting resources; for instance, in a client-server model, they can be used for preventing clients from using the access channel to the server in input and stealing messages to the server; similarly they can be used in distributed programming for expressing security constraints [5]. Capabilities give rise to *subtyping:* the output capability is contravariant, whereas the input

* Work supported by EU project PROFUNDIS.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 445–456, 2004.

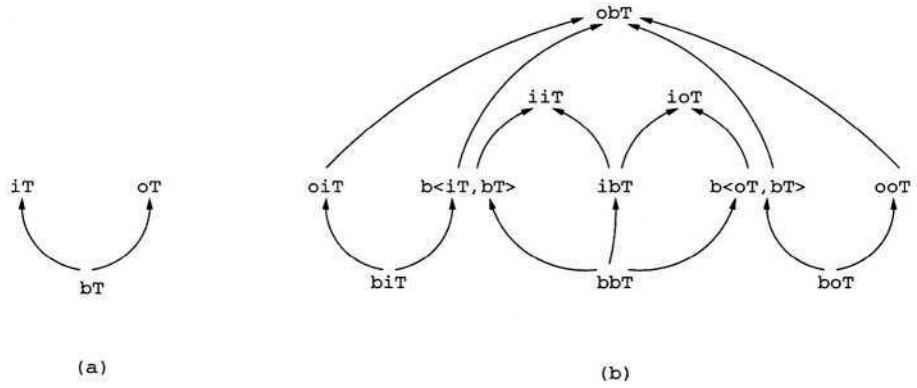


Fig. 1. An example of subtyping relation, with T = unit

capability is covariant. As an example, we show a subtyping relation in Figure 1, where an arrow indicates the subtyping relation between two related types. The depth of nesting of capabilities is 1 for all types in diagram (a), and 2 for all types in diagram (b). (The formal definitions of types and subtyping relation will be given in Section 2.) Subtyping is useful when the π-calculus is used for object-oriented programming, or for giving semantics to object-oriented languages.

To see why the addition of capability types has semantic consequences, consider

$$P \stackrel{\text{def}}{=} \nu c\, \bar{b}c.a(y).(\bar{y} \mid c) \qquad Q \stackrel{\text{def}}{=} \nu c\, \bar{b}c.a(y).(\bar{y}.c + c.\bar{y})$$

These processes are not behaviorally equivalent in the untyped π-calculus. For instance, if the channel received at a is c, then P can terminate after 2 interactions with the external observer. By contrast, Q always terminates after 4 interactions with the observer. However, if we require that only the input capability of channels may be communicated at b, then P and Q are indistinguishable in any (well-typed) context. For instance, since the observer only receives the input capability on c, it cannot resend c along a: channels sent at a require at least the output capability (cf: the occurrence of $\bar{y}$). Therefore, in the typed setting, processes are compared w.r.t. an observer with certain capabilities (i.e., types on channels). Denoting with Δ these capabilities, then typed bisimilarity between P and Q is written $P \sim_{\Delta} Q$.

In the untyped π-calculus, labelled transition systems are defined on processes; the transition $P \xrightarrow{\alpha} P'$ means that P can perform action α and then become P'. In the typed π-calculus, the information about the observer capabilities is relevant because the observer can only test processes on interactions for which the observer has all needed capabilities. Hence typed labelled transition systems are defined on configurations, and a configuration $\Delta \sharp P$ is composed of a process P and the observer capabilities Δ (we sometimes call Δ the external environment). A transition $\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ now means that P can evolve into

P' after performing an action α allowed by the environment Δ, which in turn evolves into Δ'.

Capability types have been introduced in [10]. A number of variants and extensions have then been proposed. We follow Hennessy and Riely's system [5], in which, in contrast with the system in [10]: (i) there are partial meet and join operations on types; (ii) the typing rule for the *matching* construct (the construct used for testing equality between channels) is very liberal, in that it can be applied to channels of arbitrary types (in [10] only channels that possess both the input and the output capability can be compared). While (i) only simplifies certain technical details, (ii) seems essential. Indeed, the importance of matching for the algebraic theory of the π-calculus is well-known (it is the main reason for the existence of matching in the untyped calculus).

Typed bisimilarity and the use of configurations for defining typed bisimilarity have been introduced in [2]. We follow a variant of them put forward by Hennessy and Rathke [4], because it uses the type system of [5].

The main results in this paper are an axiomatisation and a proof system for typed bisimilarity ($\sim$). The axiomatisation is for all finite processes. The proof system has a simple correctness proof but only works on the closed terms. The bisimilarity $\sim$ is a variant of that in [4]. For the typed bisimilarity in [4] we provide a proof system for the closed terms, and an indirect axiomatisation of all terms that exploits the system of $\sim$. We have not been able to give a direct axiomatisation: the main difficulties are discussed in Section 5. All results are given for both the late and the early versions of the bisimilarities.

The axiomatisation and the proof systems are obtained by modifying some of the rules of the systems for the untyped π-calculus, and by adding a few new laws. The proofs of soundness and completeness, although follow the general schema of the proofs of the untyped calculus, have quite different details. An example of this is the treatment of fresh channels in input actions and the closure under injective substitutions, that we comment on below.

In the untyped π-calculus, the following holds:

$$\textit{If } P \sim Q \textit{ and } \sigma \textit{ is injective on } fn(P,Q), \textit{ then } P\sigma \sim Q\sigma.$$

Hence it is sufficient to consider all free channels in P, Q and *one* fresh channel when comparing the input actions of P and Q in the bisimulation game. This result is crucial in the algebraic theory of untyped calculi. For instance, in the proof system for (late) bisimilarity the inference rule for input is:

If $P\{b/x\} = Q\{b/x\}$ *for all* $b \in fn(P,Q,c)$, *where* c *is a fresh channel, then* $a(x).P = a(x).Q$.

For typed bisimilarity the situation is different. Take the processes

$$P \stackrel{\text{def}}{=} a(x : \mathtt{ob}T).\bar{x}c.\bar{c} \qquad Q \stackrel{\text{def}}{=} a(x : \mathtt{ob}T).\bar{x}c$$

and compare them w.r.t. an observer Δ. Consider what happens when the variable x is replaced by a fresh channel b, whose type in Δ is S. By the constraint imposed by types, S must be a subtype of the type $\mathtt{ob}T$ for x (see Figure 1 (b)).

Now, different choices for S will give different results. For instance, if S is $\circ\mathtt{b}T$ itself, then the observer has no input capability on b, thus can not communicate with P and Q at b. That is, from the observer's point of view the output $\bar{b}c$ is not observable and the two derivative processes are equivalent. Similarly if S is $\mathtt{b}\circ T$ then the output $\bar{c}$ is not observable. However, if S is $\mathtt{bb}T$ then $\bar{b}c.\bar{c}$ is not equivalent to $\bar{b}c$, since all outputs become observable. This example illustrates the essential difficulties in formulating proof systems for typed bisimilarities:

1. Subtyping appears in substitutions and changes the original type of a variable into one of its subtypes.
2. The choice of this subtype is relevant for behavioral equivalence.
3. Different subtypes may be incompatible (have no common subtype) with one another (for instance, $\mathtt{b}\circ T$ and $\mathtt{bb}T$ in the example above; they axe both subtypes of $\circ\mathtt{b}T$).

A consequence of (2) and (3), for instance, is that there is not a "best subtype", that is a single type with the property that equivalence under this type implies equivalence under any other types.

Another example of the consequences brought by types in the algebraic theory is the congruence rule for prefixes: we have to distinguish the cases in which the subject of the prefix is a channel from the case in which the subject is a variable. This is a rather subtle and technical difference, that is discussed in Section 3.

2 The Typed π-Calculus

In this section we review the π-calculus, capability types, and typed bisimilarity. We assume an infinite set of channels, ranged over by $a, b, \ldots$, and an infinite set of variables, ranged over by $x, y, \ldots$. Channels and variables are the *names,* ranged over by $u, v, \ldots$. Below is the syntax of finite π-calculus processes.

$$P, Q ::= \mathbf{0} \mid \tau.P \mid u(x:T).P \mid \bar{u}v.P \mid P+Q \mid P \mid Q \mid (\nu a:T)P \mid \varphi PQ$$
$$\varphi ::= [u=v] \mid \neg\varphi \mid \varphi \vee \psi$$

Here φPQ is an if-then-else construct on the boolean condition φ. We omit the else branch Q when it is $\mathbf{0}$. Binding names (in input and restriction) are annotated with their types. We write $fn(P)$ and $fv(P)$ for the set of free names and the set of free variables, respectively, in P. When φ has no variables, $[\![\varphi]\!]$ denotes the boolean value of φ.

We recall the capability types, as from [4,5]. The subtyping relation <: and the typing rules for processes are displayed in Table 1. We write $T :: \mathtt{TYPE}$ to mean that T is a legal type. There are three forms of types for channel names: $\mathtt{i}T$, $\circ S$ and $\mathtt{b}\langle T, S\rangle$. They give names the ability to receive values of type T, send values of type S, or to do both. We often abbreviate $\mathtt{b}\langle T, T\rangle$ to $\mathtt{b}T$. We refer to [5] for the definition of the two partial operators meet ($\sqcap$) and join ($\sqcup$). Intuitively, the meet (resp. join) of two types is the union (resp. intersection) of their capabilities.

Table 1. Types and typing rules

Types:

$$\frac{}{\top, \mathtt{unit} :: \mathtt{TYPE}} \qquad \frac{T :: \mathtt{TYPE} \quad T \neq \top}{\mathtt{i}T, \mathtt{o}T :: \mathtt{TYPE}} \qquad \frac{T, S :: \mathtt{TYPE} \quad S <: T \quad T \neq \top}{\mathtt{b}\langle T, S\rangle :: \mathtt{TYPE}}$$

Subtyping:

$$\frac{}{T <: T} \qquad \frac{T <: T'}{\mathtt{i}T <: \mathtt{i}T'} \qquad \frac{T <: T'}{\mathtt{o}T' <: \mathtt{o}T}$$

$$\mathtt{b}\langle T, S\rangle <: \mathtt{i}T' \qquad \mathtt{b}\langle S, T'\rangle <: \mathtt{o}T$$

$$\frac{}{T <: \top} \qquad \frac{T <: T' \quad S <: S'}{\mathtt{b}\langle T, S'\rangle <: \mathtt{b}\langle T', S\rangle}$$

Typing rules:

$$\frac{\Gamma(u) <: T}{\Gamma \vdash u : T} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P + Q} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash \varphi\, P\, Q}$$

$$\frac{}{\Gamma \vdash \mathbf{0}} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash v : T \quad \Gamma \vdash u : \mathtt{o}T}{\Gamma \vdash \bar{u}v.P}$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash \tau.P} \qquad \frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\nu a : T)P} \qquad \frac{\Gamma, x : T \vdash P \quad \Gamma \vdash u : \mathtt{i}T}{\Gamma \vdash u(x : T).P}$$

We use Δ and Γ for type environments. A type environment Δ is a partial function from channels and variables to types; we write $\Delta_{\mathtt{c}}$ and $\Delta_{\mathtt{v}}$ for the channel and variable parts of Δ, respectively. A type environment is undefined on infinitely many channels and variables (to make sure it can always be extended). We often view, and talk about, $\Delta_{\mathtt{c}}$ as a set of assignments of the form $a : T$. Similarly for $\Delta_{\mathtt{v}}$. We write $dom(\Delta)$ for the channels and variables on which Δ is defined ($dom(\Delta)$ can be infinite). Using the partial meet operation, we can extend a type environment Δ to $\Delta \sqcap u : T$, which is just $\Delta, u : T$ if $u \notin dom(\Delta)$, otherwise it differs from Δ at name u because the capability of this name is extended to be $\Delta(u) \sqcap T$ (if $\Delta(u) \sqcap T$ is undefined, then so is $\Delta \sqcap u : T$). When $dom(\Delta) \cap dom(\Delta') = \emptyset$, we use Δ, Δ' to represent the union of Δ and Δ'. Subtyping is extended to type environments, but only considering the types of channels. So $\Gamma <: \Delta$ means that $\Gamma_{\mathtt{v}} = \Delta_{\mathtt{v}}$, $dom(\Delta_{\mathtt{c}}) \subseteq dom(\Gamma_{\mathtt{c}})$ and $\Gamma_{\mathtt{c}}(a) <: \Delta_{\mathtt{c}}(a)$ for all $a \in dom(\Delta_{\mathtt{c}})$. The intuition is that channels are capabilities while variables are obligations of the environment. The environment is obligated to fill in the variables at the specified types. Once the obligations are determined, they cannot be strengthened or weakened. So variables are invariant. If $\Delta(u)$ is defined and takes the form $\mathtt{i}T$ or $\mathtt{b}\langle T, S\rangle$, then the predicate $\Delta(u)\downarrow_{\mathtt{i}}$ holds and we write $\Delta(u)_{\mathtt{i}}$ for T, otherwise we write $\Delta(u)\not\downarrow_{\mathtt{i}}$, indicating that Δ has no input capability on u. Similarly for $\Delta(u)_{\mathtt{o}}$ and $\Delta(u)\downarrow_{\mathtt{o}}$ (output capability). The typing rules follow [10,5,4].

Table 2. Typed transition system

$$\textsf{Red}\ \frac{P \xrightarrow{\tau} P'}{\Delta \,\sharp\, P \xrightarrow{\tau} \Delta \,\sharp\, P'} \qquad \textsf{Out}\ \frac{\Delta(a)\downarrow_i}{\Delta \,\sharp\, \bar{a}b.P \xrightarrow{\bar{a}b} \Delta \sqcap b : \Delta(a)_i \,\sharp\, P}$$

$$\textsf{In}\ \frac{\Delta(a)\downarrow_o}{\Delta \,\sharp\, a(x:T).P \xrightarrow{a(x:T)} \Delta, x:T \,\sharp\, P} \qquad \textsf{Open}\ \frac{\Delta \,\sharp\, P \xrightarrow{\bar{a}b} \Delta' \,\sharp\, P' \quad a \neq b}{\Delta \,\sharp\, (\nu b:T)P \xrightarrow{\bar{a}(b:T)} \Delta' \,\sharp\, P'}$$

$$\textsf{Res}\ \frac{\Delta \,\sharp\, P \xrightarrow{\alpha} \Delta' \,\sharp\, P' \quad a \notin n(\alpha)}{\Delta \,\sharp\, (\nu a:T)P \xrightarrow{\alpha} \Delta' \,\sharp\, (\nu a:T)P'} \qquad \textsf{Sum}\ \frac{\Delta \,\sharp\, P \xrightarrow{\alpha} \Delta' \,\sharp\, P'}{\Delta \,\sharp\, P+Q \xrightarrow{\alpha} \Delta' \,\sharp\, P'}$$

$$\textsf{True}\ \frac{[\![\varphi]\!] = \mathit{True} \quad \Delta \,\sharp\, P \xrightarrow{\alpha} \Delta' \,\sharp\, P'}{\Delta \,\sharp\, \varphi PQ \xrightarrow{\alpha} \Delta' \,\sharp\, P'} \qquad \textsf{False}\ \frac{[\![\varphi]\!] = \mathit{False} \quad \Delta \,\sharp\, Q \xrightarrow{\alpha} \Delta' \,\sharp\, Q'}{\Delta \,\sharp\, \varphi PQ \xrightarrow{\alpha} \Delta' \,\sharp\, Q'}$$

$$\textsf{Par}\ \frac{\Delta \,\sharp\, P \xrightarrow{\alpha} \Delta' \,\sharp\, P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{\Delta \,\sharp\, P \mid Q \xrightarrow{\alpha} \Delta' \,\sharp\, P' \mid Q}$$

Definition 1. *A configuration is a pair $\Delta\sharp P$ which respects some type environment Γ, i.e., $\Gamma \lessdot \Delta$ and $\Gamma \vdash P$.*

The transition system for configurations is in Table 2. Bound names, names and the subject of a prefix α, written $bn(\alpha)$, $n(\alpha)$ and $subj(\alpha)$ respectively, are defined in the usual way. We identify terms up to alpha conversion and assume $bn(P) \cap dom(\Delta) = \emptyset$ for any configuration $\Delta\sharp P$. In the premise of rule Red, $P \xrightarrow{\tau} P'$ stands for the normal reduction relation of the untyped π-calculus. In rule Out, the process sends channel b to the environment, so the latter should be dynamically extended with the capability on b thus received. For this, we use the meet operator, and exploit the following property on types:

$$R \lessdot T \text{ and } R \lessdot S \text{ imply } T \sqcap S \text{ defined and } R \lessdot T \sqcap S$$

for any type T, S and R. (This property does not hold for the capability types as in [10].)

A process is *closed* if it does not have free variables; similarly a type environment is closed if it is only defined on channels. Otherwise, processes and type environments are *open*. We first define $\sim_\Delta$ on the closed terms, then on the open terms. We write $|\alpha|$ for the action α where its type annotations have been stripped off.

Definition 2. *A family of symmetric binary relations over closed terms, indexed by type environments, and written $\{\mathcal{R}_\Delta\}_\Delta$, is a* typed bisimulation *whenever $P\ \mathcal{R}_\Delta\ Q$ implies that, for two configurations $\Delta\sharp P$ and $\Delta\sharp Q$,*

1. *if $\Delta\sharp P \xrightarrow{\alpha} \Delta'\sharp P'$ and α is not an input action, then for some Q', $\Delta\sharp Q \xrightarrow{\beta} \Delta'\sharp Q'$, $|\alpha|=|\beta|$ and $P'\ \mathcal{R}_{\Delta'}\ Q'$.*
2. *if $\Delta\sharp P \xrightarrow{a(x:T)} \Delta'\sharp P'$, then for some Q', $\Delta\sharp Q \xrightarrow{a(x:S)} \Delta''\sharp Q'$ and for all b with $\Delta_c \vdash b : \Delta(a)_o$ it holds that $P'\{b/x\}\ \mathcal{R}_\Delta\ Q'\{b/x\}$.*

Two processes P and Q are typed Δ-bisimilar, written $P \sim_\Delta Q$, if there exists a typed bisimulation $\{\mathcal{R}_\Delta\}_\Delta$ such that $P\ \mathcal{R}_\Delta\ Q$.

Definition 3. *Two processes P and Q are bisimilar under the environment Δ, written $P \sim_\Delta Q$, if $fv(P,Q) \subseteq dom(\Delta_v)$ and, for all $\widetilde{b}$ with $\Delta_c \vdash \widetilde{b} : \widetilde{T}$, it holds that $P\{\widetilde{b}/\widetilde{x}\} \sim_\Delta Q\{\widetilde{b}/\widetilde{x}\}$.*

Since all processes are finite, and we do not use recursive types, in $P \sim_\Delta Q$, the environment Δ can always be taken to be *finite* (i.e., defined only on a finite number of channels and variables): it is sufficient that Δ has enough names fresh w.r.t. P and Q, for all relevant types. This can be proved with a construction similar to that in Lemma 2. In the remainder of the paper all type environments are assumed to be finite.

3 Axioms for Typed Bisimilarity

The axiom system $\mathcal{A}$ for typed bisimilarity is in Table 3. Whenever we write $P =_\Delta Q$ it is intended that both $\Delta \sharp P$ and $\Delta \sharp Q$ are configurations. The rules are divided into seven groups, namely those for: substitutivity, sums, looking up the type environment, conditions, restrictions, the expansion law and alpha-conversion. The rules that are new or different w.r.t. those of the untyped π-calculus are marked with an asterisk. **Tin*** shows that an input prefix is not observable if the observer has no output capability on the subject of the input. **Tout*** is the symmetric rule, for output. **Twea*** gives us weakening for type environments. **Tvar*** shows that a variable can only be instantiated with channels that in the type environment have types compatible with that of the variable. **Tpre*** is used to replace names underneath a match. (In the untyped setting, the rule has no side condition. Here we need one to ensure well-typedness of the process resulting from the substitution, since the names in the match can have arbitrary — and possibly unrelated — types. Similarly for the conditions on types in the expansion law **E.**) In **Ires*,** possibly different types T_1, T_2 are used for the processes in the conclusion.

Iin* and **Iout*** are the rules for substitutivity for input and output prefixes. In **Iin***, well-definedness of the configurations $\Delta \sharp a(x:T_1).P$ and $\Delta \sharp a(x:T_2).Q$ implies the condition: $\Delta(a)_o \lessdot T_i$ for $i = 1, 2$. (Similarly for rules **Iv1, Iin** and **Iin′** below.) In **Iout*,** the observer knowledge of the type of b may increase when the processes emit b themselves (for the type under which b is emitted is composed with the possible type of b in Δ). Both in **Iin*** and in **Iout*,** the free names of the input and output prefixes are channels rather than variables. Below we discuss: (i) the unsoundness of the rules in which (some or all) the channels are replaced by variables; (ii) other rules, that are valid for variables; (iii) why these other rules are not needed in the axiom system.

To see that **Iin*** is unsound when the subject of the prefix is a variable, take $\Delta_c \stackrel{\text{def}}{=} a : \mathtt{bo}T, b : \mathtt{o}T$ and $\Delta \stackrel{\text{def}}{=} \Delta_c, x : \mathtt{b}\langle \mathtt{o}T, \mathtt{b}T \rangle$. Then we have

$$[y = b]\tau \sim_{\Delta, y:\Delta(x)_o} \mathbf{0}$$

Table 3. The axiom system $\mathcal{A}$

Iin*	If $P =_{\Delta,x:\Delta(a)_o} Q$ then $a(x:T_1).P =_\Delta a(x:T_2).Q$
Iout*	If $P =_{\Delta\sqcap b:\Delta(a)_i} Q$ then $\bar{a}b.P =_\Delta \bar{a}b.Q$
Itau	If $P =_\Delta Q$ then $\tau.P =_\Delta \tau.Q$
Isum	If $P =_\Delta Q$ then $P+R =_\Delta Q+R$
Ires*	If $P =_\Delta Q$ then $(\nu a:T_1)P =_\Delta (\nu a:T_2)Q$
Icon	If $P =_\Delta Q$ then $\varphi P =_\Delta \varphi Q$
Ipar*	If $P =_\Delta Q$ and $\Delta, \Delta' \vdash R$ for some Δ' then $P \mid R =_\Delta Q \mid R$
S1	$P + \mathbf{0} =_\Delta P$
S2	$P + P =_\Delta P$
S3	$P + Q =_\Delta Q + P$
S4	$P + (Q+R) =_\Delta (P+Q) + R$
Tin*	If $\Delta(a)\not\downarrow_o$ then $a(x:T).P =_\Delta \mathbf{0}$
Tout*	If $\Delta(a)\not\downarrow_i$ then $\bar{a}u.P =_\Delta \mathbf{0}$
Twea*	If $P =_\Delta Q$ and $\Delta \lessdot \Delta'$ then $P =_{\Delta'} Q$
Tvar*	$[x \neq a_1]\cdots[x \neq a_m]P =_\Delta \mathbf{0}$ if $\{b \in dom(\Delta_c) \mid \Delta(b) \lessdot \Delta(x)\} \subseteq \{a_1, \cdots, a_m\}$
Tpre*	$[x=a]\alpha.P =_\Delta [x=a](\alpha\{a/x\}).P$ if $\Delta(a) \lessdot \Delta(x)$
C1	$\varphi\ P =_\Delta \psi\ P$ if $\varphi \Longleftrightarrow \psi$
C2	$[a=b]P =_\Delta [a=b]Q$ if $a \neq b$
C3	$\varphi\ P\ P =_\Delta P$
C4	$\varphi\ P\ Q =_\Delta \neg\varphi\ Q\ P$
C5	$\varphi(\psi P) =_\Delta [\varphi \wedge \psi]P$
C6	$\varphi\ (P_1+P_2)\ (Q_1+Q_2) =_\Delta \varphi\ P_1\ Q_1 + \varphi\ P_2\ Q_2$
C7	$\varphi\ (\alpha.P) =_\Delta \varphi\ (\alpha.\varphi P)$ if $bn(\alpha) \cap n(\varphi) = \emptyset$
R1	$(\nu a:T)(\nu b:S)P =_\Delta (\nu b:S)(\nu a:T)P$
R2	$(\nu a:T)(P+Q) =_\Delta (\nu a:T)P + (\nu a:T)Q$
R3	$(\nu a:T)\alpha.P =_\Delta \alpha.(\nu a:T)P$ if $a \notin n(\alpha)$
R4	$(\nu a:T)\alpha.P =_\Delta \mathbf{0}$ if $subj(\alpha) = a$
R5	$(\nu a:T)[a=u]P =_\Delta \mathbf{0}$ if $a \neq u$
R6	$(\nu a:T)[u=v]P =_\Delta [u=v](\nu a:T)P$ if $a \neq u, v$

E Assume $P \equiv \Sigma_i \varphi_i \alpha_i.P_i$ and $Q \equiv \Sigma_j \psi_j \beta_j.Q_j$ where no α_i (resp. β_j) binds a name free in Q (resp. P). Let $\Gamma \vdash P \mid Q$. Then infer:

$$P \mid Q =_\Delta \textstyle\sum_i \varphi_i\alpha_i.(P_i \mid Q) + \sum_j \psi_j\beta_j.(P \mid Q_j) + \sum_{\alpha_i\ opp\ \beta_j} [\varphi_i \wedge \psi_j \wedge (u_i = v_j)]\tau.R_{ij}$$

where $\alpha_i\ opp\ \beta_j, u_i, v_j$ and R_{ij} are defined as follows:
1. α_i is $\bar{u}_i w$, β_j is $v_j(x:T)$ and $\Gamma(w) \lessdot T$; then R_{ij} is $P_i \mid Q_j\{w/x\}$;
2. α_i is $\bar{u}_i(w:S)$, β_j is $v_j(x:T)$ and $S \lessdot T$; then R_{ij} is $(\nu w:S)(P_i \mid Q_j\{w/x\})$;
3. the converse of (1) or (2).

A $P =_\Delta Q$ if P alpha-equivalent to Q

because $\Delta(x)_o = \mathsf{b}T$ and no c in Δ satisfies the condition $\Delta_c \vdash c : \mathsf{b}T$ and can therefore instantiate y. However,

$$x(y : \mathsf{o}T).[y = b]\tau \not\sim_\Delta x(y : \mathsf{o}T).\mathbf{0}.$$

To see this, let us look at the possible closing substitutions. In $dom(\Delta_c)$, a is the only channel satisfying $\Delta_c \vdash a : \Delta(x)$, and so the only substitution we need to consider is $\{a/x\}$. After applying this substitution, the resulting closed terms are not bisimilar:

$$a(y : \mathsf{o}T).[y = b]\tau \not\sim_\Delta a(y : \mathsf{o}T).\mathbf{0}$$

This holds because the observer can send b along a and, after the communication, y is instantiated to be b, thus validating the condition $y = b$ and liberating the prefix τ. When the subject of the prefix is a variable, the following rule is needed in place of **Iin***:

Iv1 If $P =_{\Delta, y:\Delta(x)_i} Q$ then $x(y : T_1).P =_\Delta x(y : T_2).Q$

In rule **Iout***, both the subject and object of the output prefix are channels. The rule is also valid when the object is a variable. However, it is not valid if the subject is a variable. In that case, we need the following rule:

Iv2 If $P =_{\Delta \sqcap v:\Delta(x)_o} Q$ then $\bar{x}v.P =_\Delta \bar{x}v.Q$

By using **Tpre***, **Tvar*** and some other rules, we can show that **Iv1** and **Iv2** are derivable. That is, rules **Iin*** and **Iout*** are sufficient in the axiom system.

Theorem 1 (Soundness and Completeness of $\mathcal{A}$). $\mathcal{A} \vdash P =_\Delta Q$ *iff* $P \sim_\Delta Q$.

The schema of the completeness proof is similar to that for the untyped π-calculus [9]. The details, however, are quite different. See [3] for the subtleties of the proof and more comparisons.

4 A Proof System for the Closed Terms

The system of Section 3 can be simplified if we limit ourselves to proving equalities on *closed* terms. With one caveat: the substitutivity rule for input is replaced by an inference rule, where (possibly several) instantiations of the bound variable of the input are considered. We call $\mathcal{P}$ the system of rules; it is presented in Table 4.

Note that rules **Icon, Tvar*, Tpre*, R5-6** are not needed, and that the set of rules **C1-7** for conditions is cut down to just **Ca-b.** In the previous system $\mathcal{A}$, axiom **R** was redundant in view of **C3** and **R5.**

Theorem 2 (Soundness and completeness of $\mathcal{P}$). $\mathcal{P} \vdash P =_\Delta Q$ *iff* $P \sim_\Delta Q$, *where P and Q are closed.*

Also the proofs of soundness and completeness are simpler, because all terms are closed and so conditions are removed as soon as possible, by means of **Ca-b.** Compared with proof systems for untyped π-calculus [9], **Tin*** and **Tout*** are the main differences.

Table 4. The proof system $\mathcal{P}$ for the closed terms

Rules **Tin*, Tout*, Iout*, Itau, Isum, Ires*, Ipar*, Twea*, S1-4, R1-4, E, A** as in Table 1, plus the following ones:

Iin If $P\{b/x\} =_\Delta Q\{b/x\}$ for all b s.t. $\Delta_c \vdash b : \Delta(a)_o$ then $a(x:T_1).P =_\Delta a(x:T_2).Q$.

Ca $\varphi\ P\ Q =_\Delta P$ if $[\![\varphi]\!] = True$

Cb $\varphi\ P\ Q =_\Delta Q$ if $[\![\varphi]\!] = False$

R $(\nu a:T)\mathbf{0} =_\Delta \mathbf{0}$

5 Other Equivalences

In the input clause of $\sim$ (Definition 2), the type environment Δ is not extended. By contrast, extensions are allowed in the bisimilarity used in [4]. We denote with $\asymp_\Delta$ the variant of $\sim_\Delta$ which allows extension; its definition is obtained from that of $\sim_\Delta$ by using the following input clause:

– if $\Delta \sharp P \xrightarrow{a(x:T)} \Delta' \sharp P'$, then for some Q', $\Delta \sharp Q \xrightarrow{a(x:S)} \Delta'' \sharp Q'$ and $\Delta, \Delta' \vdash b : \Delta(a)_o$ implies $P'\{b/x\}\ \mathcal{R}_{\Delta,\Delta'}\ Q'\{b/x\}$, for any channel b and closed type environment Δ' with $dom(\Delta') \cap (fn(P,Q) \cup dom(\Delta)) = \emptyset$.

Similarly, Δ can be extended in the definition on open terms.

In $\asymp_\Delta$, the environment collects the knowledge of the observer *relative* to the tested processes, in the sense that the environment only tells us what the observer knows of the free channels of the processes. In contrast, in $\sim_\Delta$, the environment collects the *absolute* knowledge of the observer, including information on channels that at present do not appear in the tested processes, but that might appear later — if the observer decides to send them to the processes. The main advantage of $\asymp_\Delta$ is that the environment is smaller. On the other hand, $\sim_\Delta$ allows us to express more refined interrogations on the equivalence of processes, for it gives us more flexibility in setting the observer knowledge. Indeed, while $\asymp$-equivalences can be expressed using $\sim$ (Lemma 1), the converse is false. For instance, the processes

$$P \stackrel{\text{def}}{=} a(x : \mathtt{bo}T).[x = y]\tau \qquad Q \stackrel{\text{def}}{=} a(x : \mathtt{bo}T).\mathbf{0}$$

are in the relation $\sim_\Delta$, for $\Delta \stackrel{\text{def}}{=} a : \mathtt{obo}T, b : \mathtt{bb}T, y : \mathtt{ob}T$. However, they are not in a relation $\asymp_\Gamma$, for any Γ: the observer can always create a new channel of type $\mathtt{bo}T$, and use it to instantiate both x and y, thus validating the condition $[x = y]$.

Lemma 1. *If $P \asymp_\Delta Q$ then $P \sim_\Delta Q$.*

We can derive a proof system for $\asymp$ with a simple modification of that for $\sim$ in Section 4. Let $\mathcal{P}'$ be the system obtained from $\mathcal{P}$ by replacing rule **Iin** with **Iin′**:

Iin′ If $-\ P\{b/x\} =_\Delta Q\{b/x\}$ for all b with $\Delta(b) <: \Delta(a)_o$, and
$-$ given $c \notin fn(P,Q) \cup dom(\Delta)$,
$P\{c/x\} =_{\Delta,c:T} Q\{c/x\}$ for all $T <: \Delta(a)_o$,
then $a(x:T_1).P =_\Delta a(x:T_2).Q$.

The quantification on T in the premises is finite: any type has only finitely-many subtypes.

Theorem 3. $\mathcal{P}' \vdash P =_\Delta Q$ *iff* $P \asymp_\Delta Q$, *where* P *and* Q *are closed.*

By contrast, we have tried and failed to obtain the counterpart of Theorem 1 for $\asymp$. The encountered problem is discussed at the end of this section. Using Lemma 2, that relates $\asymp$ to $\sim$, we however obtain an indirect axiomatisation of $\asymp$.

We say that $P_1 \asymp_\Delta P_2$ *under* Γ_1, Γ_2 if $\Gamma_i <: \Delta$ and $\Gamma_i \vdash P_i$ $(i = 1,2)$. From Δ, P_i, Γ_i, we can construct a finite environment $Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)$ (see [3] for the details) and relate $\sim$ to $\asymp$.

Lemma 2. $P_1 \asymp_\Delta P_2$ *under* Γ_1, Γ_2 *iff* $P_1 \sim_{\Delta, Env(\Delta,P_1,P_2,\Gamma_1,\Gamma_2)} P_2$.

As a consequence of this lemma, we obtain the following theorem.

Theorem 4. $P_1 \asymp_\Delta P_2$ *under* Γ_1, Γ_2 *iff* $\mathcal{A} \vdash P_1 =_{\Delta, Env(\Delta,P_1,P_2,\Gamma_1,\Gamma_2)} P_2$.

Directly axiomatizing $\asymp$ appears far from straightforward due to complications entailed by subtyping. We consider an example. Let $T \stackrel{\text{def}}{=} \mathtt{unit}$ and

$$\begin{aligned}
\Delta &\stackrel{\text{def}}{=} a : \mathtt{obo}T, y : \mathtt{ob}T \\
R &\stackrel{\text{def}}{=} \tau.((\nu c : \mathtt{b}T)\bar{y}c.\bar{c} + a(x : \mathtt{bo}T).[x = y]\tau) \\
R_1 &\stackrel{\text{def}}{=} \tau.((\nu c : \mathtt{b}T)\bar{y}c.\mathbf{0} + a(x : \mathtt{bo}T).[x = y]\tau) \\
R_2 &\stackrel{\text{def}}{=} \tau.((\nu c : \mathtt{b}T)\bar{y}c.\bar{c} + a(x : \mathtt{bo}T).\mathbf{0}).
\end{aligned}$$

It holds that

$$R + R_1 + R_2 \asymp_\Delta R_1 + R_2.$$

Here y can be instantiated by channels with subtypes of $\mathtt{ob}T$, which can be seen in Figure 1 (b). When y is instantiated by a channel with type $\mathtt{bo}T$, we can simulate R with R_1. For other subtypes of $\mathtt{ob}T$, we can simulate R with R_2. That is, we have two equivalent processes, say P and Q, with a free variable y, and the actions from a summand of P have to be matched by different summands of Q, depending on the types used to instantiate y. It appears hard to capture this relationship among terms using axioms involving only the standard operators of the π-calculus.

All bisimilarities considered so far in the paper are in the "late" style [12]. See [3] for the "early" versions and the corresponding proof systems.

Acknowledgements. We are grateful to Catuscia Palamidessi, Pierre-Louis Curien and the anonymous referees for comments on a preliminary version of the paper.

References

1. M. Boreale and R. De Nicola. Testing equivalences for mobile processes. *Journal of Information and Computation,* 120:279–303, 1995.
2. M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *Proceedings of LICS '98.* IEEE, Computer Society Press, July 1998.
3. Y. Deng and D. Sangiorgi. Towards an algebraic theory of typed mobile processes (full version). `http://www-sop.inria.fr/mimosa/personnel/Davide.Sangiorgi/mypapers.html`.
4. M. Hennessy and J. Rathke. Typed behavioural equivalences for processes in the presence of subtyping, 2003. To appear in Mathematical Structures in Computer Science.
5. M. Hennessy and J. Riely. Resource access control in systems of mobile agents. In U. Nestmann and B. C. Pierce, editors, *Proceedings of HLCL '98,* volume 16.3 of *ENTCS,* pages 3–17. Elsevier Science Publishers, 1998.
6. H. Lin. Symbolic bisimulation and proof systems for the π-calculus. Technical Report 7/94, School of Cognitive and Computing Sciences, University of Sussex, UK, 1994.
7. H. Lin. Complete inference systems for weak bisimulation equivalences in the π-calculus. In P. D. Mosses, M. Nielsen, and M. I. Schwarzbach, editors, *Proceedings of TAPSOFT '95,* volume 915 of *LNCS,* pages 187–201. Springer, 1995.
8. R. Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, May 1999.
9. J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Journal of Information and Computation,* 120(2):174–197, 1995.
10. B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science,* 6(5):409–454, 1996.
11. D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica,* 33:69–97, 1996.
12. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.

Ecological Turing Machines

Bruno Durand[1], Andrei Muchnik[2], Maxim Ushakov[3], and Nikolai Vereshchagin[3]*

[1] Laboratoire d'Informatique Fondamentale de Marseille, CNRS - Université de Provence, CMI, 39 rue Joliot-Curie, 13453 Marseille Cedex 13, France;
Bruno.Durand@lif.univ-mrs.fr
[2] Institute of New Technologies, Moscow, muchnik@lpcs.math.msu.su
[3] Moscow State University
{ushakov,ver}@mccme.ru

Abstract. The goal of this paper is to investigate the power of Turing machines with homogeneous memory. The standard models of Turing machines often use tricks such as "special" symbols (delimiter) or several different tapes. These tricks are not natural: computers use only 0's and 1's and operating systems consider memory as a whole. When memory is divided into several parts, (e.g. hard disks, devices …) they play the same role and we cannot say that one of them is devoted to input while the rest is a working space.

We address the question of computing power of variants of Turing machines with no delimiter, and also investigate how a Turing machine is forced to transform its environment while computing. For this last question, we consider the rest of its tape as an oracle and see whether it is possible to compute all recursive functions relativized to this oracle. If yes it means that, in the considered model, Turing machines can perform computations without destroying their environment (the model is thus called *ecological*), otherwise computing implies environment transformation.

These problems seem at first sight straightforward but they are not. We could prove 4 main results explained below, but some very simple and intuitive models are yet not clear (we do not know if they are as powerful as standard Turing machines).

1 Introduction

In "real-life" computers the memory is a uniform sequence of bits, so the standard tricks of adding delimiter symbols seem completely artificial in this context. The separation of the memory into several tapes is not completely artificial since in computer, different kinds of memory are treated separately but the idea that one of them is devoted to input and the other to workspace is not natural.

In this paper, work alphabet is equal to the input alphabet. The input is written on the work tape and no additional tape is permitted. We call such

* The work was done while visiting LIF, CNRS - University of Provence; also supported in part by the RFBR grants 02-01-22001, 03-01-00475.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 457–468, 2004.

machines "ecological" because they should interact with environment without any pollution. The input string to such a machine is written on the tape starting from the leftmost cell and is not delimited by any end-marker. All the cells to the right of the input may contain any garbage, that is, any symbols of the input alphabet. The computation is required to give the same result whatever garbage follows the input. Obviously, if an ecological machine computes a result on some input x then it also computes the same result on all inputs of the form xy. Hence such a machine can compute only functions f satisfying the following requirement: if $f(x)$ is defined and x is a prefix of x' then $f(x')$ is defined too and $f(x') = f(x)$. Such functions are called *prefix* functions.

We present 4 main results. The first 3 (Theorems 1, 2 and 3) address the computational power of such a model. The last one (Theorem 4) explains how computations modify the environment. The environment can be seen as the content of the memory of the computer before the computation starts, and we want to relate this to oracle computations.

We first focus on the question: "Are ecological Turing machines equivalent to usual Turing machines on the domain of prefix functions?" Our main result states that for the above described model of ecological Turing machines with 1 semi-infinite tape the answer is negative: there is a computable prefix function that cannot be computed by any ecological Turing machine with 1 semi-infinite tape. In this result, it is important that there is no delimiter at the end of the input. Note that if there is a special delimiter at the end of the input then every computable function can be computed by an ecological machine (machines can distinguish the delimiter from other tape symbols but cannot write it).

We consider similar questions for multi-tape Turing machines and multi-tape machines with bi-infinite tapes. For the reasons explained above we do not want to specify different rules for each tapes. They are treated as different memory sources, nothing else. Ecological Turing machines with s semi-infinite tapes receive s input strings, each string is written on a separate tape. We show that, in contrast to the case $s = 1$, for every $s \geq 2$ every computable function f of s arguments that is prefix with respect to each argument can be computed by an ecological machine with s semi-infinite tapes.

In the case of machines with s bi-infinite tapes the input to machines is an s-tuple of pairs of binary strings. Each pair is written on a separate tape: one string before the head and the other after it. For every s such a model is weaker than usual Turing machines: there is a computable prefix function of $2s$ variables that cannot be computed by any ecological Turing machine. Here, it is important that every tape has two inputs. If some tape has only one input written to the right of the head (say) then the resulting ecological model is equivalent to usual Turing machines: for every s every computable prefix function of $2s-1$ arguments can be computed by an ecological Turing machine with s bi-infinite tapes that receives one argument on the first tape and the other $2s-2$ on other tapes, one string to the left of the head and the other to the right of it.

Using the same technique we prove the following fact: there is a total (not computable) function from $\{0, 1\}^*$ to $\{0, 1\}$ and a bi-infinite binary sequence α

with the following two properties: f is computable relative to α (that is, it can be computed by a usual Turing machine with oracle α) but f cannot be computed by any ecological Turing machine with an extra input tape of the length equal to the length of the input and with a bi-infinite work tape initially containing α.

2 Ecological Machines with Semi-infinite Tapes

Let $\{0, 1\}^*$ denote the set of all finite binary strings. Consider first Turing machines with one semi-infinite tape and the tape alphabet {0,1,$} that are not allowed to write $ (the end-marker). That is, no instruction of the machine has $ in the right hand side. The computation starts with $x\$$ written in $|x| + 1$ leftmost cells of the tape. All the other cells may contain initially any symbols (the garbage). For simplicity we consider only machines outputting 0 or 1. More specifically, we assume that machines have two final states s_0, s_1. The machine halts only if it enters either of these states and the output is 0 (1) when it enters s_0 (s_1). We assume that at any step of the computation the machine knows whether the head is in the leftmost cell. In usual Turing machine model we do not assume this, as the machine can label the leftmost cell by a special marker. For ecological model this assumption is important.

We say that such a machine M computes f if it outputs $f(x)$ on the input x for every x in the domain of f whatever garbage is written to the right of the input. If $f(x)$ is not defined the machine can do anything.

Lemma 1. *Every partial computable prefix function* $f : \{0,1\}^* \to \{0,1\}$ *can be computed by such a machine.*

Proof. Consider a usual Turing machine M (with no restrictions on the tape alphabet) computing f. Let M have m tape symbols. Fix any k with $2^k > m$ and construct a Turing machine N of the above type simulating M. To this end encode each symbol of the M's tape alphabet by a k-bit sequence so that 0^k is not used as a code. The machine N uses the part of the tape to the right of the $ sign to simulate the work tape of M. Each symbol of M's tape alphabet is represented as its code and 0^k is used to indicate the first cell of the tape of M that has not yet been visited. The only non-trivial thing is the initialization stage, that is, copying the input to the right of the end-marker $. First we copy the first symbol of the input replacing it by 1. Then we copy the second one, the third one, and so on, so that at any time the copied part of the input is replaced by 00…01. Thus we can find the first non-copied symbol as the symbol that comes after the first 1 from the left.

Now consider the same model but this time assume that the input x is not delimited by the end-marker. The tape alphabet is {0,1}. Again all cells to the right of the input may contain a garbage. Formally, every command of a machine has the form $s, a, b \mapsto s', a', m$ where a is the scanned symbol, s is the internal state of the machine, b is equal to 1 if the head scans the leftmost cell and to 0 otherwise, s' is the new state of the machine, $a' \in \{0,1\}$ is the new symbol

replacing the scanned one, and $m \in \{-1, 0, 1\}$ is the movement of the head. We say that such a machine M computes a partial function $f : \{0,1\}^* \to \{0,1\}$ if for every x in the domain of f the machine halts in the state $s_{f(x)}$ if at the start the tape contains $x * * * \ldots$ (stars represent any symbols of {0,1}). Here s_0, s_1 are final states of M. If $f(x)$ is not defined the machine can do anything. Such machines are called *ecological Turing machines with one semi-infinite tape.* Obviously every ecological machine computes a partial prefix function f from {0,1}* to {0,1}.

The power of ecological machines seems to be very restricted: to perform a computation, the machine should write something on the tape, thus corrupting the input data. Therefore, it seems quite plausible that the ecological computational model is weaker than the usual one. Our main result confirms this intuition: there is a computable partial prefix function that cannot be computed by ecological machines.

Theorem 1. *There is a computable partial prefix function $f : \{0,1\}^* \to \{0,1\}$ that is not computable by ecological Turing machines with one semi-infinite tape.*

Proof. Let $g : \mathbb{N} \to \mathbb{N}$ be a computable function that grows sufficiently fast. We will see further how much is this "sufficiently". Consider the following function $f(x)$. It is defined on all the strings x of the form $x = 0^k 1 y u v$ where $|y| = 2^{g(k)}$, $|u| = g(k)$ and v is any string. Consider u as the binary expansion of a number n in the range $0, \ldots, 2^{g(k)} - 1$. Then $f(x) = 1$ if nth bit of y is 1 and 0 otherwise.

Obviously f is a computable prefix function. Informally, ecological machines cannot compute f because while the head moves from the beginning of the tape to the place where n is written, the machine should store somewhere the distance between the current position of the head and the beginning of n. Thus it has to corrupt some part of y.

Assume that the function f is computable by some ecological machine with one semi-infinite tape. Fix a large k and a string y of length $2^{g(k)}$. Assume that the string y contains both 0 and 1. Run the machine on the input $0^k 1 y 000 \ldots$. At some time $\tilde{t}$ the head reaches the cell with the first bit of n, that is, the cell number $N = k + 1 + 2^{g(k)} + 1$. We claim that $\tilde{t}$ is much larger than N.

Let $k < i < N$ be a boundary between two consecutive tape cells (k is the boundary between the last 0 and 1 in $0^k 1$). Consider the sequence of states of the machine at all the times when the head crosses the boundary i (we consider the run up to the time $\tilde{t}$ only). This sequence is called the *crossing sequence* [1] *at boundary i*. We will prove that crossing sequences at different boundaries are different (as sequences).

Lemma 2. *If a machine halts on all inputs of the form $0^k 1 \alpha$ ($\alpha : \mathbb{N} \to \{0,1\}$) and on an input I of this form it reaches for the first time the cell $N > k + 1$ at the time $\tilde{t}$, then all crossing sequences of the machine up to time $\tilde{t}$ at all boundaries i where $k < i < N$ are different.*

[1] The notion of crossing sequence was used in [2,3] to prove that one-tape Turing machines cannot recognize the symmetry of input x in $o(|x|^2)$ steps.

Assume the lemma is true.

Let y of length $2^{g(k)}$ have both 0 and 1. Then on input $0^k1y000\ldots$ the machine at some time reaches the cell number $N = k+1+2^{g(k)}+1$. By the lemma all the crossing sequences are different. Let M have Q states. The number of different crossing sequences of length less than l is equal to $Q^0+Q^1+\ldots Q^{l-1}$ and is less than Q^l. Letting $l = \frac{g(k)-1}{\log_2 Q}$ we see that for at least half of the boundaries the crossing sequence is longer than $\frac{g(k)-1}{\log_2 Q}$. Therefore the machine M reaches the boundary N not faster than in $\frac{(g(k)-1)2^{g(k)}}{\log_2 Q}$ steps. All the configurations of the machine up to this time are different (a configuration includes the state, the contents of the first $2^{g(k)}+k+1$ cells, and the position of the head). The sets of configurations for different y's are disjoint too, because if we start the machine in each configuration and try all possible n, we can find y.

Hence, the total number of configurations for all y is not less than

$$\frac{(2^{2^{g(k)}}-2)2^{g(k)}(g(k)-1)}{\log_2 Q}.$$

On the other hand, the total number of all possible configurations is

$$Q \cdot 2^{k+1+2^{g(k)}}(k+1+2^{g(k)}).$$

Neglecting multiplicative constants (not depending on k) we see that the first number is equal to $2^{2^{g(k)}+g(k)}g(k)$ and the second number to $2^{2^{g(k)}+g(k)+k}$. Their ratio is $g(k)/2^k$. Letting say $g(k) = 2^{2k}$ we get a contradiction if k is sufficiently large.

It remains to prove the lemma.

Proof. Assume that the crossing sequences at boundaries $i > k$ and $i+j$ in the computation on the input I coincide for some $j > 0$. Let w stand for the part of I to the left of i and v between the boundaries i and $i+j$. We will prove that the machine M does not halt on the input *wvvv*....

The proof is by "cut-and-paste" method which is rather folklore. Nevertheless we put it here for completeness.

Let A denote the computation on the input I and B on the input *wvvv*.... Let the crossing sequences in the computation A at boundaries i and $i+j$ be q_1, q_2, ..., q_n. We indicate only the states of the machine when it crosses the boundary; the direction is obvious: odd states correspond to crossings from left to right and even states correspond to crossings from right to left.

Let us partition the tape into zones as follows. The zone 0 stretches from the beginning of the tape up to the boundary i, zone 1 from the boundary i up to $i+j$, zone 2 from $i+j$ to $i+2j$ etc.

We will compare the computation B within zone k with the computation A within zone $p(k)$ where

$$p(k) = \begin{cases} 0, & \text{if } k = 0, \\ 1, & \text{if } k \neq 0. \end{cases}$$

We say that computation B is correct at time T with respect to zone k if there is a time t in the computation A such that the following holds.

a. The content of zone k in the computation B at the time T is equal to the content of zone $p(k)$ in the computation A at the time t.
b. Let l denote the number of crossings, in the computation A, of the left boundary of zone $p(k)$ up to time t (if $p(k) = 0$ then l is undefined), and m the number of crossings of the right boundary. Then in the computation B up to time T the left boundary of zone k has been crossed exactly l times, in the states $q_1, \ldots, q_l$, and the right boundary has been crossed exactly m times in the states $q_1, \ldots, q_m$. This implies that if at the time t in computation A the head is to the left of (to the right of, within) zone $p(k)$ then at time T in computation B the head is to the left of (to the right of, within) the zone k.
c. If in the computation B at time T the head is within the zone k and scans mth symbol of the zone, and the machine is in the state q then in the computation A at time t the head scans mth symbol of the zone $p(k)$, and the machine is in the same state q.

Let us prove by induction that the computation B is correct at all the times with respect to all zones. The base of induction is evident. Let us prove the induction step.

Assume first that at the time $T+1$ the machine is in the same zone k as it is at time T. By induction hypothesis computation B is correct at time T with respect to zone k with some t. The states of the machine at time T in the computation B and at time t in the computation A coincide, and the scanned symbols coincide too, as the contents of the corresponding zones coincide. Therefore the states, head positions and contents of zones at times $T + 1$ and $t + 1$ coincide. Hence B is correct at time $T+1$ with respect to zone k. The correctness with respect to other zones obviously is not disturbed.

Assume that the machine crosses a boundary between zones, say, comes from zone k into zone $k + 1$. By induction hypothesis for zone k there is a time t_k in the computation A. At time $t_k + 1$ in the computation A the head comes out the zone $p(k)$ into the zone $p(k) + 1$ in the state q_{m+1}, where m is the number from the item b. In the computation B, at time T the content of zone k coincides with the content of zone $p(k)$ at time t_k in the computation A. Therefore, in the computation B, the head crosses the boundary between zones in the same state q_{m+1}. Hence the computation is correct at time $T+1$ with respect to zone k with $t = t_k + 1$.

Let us prove that the computation is correct at time $T + 1$ with respect to zone $k + 1$ too. At time $T + 1$ the machine crosses the right boundary of zone k and enters zone $k + 1$ via its left boundary. By induction hypothesis the computation is correct at time T with respect to zone $k + 1$. Therefore at some time t in the computation A the content of zone $p(k + 1) = 1$ is equal to the content of zone $k + 1$ at time T in the computation B. And the last time before T in the computation B the head has crossed the right boundary of zone $k + 1$ in the state q_m (from right to left), and the left boundary in the state q_l (from

right to left), where l, m are numbers from the item b. We have just proved that at time $T + 1$ in the computation B the head returns in the zone $k + 1$ in the state q_{l+1}. Let t be equal to the time when in computation A the head crosses the boundary i for $l + 1$st time (from left to right). At that time the content of zone 1 in the computation A coincides with the content of zone $k + 1$ at time $T + 1$ in the computation B and the machine's state is the same as it is equal to q_{l+1}. Hence the computation is correct at time $T + 1$ with respect to zone $k + 1$ with this t.

The case when the head crosses the boundary from right to left is similar and the induction step is done.

Since computation B is correct at all times with respect to all zones, we conclude that the computation B is infinite. Indeed, at every step the state of the machine is equal to a state of the machine in the computation A before time $\tilde{t}$. Obviously that state is not final and hence in computation B the machine is never in a final state.

Now we proceed to machines having several semi-infinite tapes. Every command of a machine with k tapes has the form

$$s, a_1, b_1, \ldots, a_k, b_k \mapsto s', a'_1, \ldots, a'_k, m_1, \ldots, m_k$$

where $a_1, \ldots, a_k$ are the scanned symbols, s is the internal state of the machine, b_i is equal to 1 if the head on the ith tape scans the leftmost cell and to 0 otherwise, s' is the new state of the machine, $a'_i \in \{0, 1\}$ is the new symbol replacing the scanned symbol on ith tape, and $m_i \in \{-1, 0, 1\}$ is the movement of the head on ith tape. We say that such a machine M computes a partial function $f : (\{0,1\}^*)^k \to \{0,1\}$ if for every x in the domain of f the machine halts in the state $s_{f(x_1,\ldots,x_k)}$ if at the start the ith tape contains $x_i * * * \ldots$. If $f(x_1, \ldots, x_k)$ is not defined the machine can do anything. Such machines are called *ecological Turing machines with k semi-infinite tapes.* Obviously every ecological machine computes a partial prefix function f from $(\{0,1\}^*)^k$ to $\{0,1\}$. The latter means that if $f(x_1, \ldots, x_k)$ is defined and x'_i extends x_i then $f(x'_1, \ldots, x'_k)$ is defined and equal to $f(x_1, \ldots, x_k)$.

Theorem 2. *For every $k \geq 2$ every partial computable prefix function $f : (\{0,1\}^*)^k \to \{0,1\}$ is computable by an ecological machine with k semi-infinite tapes.*

Proof. First we define another computational model—a version of counter machines. The machine has a finite state control unit and n *counters,* each containing a natural number, initially zero. Besides, the machine has k semi-infinite tapes, all of them one-way (one-way tape is a tape with a head that can move only to the right and at the start scans the leftmost cell). The control unit can increment and decrement the counters by 1 (decrementing a counter whose value is zero does not change its value), and ask whether the value of a counter is zero. Thus the difference with the model in the statement of the theorem is as follows:

the machine has extra n counters but is not allowed to move heads to the left. Formally, every command of a machine has the form

$$s, a_1, \ldots, a_k, b_1, \ldots, b_n \mapsto s', i_1, \ldots, i_n, m_1, \ldots, m_k$$

where s is the internal state of the machine, $a_1, \ldots, a_k$ are the scanned symbols, b_j is 1 if the value c_j stored in the jth counter is zero and 0 ohterwise, s' is the new state of the machine, $i_j \in \{-1, 0, 1\}$ is the increment/decrement of the jth counter (after executing the command the new content is equal to $c'_j = c_j + i_j$), and $m_i \in \{0, 1\}$ is the movement of the head on ith tape (note that $m_i = -1$ is not allowed). The input to such a machine is a tuple of k binary strings written initially on the tapes and followed by garbage. We say that such a machine M computes a partial function $f : (\{0,1\}^*)^k \to \{0,1\}$ if for every x in the domain of f the machine halts in the state $s_{f(x_1,\ldots,x_k)}$ if at the start the ith tape contains $x_i * * * \ldots$. If $f(x_1, \ldots, x_k)$ is not defined the machine can do anything. We will call these machines n *counter machines,* or machines with n counters.

Lemma 3. *For some constant n every partial computable function $f : (\{0,1\}^*)^k \to \{0,1\}$ is computable by a machine with n counters.*

Proof. If we have a sufficient number of counters, we can add, subtract, multiply, and divide numbers in two given counters. For example, to add numbers in counters A and B we need to decrease B simultaneously increasing A until B becomes zero. In a similar way we can implement subtraction; multiplication and division are implemented via addition and subtraction. So, we can check primality, and find primes in increasing order.

Let $f : (\{0,1\}^*)^k \to \{0,1\}$ be a partial computable function. Then f is computable by a Turing machine M having k semi-infinite one-way tapes and an extra read/write two-way work tape (as the work tape has no input at the start, we can use it in usual way as the memory in the computation). It suffices to simulate such a machine by a counter machine. To this end we need to keep the content of the work tape of M in one of the counters. We can do this by encoding each tape symbol as a number, and storing in the first counter the number $T = 2^{t_1} 3^{t_2} 5^{t_3} \ldots p_N^{t_N}$, where 2, 3, 5, ... , p_N are consecutive primes, and t_1, $t_2, \ldots, t_N$ are codes of symbols of the tape of the simulated machine. The second counter keeps the position of the head on the work tape of the simulated machine.

Moving the head of M on the work tape corresponds to increasing/decreasing the second counter. Changing the scanned symbol on the work tape corresponds to dividing/multiplying (several times) the value T of the first counter by p_i where i is the value stored in the second counter. This can be done using constant number of other counters as described above.

Moving the heads of M on input tapes corresponds to moving the same heads on input tapes (recall that the simulating machine has the same number if input tapes).

Lemma 4. *Every computable prefix function* $f: (\{0,1\}^*)^k \to \{0,1\}$ *is computable by a machine with 2 counters.*

Proof. By Lemma 3, for some n there is a machine with n counters that computes f. Encode values of counters into one number $S = 2^{c_1}3^{c_2}\dots p_n^{c_n}$, where, as before, p_i is ith prime number and c_i is the value of ith counter. We store the number S in the first counter. The second one will be used for intermediate computations.

Now we have to show how to simulate operations with counters. Incrementing and decrementing ith counter correspond to multiplying and dividing S by p_i. Multiplying is done like this: while the first counter is not zero, decrement it and then p_i times increment the other one. Division is done in a similar way.

When the simulated machine moves its head on any of the tapes so does the simulating 2 counter machine.

Note that the simulating 2 counter machine constructed in the proof of Lemma 3 has the following property: at any step of the simulation the second counter is empty when the 2 counter machine moves any of its heads. We call such 2 counter machines *correct.*

Thus, given a correct machine M_1 with 2 counters that computes f, we have to construct an ecological machine M_2 computing f. We will assume first that $k = 2$. Let $s_0s_1s_2\dots$ be the content of the first tape of M_1 and i the position of the head on it, $t_0t_1t_2\dots$ the content of the second tape of M_1 and j the position of the head on it. Let C_1, C_2 be the contents of the counters. The corresponding configuration of the ecological machine looks as follows: the first tape contains $00\dots1s_{i+1}s_{i+2}\dots$, the second tape $00\dots1t_{j+1}t_{j+2}\dots$. The position of the head is C_1 on the first tape and C_2 on the second one.

If the counter machine increments/decrements a counter, the ecological machine just moves the corresponding head. As the ecological machine knows whether the head is in the beginning of the tape, it knows whether the counter is empty. Assume that the counter machine moves either of its heads to the right and scans a new input symbol. Note that in this case the second counter is empty and hence the second head of the ecological machine is in the beginning of the tape. If the counter machine moves the second head then the ecological machine finds the first 1 on the second tape, replaces it by 0, scans the next symbol, replaces it by 1, and returns to the beginning of the tape. If the counter machine moves the first head then the ecological machine first copies the content of the counter to the second tape (moving both heads simultaneously in the opposite directions until the first one is in the beginning) and then does the same thing. After that it copies the content of the counter back on the first tape.

For $k > 2$ the simulation is entirely similar: we use the first two tapes as explained above to store counters. The heads on other tapes of simulating machine move just as the heads of the simulated machine (and they do not change the contents of the tapes).

Remark 1. It is important, in the above theorem, that the simulating machine knows whether its heads are at the leftmost cell. If machines do not know that

and are required not to try to come beyond the tape then the resulting model is weaker than usual Turing machine model. This can be proven by arguments similar to those used in the proof of Theorem 3 below.

3 Ecological Machines with Bi-infinite Tapes

Now we proceed to machines having several bi-infinite tapes. Every command of a machine with k bi-infinite tapes has the form

$$s, a_1, \ldots, a_k \mapsto s', a'_1, \ldots, a'_k, m_1, \ldots, m_k$$

where $a_1, \ldots, a_k$ are the scanned symbols, s is the internal state of the machine, s' is the new state of the machine, $a'_i \in \{0,1\}$ is the new symbol replacing the scanned symbol on ith tape, and $m_i \in \{-1,0,1\}$ is the movement of the head on ith tape.

There are two options to give inputs to machines with bi-infinite tapes. The first one is to write inputs (followed by garbage) to the right of the initial positions of heads. That is, an ecological machine with k semi-infinite tapes receives k inputs and computes a prefix function of k variables. In this case every computable prefix function is computable by an ecological machine.

The second option is to write two inputs on one bi-infinite tape: one input to the right of the initial position of the head and another one to the left in the reverse order. Note that if on at least one tape there is only one input then the ecological model is equivalent to the usual one. Therefore we will assume that every tape has two inputs. We say that such a machine M computes a partial function $f : (\{0,1\}^*)^{2k} \to \{0,1\}$ if for every x in the domain of f the machine halts in the state $s_{f(x_1,y_1,\ldots,x_k,y_k)}$ if at the start the ith tape contains $\cdots * * * y_i^R x_i * * * \ldots$ and the head is between y_i^R and x_i (y^R means y in reversed order). If $f(x_1, y_1, \ldots, x_k, y_k)$ is not defined the machine can do anything. Thus an ecological machine with k bi-infinite tapes computes a prefix function of $2k$ arguments. It turns that for every k such machines are weaker than usual ones.

Theorem 3. *For every $s \geq 1$ there exists a computable prefix function $f: (\{0,1\}^*)^{2s} \to \{0,1\}$ that is computable in the usual sense, but is not computable by any ecological Turing machine with s bi-infinite tapes.*

Proof. First, we prove the statement for $s = 1$. Consider the following function $f(x,y)$. It is defined on all the strings x, y of the form $y = 0^k 1 z u v$, $x = rw$, $|r| = |z| = 2^{2k}$, $|u| = 2k+1$. Consider u as the binary expansion of a number n in the range $0, \ldots, 2^{2k+1} - 1$. Then $f(x,y) = 1$ if nth bit of the string rz is 1 and 0 otherwise.

Informally, ecological machines cannot compute f because while the head moves from the beginning of the tape to the place where n is written, the machine should store somewhere the distance between the current position of the head and the beginning of n. Thus it has to corrupt some part of rz.

Fix k, r, z, P where P is a number in the range $1, \ldots, 2^{2k}$ and r, z are binary strings of length 2^{2k}. Run the machine with r to the left of the head and $0^k 1 z$ to

the right of the head. Consider the first time T the head moves beyond position P or $-P$ (position $-P$ is the Pth cell to the left of the initial position of the head). Let a stand for the content of the tape from the position -2^{2k} up to the position of the head and b the content of the tape from the position of the head up to the position $2^{2k}+k+1$ (at the time T). Let q be the state of the machine at the time T. To derive a contradiction we will prove that both r, z and P can be reconstructed given the pair $\langle q, ba\rangle$ unless the string rz consists of blocks 00 and 11. Note that we are not given the lengths of a, b and do not know the place where b ends in the string ba. Here the order in which we concatenate b and a is important—we do not claim that r, z and P can be reconstructed given the pair $\langle q, ab\rangle$. To reconstruct r, z and P it is enough to find a and b (then we can run the machine from the configuration at time T and try all possible n; recall that the machine has not yet visited the place where n is written before time T). To this end try all possible divisions of ba into b' and a' starting with the longest b'. For each division and for all $n \leq |rz|$ run the machine in the state q, with ba on the tape before the head, and $b'n$ after it. Combine the machine's outputs into string $r'z'$. If b' is longer than b, $b' = bb''$, then $r'z'$ will consist of blocks 00 and 11. Indeed, the machine will consider b'' as the most significant bits of n. And therefore the answer will not depend on several least significant bits of actual n. Hence the resulting string $r'z'$ will consist of blocks 00 and 11. This can be easily checked. If this is the case, the division is wrong, unless the string rz consists of blocks 00 and 11, and we proceed to the next division until the correct division is found.

The number of different triples $\langle r, z, P\rangle$ such that the string rz does not consist of blocks 00 and 11 is about $2^{2^{2k+1}}2^{2k}$ (it is equal to $(2^{2^{2k+1}} - 2^{2^{2k}})2^{2k}$) and the number of different pairs $\langle q, ba\rangle$ is $Q2^{2^{2k+1}+k+1}$ where Q stands for the number of the states of the machine. For large enough k the latter number is less than the former one and we obtain a contradiction.

The proof generalizes easily to the case $s > 1$. In that case the function $f(x_1, y_1, \ldots, x_s, y_s)$ is defined as follows. It is defined when $y_1 = 0^k 1 z_1 u_1 v_1$, $x_1 = r_1 w_1$, and $y_i = z_i u_i w_i$, $x_i = r_i w_i$, for $i > 1$, and $|r_i| = |z_i| = 2^{2k}$, $|u_i| = 2k + 1 + \lceil \log_2 s \rceil$. Let n_i be the number whose binary expansion is u_i and let n be equal to the sum of all n_i modulo $s2^{2k+1}$. Then $f(x_1, y_1, \ldots, x_s, y_s)$ is the nth bit of the string $Z = r_1 z_1 \ldots r_s z_s$.

The above proof is modified as follows. We consider the tuple $\langle r_1, z_1, \ldots, r_s, z_s, P\rangle$, where P is a number in the range $1, \ldots, 2^{2k}$, and consider the first time when some head moves beyond position P or $-P$. The strings a_i and b_i are defined in the same way as above: a_i is the content of ith tape from the position -2^{2k} to the current position of the head and b_i is the content of the ith tape from the position of the head up to position 2^{2k} ($2^{2k}+k+1$ if $i=1$).

To derive a contradiction we need to prove that Z and P can be reconstructed given $\langle q, b_1 a_1, \ldots, b_s a_s\rangle$ unless Z consists of blocks of 00 and 11. To do this, we find a_i and b_i for all i in succession as follows. Try all possible divisions of $b_i a_i$ into a_i' and b_i' starting from the longest b_i'. For each division run the machine,

writing on all tapes except ith one the strings $b_1a_1, \ldots, b_sa_s$ both before the heads and after them. On ith tape write b_ia_i before the head, and $b_i'n$ after it (for all $n \leq s2^{2k+1}$). If b_i' is longer than b_i, the resulting string Z' will consist of blocks 00 and 11. Thus we will know that the division is incorrect, as, for the correct division, the string Z' is a cyclic shift of Z, so it cannot consist of such blocks.

4 Computing Functions with an Oracle

Consider now ecological Turing machines of the following type. The machine has a two-way read/write finite input tape of the length equal to the length of the input. It has a bi-infinite work tape initially containing a bi-infinite binary sequence α. The machine knows whether the head on the input tape is in the beginning of the tape or in the end. Both the input tape alphabet and the work tape alphabet are binary.

Theorem 4. *There is a total function from* $\{0,1\}^*$ *to* $\{0,1\}$ *and a bi-infinite binary sequence* α *such that* f *is computable relative to* α *but* f *cannot be computed by any ecological Turing machine of the above described type.*

The proof is omitted due to space restrictions.

5 Open Questions

Many questions about the relative power of usual and ecological computational models remain open. We list some of them.

1. What about ecological machines with one bi-infinite tape and one semi-infinite tape? Are they weaker than usual machines?

2. Is there an analog of Theorem 4 for machines with one semi-infinite tape?

3. Is every prefix partial function $f(x, y)$ computable in polynomial time $\mathrm{poly}(|x|+|y|)$ computable also in polynomial time by an ecological machine with two semi-infinite tapes? Note that the simulation in the proof of Theorem 2 has exponential overhead in time.

4. What can be said of the power of ecological RAM (Random access machines)? The input data to such a machine is an infinite sequence of natural numbers stored in its registers.

References

1. V.N. Agafonov. Complexity of algorithms and computations: A course for students of University of Novosibirsk, part 1. Publishing house of University of Novosibirsk, 1975. (Russian)
2. Ya.M. Barzdin. Complexity of symmetry recognition on Turing machines, Problemy kibernetiki, v. 15 (1965), 245–248. (Russian)
3. F.C. Hennie. One tape off-line Turing machine computations. Information and Control, 8:6 (1965) 553–578.

Locally Consistent Constraint Satisfaction Problems (Extended Abstract)

Zdeněk Dvořák, Daniel Král', and Ondřej Pangrác

Department of Applied Mathematics and
Institute for Theoretical Computer Science (ITI)**
Charles University
Malostranské náměstí 25, 118 00 Prague, Czech Republic
{rakdver,kral,pangrac}@kam.mff.cuni.cz

Abstract. An instance of a constraint satisfaction problem is l-consistent if any l constraints of it can be simultaneously satisfied. For a fixed constraint type P, $\rho_l(P)$ denotes the largest ratio of constraints which can be satisfied in any l-consistent instance. In this paper, we study locally consistent constraint satisfaction problems for constraints which are Boolean predicates. We determine the values of $\rho_l(P)$ for all l and all Boolean predicates which have a certain natural property which we call 1-extendibility as well as for all Boolean predicates of arity at most three. All our results hold for both the unweighted and weighted versions of the problem.

1 Introduction

Constraint satisfaction problems form an important abstract computational model for a lot of problems arising in practice. This is witnessed by an enormous recent interest in the computational complexity of various constraint satisfaction problems [2,3,4,13]. However, some instances of real problems do not require all the constraints to be satisfied but it is enough to satisfy a large fraction of them. In order to maximize this fraction, the input can be usually pruned at the beginning by removing small sets of contradictory constraints. In this paper, we study for a fixed constraint type how large fraction of the constraints can be simultaneously satisfied if no l constraints are contradictory. Formally, an instance of the constraint satisfaction problem is *l-consistent* if any l constraints of it can be simultaneously satisfied.

This problem was first introduced and studied by Trevisan [11]. He showed that for each fixed $k \geq 2$ and each $l \geq 2$, if we allow as constraints all Boolean predicates of arity k, then there exist l-consistent problems in which the fraction of constraints which can be simultaneously satisfied does not exceed 2^{1-k} and the bound is tight. In the upper bound, he used only a single type of predicate (the

** Institute for Theoretical Computer Science is supported by Ministry of Education of Czech Republic as project LN00A056.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 469–480, 2004.

predicate $P(x_1, \ldots, x_k) = (x_1 \not\Leftrightarrow x_2 \Leftrightarrow x_3 \Leftrightarrow \cdots \Leftrightarrow x_k)$); his lower bound was based on a simple probabilistic argument similar to that used by Yannakakis [14] for locally consistent CNF formulas.

The variant of the problem for locally consistent CNF formulas is extremely well-studied as witnessed by a separate section (20.6) devoted to this concept in a recent monograph on extremal combinatorics by Jukna [7]. A CNF formula Φ is *l-consistent* if any l clauses of Φ can be satisfied. Formulas which are l-consistent are also called *l-satisfiable*. The number ρ_l^{SAT} denotes the largest fraction of clauses which can be satisfied in any l-consistent CNF formula. Clearly, $\rho_1^{\mathrm{SAT}} = 1/2$. The value $\rho_2^{\mathrm{SAT}} = \frac{\sqrt{5}-1}{2} \approx 0.6180$ was determined by Lieberherr and Specker [9]. They consequently established $\rho_3^{\mathrm{SAT}} = 2/3$ [10]. Later, Yannakakis [14] simplified proofs of both the lower bounds on ρ_2^{SAT} and ρ_3^{SAT} using a probabilistic argument. The value $\rho_4^{\mathrm{SAT}} \approx 0.6992$ has been recently computed by one of the authors [8]. Huang and Lieberherr [6] studied the asymptotic behavior and they proved $\lim_{l\to\infty} \rho_l^{\mathrm{SAT}} \leq 3/4$. The limit was settled by Trevisan [11] who showed $\lim_{l\to\infty} \rho_l^{\mathrm{SAT}} = 3/4$. Let us remark that the cases of 1, 2 and 3-consistent CNF formulas somewhat unexpectedly differ from the case of l-consistent formulas for $l \geq 4$. First, $\rho_l^{\mathrm{SAT}} = \rho_l^{2\mathrm{SAT}}$ for $l = 1, 2, 3$ but $\rho_4^{\mathrm{SAT}} < \rho_4^{2\mathrm{SAT}}$ where $\rho_l^{2\mathrm{SAT}}$ is the largest fraction of clauses which can be satisfied in any l-consistent 2-CNF formula, i.e., CNF formulas with clauses of sizes at most two. We suspect the inequality to be strict for all $l \geq 4$, i.e., $\rho_l^{\mathrm{SAT}} < \rho_l^{2\mathrm{SAT}}$ for all $l \geq 4$. Second, the values ρ_l^{SAT} for $l = 1, 2, 3$ coincide with the similar values defined for a "fractional" version of the problem (which are known for all $l \geq 1$ [8] and are equal to so-called Usiskin's numbers [12]) but the value ρ_4^{SAT} differs.

In the present paper, we study the more general problem of locally consistent constraint satisfaction problems. We restrict our attention to problems whose constraints are copies of a single Boolean predicate P. The arguments of the predicates can be both positive and negative literals. Similarly as in the case of CNF formulas, $\rho_l(P)$ denotes the largest possible fraction of constraints which can be simultaneously satisfied in any l-consistent instance. We determine the values of $\rho_l(P)$ for all $l \geq 1$ and all Boolean predicates P of arity at most three (see Tables 1 and 2) and for all Boolean predicates which are 1-extendable. A predicate P is said to be 1-*extendable* if it has the following property: If we fix one of its arguments, we can choose the remaining ones in such a way that the predicate is satisfied. Let us point out a somewhat exceptional case of the predicate $P(x, y, z) = x \wedge (y \vee z)$ which is not 1-extendable (fix x to be false). Therefore, our general Theorem 1 does not apply. In Section 5, we show for this predicate that $\rho_1(P) = 3/8$, $\rho_2(P) = 2\sqrt{3}/9$ and somewhat surprisingly that $\rho_l(P) = \rho_{l-2}^{2\mathrm{SAT}}$ for all $l \geq 3$. Since the values $\rho_l^{2\mathrm{SAT}}$ were exactly computed before only for $l = 1, 2, 3$, we have to prove a special result on structure of locally consistent 2-CNF formulas (Lemma 8) which is later used in the analysis of the predicate $P(x, y, z) = x \wedge (y \vee z)$ and which also yields a formula for $\rho_l^{2\mathrm{SAT}}$ (Corollary 1). Let us remark that all our results hold both for the unweighted and weighted versions of the studied problems, i.e., the instances witnessing the upper bounds contain each constraint at most once and our lower bound proofs

translate smoothly for instances with weighted constraints. From the algorithmic point of view, our results can be interpreted in the following way: The simplest probabilistic algorithms (of the kind used in [8,11,14]) are approximation algorithms for locally consistent CSPs with optimum worst-case performance.

Table 1. The values $\rho_l(P)$ for all non-isomorphic essentially unary and binary Boolean predicates.

$\sigma(P)$	P	$l=1$	$l\geq 2$
1	x	1/2	1
1	$x \wedge y$	1/4	1
2	$x \Leftrightarrow y$	1/2	
3	$x \vee y$	3/4	

Table 2. The values $\rho_l(P)$ for all non-isomorphic essentially ternary Boolean predicates.

$\sigma(P)$	P	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l\geq 6$	$l\to\infty$
1	$x \wedge y \wedge z$	1/8	1	1	1	1	1	1
2	$x \Leftrightarrow y \Leftrightarrow z$	1/4	1/4	1/4	1/4	1/4	1/4	1/4
	$x \wedge (y \Leftrightarrow z)$	1/4	8/27	1/2	1/2	1/2	1/2	1/2
3	exactly one	3/8	3/8	3/8	3/8	3/8	3/8	3/8
	$x \wedge (y \vee z)$	3/8	$\frac{2\sqrt{3}}{9}$	1/2	$\frac{\sqrt{5}-1}{2}$	2/3	$\rho_{l-2}^{2\mathrm{SAT}}$	3/4
	$(x \Leftrightarrow y) \wedge (x \Rightarrow z)$	3/8	3/8	3/8	3/8	3/8	3/8	3/8
4	$x \Rightarrow y \Rightarrow z$	1/2	1/2	1/2	1/2	1/2	1/2	1/2
	$(x \wedge y) \Leftrightarrow z$	1/2	1/2	1/2	1/2	1/2	1/2	1/2
	at most one	1/2	1/2	1/2	1/2	1/2	1/2	1/2
	one or three	1/2	1/2	1/2	1/2	1/2	1/2	1/2
5	$\neg$ exactly one	5/8	5/8	5/8	5/8	5/8	5/8	5/8
	$x \vee (y \wedge z)$	5/8	5/8	5/8	5/8	5/8	5/8	5/8
	$(x \Leftrightarrow y) \vee (x \wedge z)$	5/8	5/8	5/8	5/8	5/8	5/8	5/8
6	$\neg(x \Leftrightarrow y \Leftrightarrow z)$	3/4	3/4	3/4	3/4	3/4	3/4	3/4
	$x \vee (y \Leftrightarrow z)$	3/4	3/4	3/4	3/4	3/4	3/4	3/4
7	$x \vee y \vee z$	7/8	7/8	7/8	7/8	7/8	7/8	7/8

2 Preliminaries

In this paper, we mainly deal with constraints which are Boolean predicates and we prefer to call them *predicates* to emphasize their kind. If P is a Boolean predicate, $\sigma(P)$ denotes the number of combinations of arguments which satisfy P. If the constraint satisfaction problem consists of copies of a single constraint P, its instances are called *P-systems*. The arguments of the predicates may be

both positive and negative literals, but a single variable cannot be contained in two distinct arguments of the same predicate. The goal is to find a truth assignment which satisfies the largest number of the predicates. If Σ is a P-system, then $\rho(\Sigma)$ is the largest fraction of the predicates of Σ which can be simultaneously satisfied. Hence, $\rho_l(P) = \inf \rho(\Sigma)$ where the infimum is taken over all l-consistent P-systems Σ.

Two Boolean predicates P and P' are *isomorphic* if they differ by permutation of the arguments and negations of some of them, e.g., if $P(x_1, x_2) = P'(x_2, \neg x_1)$, then the predicates P and P' are isomorphic. Clearly, if P and P' are two isomorphic predicates, then $\rho_l(P) = \rho_l(P')$ for all $l \geq 1$. A k-ary predicate is *essentially k-ary* if it depends on all its k arguments. If the predicate P is not essentially k-ary, it is isomorphic to a predicate P' such that $P'(x_1, \ldots, x_k) = P''(x_1, \ldots, x_{k-1})$ for some $(k-1)$-ary Boolean predicate P''. It is not hard to see that $\rho_l(P) = \rho_l(P') = \rho_l(P'')$ for all $l \geq 1$ in such case. Hence, in order to determine $\rho_l(P)$ for all unary, binary and ternary Boolean predicates P, it is enough to compute the values for representatives of isomorphism classes of essentially unary, binary and ternary Boolean predicates.

We conclude this section by stating three simple observations on locally consistent systems of Boolean predicates:

Lemma 1. *Let P be a k-ary Boolean predicate P. It holds that $\rho_l(P) \geq \sigma(P)/2^k$ for all $l \geq 1$.*

Lemma 2. *It holds that $\rho_1(P) = \sigma(P)/2^k$ for each k-ary predicate P.*

Lemma 3. *Let P be a k-ary predicate with $\sigma(P) = 1$. Then, $\rho_1(P) = 2^{-k}$ and $\rho_l(P) = 1$ for every $l \geq 2$.*

3 1-Extendable Boolean Predicates

In this section, we present an upper bound on $\rho_l(P)$ which holds for all 1-extendable Boolean predicates. The *dependence graph* $G(\Sigma)$ of a P-system Σ is the multigraph whose vertices are predicates of Σ and the number of edges between two predicates p_1 and p_2 of Σ is equal to the number of variables which appear in arguments of both the predicates p_1 and p_2 (regardless whether they appear as positive or negative literals). The *girth* of a P-system Σ is the length of the shortest cycle contained in $G(\Sigma)$. In particular, if the girth of Σ is three or more, then $G(\Sigma)$ contains no parallel edges. The following lemma relates the girth of a P-system with its consistency (we leave out the proof due to space limitations):

Lemma 4. *Let P be a 1-extendable predicate and Σ a P-system. If the girth of Σ is at least $l \geq 3$, then Σ is $(l-1)$-consistent.*

Let us recall now Chernoff's inequality [5]:

Lemma 5. *Let X be a random variable equal to the sum of N zero-one independent random variables such that each of them is equal to 1 with probability p. Then, the following holds for every $0 < \delta < 1$:*

$$\text{Prob}(X \geq (1+\delta)pN) \leq e^{-\frac{\delta^2 pN}{3}} \quad \textit{and} \quad \text{Prob}(X \leq (1-\delta)pN) \leq e^{-\frac{\delta^2 pN}{2}}.$$

We are now ready to determine the values $\rho_l(P)$ for all $l \geq 1$ and all 1-extendable Boolean predicates P. Note that the proof of Theorem 1 generalizes the standard construction of "random" graphs with large girth.

Theorem 1. *Let P be a k-ary Boolean predicate which is 1-extendable. Then, $\rho_l(P) = \sigma(P)/2^k$ for all $l \geq 1$.*

Proof. If $l = 1$, the statement follows from Lemma 2. Fix a k-ary 1-extendable Boolean predicate P and an integer $l \geq 2$. By Lemma 1, $\rho_l(P) \geq \sigma(P)/2^k$. For each $\varepsilon > 0$, we construct an l-consistent P-system Σ with $\rho(\Sigma) \leq (1+\varepsilon)\sigma(P)/2^k$. This will yield the equality $\rho_l(P) = \sigma(P)/2^k$.

Let us consider a positive real $0 < \delta < 1$ whose exact value is chosen later. Let $n \geq 2k$ be an integer which will also be chosen later. We construct an l-consistent P-system Σ with variables $x_1, \ldots, x_n$. Let S_n be the set of all possible predicates P with variables $x_1, \ldots, x_n$; the number of predicates contained in S_n is $N = 2^k n!/(n-k)!$. Note that $N \geq n^k$ because $n \geq 2k$. Construct a P-system Σ_0 from S_n by including each predicate of S_n to Σ_0 randomly and independently with probability $p = n^{-(k-1)+1/2l}$. By Lemma 5, the probability that the number $|\Sigma_0|$ of predicates of Σ_0 is smaller than $(1-\delta)pN$ is at most the following:

$$\text{Prob}(|\Sigma_0| \leq (1-\delta)pN) \leq e^{-\frac{\delta^2 pN}{2}} \leq e^{-\frac{\delta^2 n^{-(k-1)+1/2l} n^k}{2}} \leq e^{-\frac{\delta^2 n^{1+1/2l}}{2}} \qquad (1)$$

Observe that $G(S_n)$ contains at most $2^{k\lambda} k^{2\lambda} n^{(k-1)\lambda}$ cycles of length λ. Thus, the expected number of cycles of length at most l in $G(\Sigma_0)$ does not exceed:

$$\sum_{\lambda=2}^{l} 2^{k\lambda} k^{2\lambda} n^{(k-1)\lambda} p^{\lambda} \leq 2^{kl} k^{2l} \sum_{\lambda=2}^{l} n^{\lambda/2l} \leq 2^{kl} k^{2l} l n^{1/2}.$$

By Markov's inequality, the number of cycles of length at most l in $G(\Sigma_0)$ is smaller or equal to $2 \cdot 2^{kl} k^{2l} l n^{1/2}$ with probability at least 1/2.

Each truth assignment for the variables $x_1, \ldots, x_n$ satisfies exactly $\sigma(P)2^{-k}N$ predicates of S_n. The expected number of the predicates satisfied by a single fixed truth assignment τ is $\sigma(P)2^{-k}pN$. Since there are 2^n truth assignments, Lemma 5 implies that the probability that there is a satisfying assignment which satisfies more than $(1+\delta)\sigma(P)2^{-k}pN$ predicates is at most:

$$\sum_{\tau} \text{Prob}(\#\text{ satisfied clauses by } \tau \geq (1+\delta)\sigma(P)2^{-k}pN) \leq$$

$$2^n e^{-\frac{\delta^2 \sigma(P) 2^{-k} pN}{3}} \leq 2^n e^{-\frac{\delta^2 \sigma(P) 2^{-k} n^{1+1/2l}}{3}} = e^{(\ln 2)\cdot n - \frac{\delta^2 \sigma(P) 2^{-k}}{3} \cdot n^{1+1/2l}} \qquad (2)$$

We now choose the integer n to be any (large enough) integer that both the upper bounds (1) and (2) are smaller than 1/4 and the last inequality in (4) below holds. By (1) and (2), the random P-system Σ_0 has the following three properties with positive probability:

- Σ_0 contains at least $(1-\delta)pN$ predicates.
- $G(\Sigma_0)$ contains at most $2 \cdot 2^{kl} k^{2l} l n^{1/2}$ cycles of length at most l.
- There is no truth assignment satisfying more than $(1+\delta)\sigma(P)2^{-k}pN$ predicates of Σ_0.

Fix a P-system Σ_0 which has these three properties. The desired P-system Σ is obtained from Σ_0 by removing all the predicates contained in all cycles of $G(\Sigma_0)$ whose length is at most l. Hence, $G(\Sigma)$ contains no cycle of length at most l and its girth is at least $l+1$. Since P is 1-extendable, Σ is l-consistent by Lemma 4. In addition, the first two properties of Σ_0 imply that Σ contains at least $(1-\delta)pN - (2^{kl+1}k^{2l}ln^{1/2}) \cdot l$ predicates. On the other hand, the third property yields that no truth assignment can satisfy more than $(1+\delta)\sigma(P)2^{-k}pN$ predicates of Σ. Hence:

$$\rho(\Sigma) \leq \frac{(1+\delta)\sigma(P)2^{-k}pN}{(1-\delta)pN - 2^{kl+1}k^{2l}l^2n^{1/2}} \leq \frac{(1+\delta)\sigma(P)}{(1-\delta)2^k - \frac{2^{kl+k+1}k^{2l}l^2n^{1/2}}{pN}} \tag{3}$$

Observe the following (the last inequality follows from the choice of n):

$$\frac{2^{kl+k+1}k^{2l}l^2n^{1/2}}{pN} \leq \frac{2^{kl+k+1}k^{2l}l^2n^{1/2}}{n^{-(k-1)+1/2l}n^k} = \frac{2^{kl+k+1}k^{2l}l^2}{n^{1/2+1/2l}} \leq \delta 2^k \tag{4}$$

The inequalities (3) and (4) yield the following:

$$\rho(\Sigma) \leq \frac{(1+\delta)\sigma(P)}{(1-\delta)2^k - \delta 2^k} = \frac{1+\delta}{1-2\delta} \cdot \frac{\sigma(P)}{2^k}.$$

Note that for each $\varepsilon > 0$, we can choose $0 < \delta < 1$ so that $\frac{1+\delta}{1-2\delta} \leq 1+\varepsilon$. Thus, for such δ, the obtained l-consistent P-system Σ satisfies that $\rho(\Sigma) \leq (1+\varepsilon)\sigma(P)/2^k$ as desired.

4 2-CNF Formulas

In this section, we study structure of 2-CNF formulas, i.e., CNF formulas of clauses of sizes one and two. We first recall a well-known lemma about unsatisfiable formulas with clauses of sizes two which can be found, e.g., in [1]. If Φ is a 2-CNF formula with variables $x_1, \ldots, x_n$, then $G(\Phi)$ denotes the directed graph of order $2n$ whose vertices correspond to literals $x_1, \ldots, x_n$ and $\neg x_1, \ldots, \neg x_n$ and whose edge set is the following: For each clause $(a \vee b)$, $G(\varphi)$ contains an arc from the literal $\neg a$ to the literal b and an arc from $\neg b$ to a (note that both a and b represent literals, not variables). For each clause (a) (which can also be viewed as a clause $(a \vee a)$), we include an arc from the literal $\neg a$ to a.

Lemma 6. *Let Φ be a 2-CNF formula with variables $x_1, \ldots, x_n$. Then, the formula Φ is satisfiable if and only if $G(\Phi)$ contains no directed cycle through both the vertices x_i and $\neg x_i$ for any i, $1 \leq i \leq n$.*

An immediate corollary of Lemma 6 is the following:

Lemma 7. *Each minimal inconsistent set of clauses of a 2-CNF contains at most two clauses of size one.*

We now show that there exist extremal 2-CNF formulas in which each small inconsistent set of clauses contains two clauses of sizes one:

Lemma 8. *Let $2 \leq l \leq L$ be any two integers. For each $\varepsilon > 0$, there exists a 2-CNF l-consistent formula Φ with $\rho(\Phi) \leq \rho_l^{2\text{SAT}} + \varepsilon$ such that each inconsistent set of L clauses contains at least two clauses of size one. Moreover, Φ contains each single clause of size two at most once.*

Proof (sketch). Fix integers $l \geq 2$ and $L \geq l$ for the rest of the proof. Similarly, $\delta < 1$ is a positive real which will be chosen at the end of the proof. Fix an l-consistent formula Φ_0 with $\rho(\Phi_0) \leq \rho_l^{2\text{SAT}}(1+\delta)$. We now classify the variables contained in the formula Φ_0: The set A_1 is formed by variables x contained in a clause of size one; we can assume without loss of generality that each variable $x \in A_1$ appears as a positive literal in the clause of size one. The set A_i, $2 \leq i \leq \lfloor l/2 \rfloor$, consists of variables x which are not contained in any A_j, $1 \leq j \leq i-1$, and which are contained in a clause of the form $(\neg y \vee x)$ for $y \in A_{i-1}$. Since Φ is l-consistent, we can assume that all the occurrence of $x \in A_i$ in the clauses $(\neg y \vee x)$, $y \in A_{i-1}$, are positive: Otherwise, there would be a set of at most i clauses which force x to be true as well as a set of at most i clauses which force x to be false. The union of these two sets of clauses consists of at most $2i$ clauses and it is clearly inconsistent. Since $i \leq \lfloor l/2 \rfloor$, this is impossible. Finally, let A_0 be the set of the remaining variables of Φ.

Let w_{ij}, $w_{\bar{i}j}$ and $w_{\bar{i}\bar{j}}$ be the number (sum of the weights) of the clauses of the type $(x \vee y)$, $(\neg x \vee y)$ and $(\neg x \vee \neg y)$, respectively, where $x \in A_i$ and $y \in A_j$. Similarly, let w_1 be the number (sum of the weights) of the clauses of the type (x) where $x \in A_1$. We may assume that $w_1 > 0$. Otherwise, $\rho(\Phi_0) \geq 3/4$ and we can set Φ to be an L-consistent P-system Σ with $P(x,y) = (x \vee y)$ with $\rho(\Sigma) \leq 3/4 + \varepsilon$ constructed in Theorem 1. Finally, W denotes the sum of all w_1, w_{ij}, $w_{\bar{i}j}$ and $w_{\bar{i}\bar{j}}$ for $0 \leq i,j \leq \lfloor l/2 \rfloor$. By the definition of the sets $A_1, \ldots, A_{\lfloor l/2 \rfloor}$, $w_{\bar{i}j} = 0$ for all $1 \leq i,j \leq \lfloor l/2 \rfloor$ with $i+1 < j$. In addition, since Φ is l-consistent, $w_{\bar{i}\bar{j}} = 0$ for all $1 \leq i,j \leq \lfloor l/2 \rfloor$ with $i+j+1 \leq l$.

We now define W_p to be the maximum of the sum:

$$w_1 p_1 + \sum_{0 \leq i \leq j \leq \lfloor l/2 \rfloor} w_{ij}(p_i + p_j - p_i p_j) + w_{\bar{i}\bar{j}}(1 - p_i p_j) + \sum_{0 \leq i,j \leq \lfloor l/2 \rfloor} w_{\bar{i}j}(1 - p_i + p_i p_j) \tag{5}$$

where the maximum is taken over all $0 \leq p_0, \ldots, p_{\lfloor l/2 \rfloor} \leq 1$. Clearly, $W_p/W \leq \rho(\Phi_0)$: Consider the probabilities $p_0, \ldots, p_{\lfloor l/2 \rfloor}$ for which the maximum in the above expression is attained. If each of the variables of the set A_i, $0 \leq i \leq \lfloor l/2 \rfloor$,

is chosen to be true randomly and independently with the probability p_i, then the expected number (weight) of the satisfied clauses is W_p. Therefore, there is a truth assignment which satisfies at least this number of clauses and consequently $W_p/W \leq \rho(\Phi_0)$.

Let n be an integer which we fix later. Let X_i, $0 \leq i \leq \lfloor l/2 \rfloor$, be $\lfloor l/2 \rfloor + 1$ disjoint sets consisting of n variables each. We construct a 2-CNF formula Φ with variables $X_0 \cup \cdots \cup X_{\lfloor l/2 \rfloor}$. The formula Φ contains $n^{1/2L}$ copies of a clause (x) for each $x \in X_1$. The other clauses are included to the formula Φ randomly and independently as follows: The clauses $(x \vee y)$, $(\neg x \vee y)$ and $(\neg x \vee \neg y)$ where $x \in X_i$ and $y \in X_j$ with $i \neq j$ are included to Φ with the probabilities $w_{ij}n^{-1+1/2L}/w_1$, $w_{\bar{i}j}n^{-1+1/2L}/w_1$ and $w_{\bar{i}\bar{j}}n^{-1+1/2L}/w_1$, respectively. The clauses $(x \vee y)$, $(\neg x \vee y)$ and $(\neg x \vee \neg y)$ where $x, y \in X_i$ are included to Φ with the probabilities $2w_{ii}n^{-1+1/2L}/w_1$, $w_{\bar{i}i}n^{-1+1/2L}/w_1$ and $2w_{\bar{i}\bar{i}}n^{-1+1/2L}/w_1$, respectively.

It is possible to show using Chernoff's inequality that the number of clauses of Φ is at least $Wn^{1+1/2L}(1-\delta)/w_1$ and the number of clauses which can be simultaneously satisfied does not exceed $W_p n^{1+1/2L}(1+\delta)/w_1 + 3Wn^{1+1/2L}\delta/w_1$ with the probability which tends to 1 as n goes to infinity. In addition, the expected number of minimal inconsistent sets of at most L clauses containing zero or one clause of size one is at most the following: $2L(l+2)^L W_p^L n^{1/2}/w_1^L$. The proofs of these claims are left due to space limitations. By Markov's inequality, the probability that there are more than $4L(l+2)^L W_p^L n^{1/2}/w_1^L$ minimal inconsistent sets of at most L clauses with zero or one clause of size one is at most 1/2. Therefore, if n is sufficiently large (with respect to a previously fixed $\delta > 0$), with positive probability, the random formula Φ has at least $Wn^{1+1/2L}(1-\delta)/w_1$ clauses, at most $W_p n^{1+1/2L}(1+\delta)/w_1 + 3Wn^{1+1/2L}\delta/w_1$ clauses of Φ can be simultaneously satisfied and Φ contains at most $4L(l+2)^L W_p^L n^{1/2}/w_1^L$ inconsistent sets of at most L clauses with no or a single clause of size one. Fix such a formula Φ. We obtain Φ' from Φ by removing all (at most $4L^2(l+2)^L W_p^L n^{1/2}/w_1^L$) clauses of size two contained in an inconsistent set of at most L clauses with no or a single clause of size one. Note that the number of clauses which can be simultaneously satisfied cannot increase.

We now estimate $\rho(\Phi')$ (observe that $W_p \geq W/2$):

$$\rho(\Phi') \leq \frac{W_p n^{1+1/2L}(1+\delta)/w_1 + 3Wn^{1+1/2L}\delta/w_1}{Wn^{1+1/2L}(1-\delta)/w_1 - 4L^2(l+2)^L W_p^L n^{1/2}/w_1^L} =$$

$$\frac{W_p(1+\delta) + 3\delta W}{W(1-\delta) - 4L^2(l+2)^L W_p^L n^{-1/2-1/2L}/w_1^{L-1}} \leq \frac{W_p(1+7\delta)}{W(1-\delta) - O(n^{-1/2-1/2L})}.$$

Therefore, if n is sufficiently large, then:

$$\rho(\Phi') \leq \frac{W_p(1+7\delta)}{W(1-2\delta)} \leq \rho(\Phi_0)\frac{1+7\delta}{1-2\delta} \leq \rho_l^{2\mathrm{SAT}}\frac{(1+\delta)(1+7\delta)}{1-2\delta}.$$

Hence, for each $\varepsilon > 0$, we can choose $\delta > 0$ small enough that $\rho(\Phi') \leq \rho_l^{2\mathrm{SAT}} + \varepsilon$.

A straightforward proof that the constructed formula Φ' is l-consistent is left out due to space limitations.

A close inspection of the proof of Lemma 8 yields that for any weights w_1, w_{ij}, $w_{\bar{i}j}$ and $w_{i\bar{j}}$ with $w_{\bar{i}j} = 0$ for all $1 \leq i \leq j-1$ and $w_{i\bar{j}} = 0$ for all $1 \leq i, j$ with $i + j + 1 \leq l$. there is an l-consistent formula Φ with $\rho(\Phi) \leq W_p/W + \varepsilon$ where $W = w_1 + \sum_{i,j}(w_{ij} + w_{\bar{i}j} + w_{i\bar{j}})$ and W_p is the maximum of the sum (5) taken over all $0 \leq p_0, \ldots, p_{\lfloor l/2 \rfloor} \leq 1$. Therefore, we have the following formula for $\rho_l^{2\text{SAT}}$ for all $l \geq 2$:

Corollary 1. *For each $l \geq 2$, the following holds:*

$$\rho_l^{2\text{SAT}} = \min_{\substack{0 \leq w_1, w_{ij}, w_{\bar{i}j}, w_{i\bar{j}} \\ w_1 + \sum_{i,j}(w_{ij} + w_{\bar{i}j} + w_{i\bar{j}}) = 1}} W_p,$$

where the minimum is taken over all combinations of weights with $w_{\bar{i}j} = 0$ for all $1 \leq i \leq j-1$ and $w_{i\bar{j}} = 0$ for all $1 \leq i, j$ with $i + j + 1 \leq l$ and W_p is the maximum of the sum (5) taken over all $0 \leq p_0, \ldots, p_{\lfloor l/2 \rfloor} \leq 1$.

5 Unary, Binary, and Ternary Boolean Predicates

As noted in Section 2, it is enough to determine the values $\rho_l(P)$ for representatives of isomorphism classes of essentially unary, binary and ternary Boolean predicates. The case of 1-extendable Boolean predicates was handled in Theorem 1. The only essentially unary, binary and ternary Boolean predicates which are not 1-extendable (upto isomorphism) are the following: $P(x) = x$, $P(x, y) = x \wedge y$, $P(x, y, z) = x \wedge y \wedge z$, $P(x, y, z) = x \wedge (y \Leftrightarrow z)$ and $P(x, y, z) = x \wedge (y \vee z)$. The first three of these predicates satisfy that $\sigma(P) = 1$ and so the values $\rho_l(P)$ for these three predicates were determined in Lemma 3. Therefore, we know the values $\rho_l(P)$ for all essentially unary and binary Boolean predicates (see Tables 1 and 2). We focus on l-consistent P-systems with $P(x, y, z) = x \wedge (y \Leftrightarrow z)$ and $P(x, y, z) = x \wedge (y \vee z)$ in the rest of this section. We first need to handle the case of 2-consistent systems. Little technical proofs of Lemmas 9, 10 and 11 are left due to space limitations.

Lemma 9. *It holds that $\rho_2(P) = 8/27$ for $P(x, y, z) = x \wedge (y \Leftrightarrow z)$.*

Lemma 10. *It holds that $\rho_2(P) = 2\sqrt{3}/9$ for $P(x, y, z) = x \wedge (y \vee z)$.*

We can now analyze locally consistent P-systems for $P(x, y, z) = x \wedge (y \Leftrightarrow z)$:

Theorem 2. *If P is the predicate $P(x, y, z) = x \wedge (y \Leftrightarrow z)$, then the following holds for all $l \geq 1$:*

$$\rho_l(P) = \begin{cases} 1/4 & \text{if } l = 1, \\ 8/27 & \text{if } l = 2, \\ 1/2 & \text{otherwise.} \end{cases}$$

Before we analyze P-systems with $P(x, y, z) = x \wedge (y \vee z)$, we need to provide an upper bound for 3-consistent P-systems:

Lemma 11. *It holds that* $\rho_3(P) \leq 1/2$ *for* $P(x,y,z) = x \wedge (y \vee z)$.

We are now ready to determine the values $\rho_l(P)$ for the predicate $P(x,y,z) = x \wedge (y \vee z)$:

Theorem 3. *Let P be the predicate* $P(x,y,z) = x \wedge (y \vee z)$. *Then, the following holds for all* $l \geq 1$:

$$\rho_l(P) = \begin{cases} 3/8 & \text{if } l = 1, \\ 2\sqrt{3}/9 & \text{if } l = 2, \\ \rho_{l-2}^{\mathrm{2SAT}} & \text{otherwise.} \end{cases}$$

Proof. The equalities $\rho_1(P) = 3/8$ and $\rho_2(P) = 2\sqrt{3}/9$ follow from Lemmas 2 and 10, respectively. We first prove that $\rho_l(P) \geq \rho_{l-2}^{\mathrm{2SAT}}$ for $l \geq 3$. Let Σ be an l-consistent P-system and let X be the set of variables of Σ which appear as the first argument in some predicates of Σ. Since Σ is 2-consistent, we can assume that all the first arguments of the predicates of Σ are positive literals. Let Y be the set of the remaining variables of Σ.

We construct an auxiliary $(l-2)$-consistent 2-CNF formula Φ with the variables Y as follows. Since Σ is 3-consistent, it does not contain a predicate $P(x, \neg x', \neg x'')$ where $x', x'' \in X$. For each predicate $P(x, \neg x', y)$ and each predicate $P(x, y, \neg x')$ of Σ with $x, x' \in X$ and $y \in Y$, we include the clause (y) to Φ. Similarly, for each predicate $P(x, \neg x', \neg y)$ and each predicate $P(x, \neg y, \neg x')$ with $x' \in X$, we include the clause $(\neg y)$. For each predicate $P(x, y, y')$ with $x \in X$ and $y, y' \in Y$, we include the clause $(y \vee y')$ to Φ. We proceed analogously for predicates $P(x, \neg y, y')$, $P(x, y, \neg y')$ and $P(x, \neg y, \neg y')$. Note that some of the clauses may be contained in the formula Φ several times.

We claim that the formula Φ is $(l-2)$-consistent. If this is not the case, let Γ be the minimum inconsistent set of clauses of Φ. By Lemma 7, Γ contains at most two clauses of size one. We now find an inconsistent set Γ' of at most $|\Gamma|+2$ predicates of Σ. For each clause of Γ of size two, include to Γ' the predicate of Σ corresponding to that clause. For each clause (y), $(\neg y)$, of Γ, include to Γ' the predicate $P(x, y, \neg x')$, $P(x, \neg y, \neg x')$, respectively, which corresponds to that clause, together with any of the predicates of Σ whose first argument is x'. Since Γ contains at most two clauses of size one, $|\Gamma'| \leq |\Gamma| + 2 \leq l$. Moreover, since Γ is inconsistent, Γ' is also inconsistent. However, this contradicts the fact that Σ is l-consistent.

Since the formula Φ is $(l-2)$-consistent, there is a truth assignment which satisfies the fraction of $\rho(\Phi) \geq \rho_{l-2}^{\mathrm{2SAT}}$ clauses of Φ. Extend this truth assignment to all the variables of Σ by assigning the true value to each variable $x \in X$. All the predicates of Σ whose arguments contain solely the variables from the set X are satisfied and, in addition, the fraction of $\rho(\Phi)$ of the remaining predicates are also satisfied. Therefore, $\rho(\Sigma) \geq \rho(\Phi) \geq \rho_{l-2}^{\mathrm{2SAT}}$. Since the choice of a P-system Σ was arbitrary, we can conclude that $\rho_l(P) \geq \rho_{l-2}^{\mathrm{2SAT}}$.

It remains to prove that $\rho_l(P) \leq \rho_{l-2}^{\mathrm{2SAT}}$ for $l \geq 3$. If $l = 3$, the upper bound follows from Lemma 11. For $l \geq 4$, choose $\varepsilon > 0$ and fix an $(l-2)$-consistent 2-CNF formula Φ with $\rho(\Phi) \leq \rho_{l-2}^{\mathrm{2SAT}} + \varepsilon$ such that each minimal inconsistent set of at most l clauses contain two clauses of size one. Such a formula Φ exists

by Lemma 8. Moreover, we can assume that each clause of size two is contained in Φ at most once. Let m' be the number of clauses of Φ of size one (counting multiplicities) and m the number of all clauses of Φ. Since Φ is 2-consistent, $m'/m \leq \rho(\Phi)$. We now construct an *l*-consistent *P*-system Σ with $\rho(\Sigma) = \rho(\Phi)$.

Let $y_1, \ldots, y_n$ be the set consisting of the variables of the formula Φ. The system Σ will contain $(m+1)n$ variables y_i^j for $1 \leq i \leq n$ and $1 \leq j \leq m+1$ and $m+1$ variables x^j for $1 \leq j \leq m+1$. Let $C_1, \ldots, C_m$ be the clauses of Φ. For each clause $C_k = (y_i \vee y_{i'})$, $1 \leq k \leq m$, we include to Σ predicates $P(x^j, y_i^j, y_{i'}^j)$ for $1 \leq j \leq m+1$. Similarly, we proceed for clauses $C_k = (y_i \vee \neg y_{i'})$ and $C_k = (\neg y_i \vee \neg y_{i'})$. If the clause C_k is of size one, say $C_k = (y_i)$, we include to Σ predicates $P(x^j, y_i^j, \neg x^{(j+k) \bmod (m+1)})$ for $1 \leq j \leq m+1$. Therefore, Σ consists of $m(m+1)$ distinct predicates.

The constructed *P*-system Σ is *l*-consistent (a straightforward proof of this fact is left due to space limitations). We now show that $\rho(\Sigma) = \rho(\Phi)$. Since $\rho(\Phi) \leq \rho_{l-2}^{2\mathrm{SAT}} + \varepsilon$ and the choice of ε was arbitrary, this would yield $\rho_l(P) \leq \rho_{l-2}^{2\mathrm{SAT}}$. Fix a truth assignment such that the fraction of $\rho(\Sigma)$ predicates of the *P*-system Σ is satisfied. We claim that there is an optimum truth assignment which assigns all the variables $x^1, \ldots, x^{m+1}$ the true value. Indeed, if x^j is false, then change the value of x^j to true. This causes at most m' previously satisfied predicates to be unsatisfied (precisely those which contain $\neg x^j$ as the third argument) and, on the other hand, we can choose values of $y_1^j, \ldots, y_n^j$ so that at least the $\rho(\Phi)m$ predicates whose first argument is x^j are satisfied. Note that none of these $\rho(\Phi)m$ predicates could be satisfied before the change of the value of x^j. Since $\rho(\Phi)m \geq m'$ (recall that $\rho(\Phi) \geq m'/m$), the number of satisfied predicates is not decreased after the change. In this way, we can switch all the variables $x^1, \ldots, x^{m+1}$ to true without decreasing the number of satisfied constraints. Hence, we can assume that all the variables $x^1, \ldots, x^{m+1}$ are set to be true by the considered optimum truth assignment. Then, the system Σ is reduced to $m+1$ independent "copies" of the formula Φ (substitute the true value for all the variables $x^1, \ldots, x^{m+1}$). We can conclude that $\rho(\Sigma) = \rho(\Phi)$.

6 Conclusion

We studied instances of constraint satisfaction problems which are locally consistent. There are several directions for possible future research. First, we were not able to fully analyze Boolean predicates which are not 1-extendable. The smallest two non-trivial such Boolean predicates, $P(x,y,z) = x \wedge (y \Leftrightarrow z)$ and $P(x,y,z) = x \wedge (y \vee z)$, already showed that the behavior of locally consistent *P*-systems for such predicates *P* can be quite weird. Another direction is to allow constraints of more types: In this setting, the previously most studied case of locally consistent CNF formulas can be viewed as a constraint satisfaction problem where constraints are just disjunctions, e.g., the case of 2-CNF formulas corresponds to problems with the constraints $P(x) = x$ and $P(x,y) = x \vee y$. The last possible direction is to consider constraints with larger domains. Some of our results can be easily translated to this more general setting, e.g., Theorem 1,

on the other hand, their detailed analysis even for small arities might be quite difficult because of their potentially rich structure.

Acknowledgement. The authors would like to thank Gerhard Woeginger for attracting their attention to locally consistent formulas and for pointing out several useful references. They would also like to thank Dimitrios M. Thilikos for suggesting the version of the problem considered in this paper.

References

1. S. Cook: The Complexity of Theorem-proving Procedures. In: Proc. of the 3rd ACM Symposium on Theory of Computing. ACM, New York (1971) 29–33.
2. S. Cook, D. Mitchell: Finding Hard Instances of the Satisfiability Problem: A Survey. In: Satisfiability Problem: Theory and Applications. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 35 AMS (1997).
3. D. Eppstein: Improved Algorithms for 3-coloring, 3-edge-coloring and Constraint Satisfaction. In: Proc. of the 12th ACM-SIAM Symposium on Discrete Algorithms. SIAM (2001) 329–337.
4. T. Feder, R. Motwani: Worst-case Time Bounds for Coloring and Satisfiability Problems. J. Algorithms 45(2) (2002) 192-201.
5. T. Hagerup, Ch. Rüb: A guided tour Chernoff bounds. Inform. Process. Letters 33 (1989) 305–308.
6. M. A. Huang, K. Lieberherr: Implications of Forbidden Structures for Extremal Algorithmic Problems. Theoretical Computer Science 40 (1985) 195–210.
7. S. Jukna: Extremal Combinatorics with Applications in Computer Science. Springer, Heidelberg (2001).
8. D. Král': Locally Satisfiable Formulas. In: Proc. of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM (2004) 323-332.
9. K. Lieberherr, E. Specker: Complexity of Partial Satisfaction. J. of the ACM, 28(2) (1981) 411–422.
10. K. Lieberherr, E. Specker: Complexity of Partial Satisfaction II. Technical Report 293, Dept. of EECS, Princeton University (1982).
11. L. Trevisan: On Local versus Global Satisfiability. SIAM J. Disc. Math. (to appear). A preliminary version is available as ECCC report TR97-12.
12. Z. Usiskin: Max-min Probabilities in the Voting Paradox. Ann. Math. Stat. 35 (1963) 857–862.
13. G. J. Woeginger: Exact Algorithms for NP-hard Problems: A Survey. In: Proc. 5th Int. Worksh. Combinatorial Optimization - Eureka, You Shrink. Lecture Notes in Computer Science, Vol. 2570. Springer-Verlag Berlin (2003) 185-207.
14. M. Yannakakis: On the Approximation of Maximum Satisfiability. J. Algorithms 17 (1994) 475–502. A preliminary version appeared in: Proc. of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM (1992) 1–9.

Quantum Query Complexity of Some Graph Problems*

Christoph Dürr[1]**, Mark Heiligman[2], Peter Høyer[3]***, and Mehdi Mhalla[4]

[1] Laboratoire de Recherche en Informatique, UMR 8623,
Université Paris-Sud, 91405 Orsay, France
durr@lri.fr
[2] Advanced Research and Development Activity, Suite 6644,
National Security Agency, 9800 Savage Road,
Fort Meade, Maryland 20755, USA
miheili@nsa.gov
[3] Dept. of Computer Science, Univ. of Calgary, Alberta, Canada
hoyer@cpsc.ucalgary.ca
[4] Laboratoire Leibniz, Institut IMAG, Grenoble, France
Mehdi.Mhalla@imag.fr

Abstract. Quantum algorithms for graph problems are considered, both in the adjacency matrix model and in an adjacency list-like array model. We give almost tight lower and upper bounds for the bounded error quantum query complexity of CONNECTIVITY, STRONG CONNECTIVITY, MINIMUM SPANNING TREE, and SINGLE SOURCE SHORTEST PATHS. For example we show that the query complexity of MINIMUM SPANNING TREE is in $\Theta(n^{3/2})$ in the matrix model and in $\Theta(\sqrt{nm})$ in the array model, while the complexity of CONNECTIVITY is also in $\Theta(n^{3/2})$ in the matrix model, but in $\Theta(n)$ in the array model. The upper bounds utilize search procedures for finding minima of functions under various conditions.

1 Introduction

A primary goal of the theory of quantum complexity is to determine when quantum computers may offer a computational speed-up over classical computers. Today there are only a few results which give a polynomial time quantum algorithm for some problem for which no classical polynomial time solution is known. We are interested in studying the potentialities for speed-up for problems for

* This paper subsumes manuscripts on arxiv.org quant-ph/9607014, quant-ph/0303131, quant-ph/0303169. We are grateful to Yaohui Lei for his permission to include results presented in quant-ph/0303169 in this paper.
** Research partially supported by the EU fifth framework program RESQ IST-2001-37559, and RAND-APX IST-1999-14036, by CNRS/STIC 01N80/0502 grant, by ACI Cryptologie CR/02 02 0040 grant of the French Research Ministry.
*** Supported in part by the Alberta Ingenuity Fund and the Pacific Institute for the Mathematical Sciences.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 481–493, 2004.

which there already are efficient classical algorithms. Basic graphs problems are interesting candidates.

We study the query complexity of these problems; meaning the minimal number of queries to the graph required for solving the problem. Throughout this paper, the symbol $[n]$ denotes the set $[0..n-1]$. We consider two query models for directed graphs:

The adjacency matrix model, where the graph is given as the adjacency matrix $M \in \{0,1\}^{n \times n}$, with $M_{ij} = 1$ if and only if $(v_i, v_j) \in E$.

The adjacency array model, where we are given the out-degrees of the vertices $d_1^+, \ldots, d_n^+$ and for every vertex u an array with its neighbors $f_i : [d_i^+] \to [n]$. So $f_i(j)$ returns the j^{th} neighbor of vertex i, according to some arbitrary but fixed numbering of the outgoing edges of i. In this paper the upper bounds for this model are all at least n, so we assume henceforth that the degrees are given as part of the input and we account only queries to the arrays f_i. We assume that f_i is injective, ensuring the graph is not a multigraph.

For undirected graphs we require an additional promise on the input, namely that M is symmetric in the matrix model, and for the array model that $\forall i, i' \in [n]$ if $\exists j \in [k] : f_i(j) = i'$ then $\exists j' \in [k] : f_{i'}(j') = i$. Note that in the matrix model this symmetry assumption does not make undirected graph problems, promise problems since we may assume that the input is upper triangular.

Weighted graphs are encoded by a weight matrix, where for convenience we set $M_{ij} = \infty$ if $(v_i, v_j) \notin E$. In the adjacency array model, the graph is encoded by a sequence of functions $f_i : [d_i^+] \to [n] \times \mathbb{N}$, such that if $f_i(j) = (i', w)$ then there is an edge $(v_i, v_{i'})$ and it has weight w.

We emphasize that the array model is different from the standard list model. In the latter, we have access to the neighbors of a given vertex only as a list, and thus querying the i^{th} neighbor requires i accesses to the list. This is also true on a quantum computer, which motivates the array model.

Many other query models are of course possible, for example we could be given an array of edges $f : [m] \to [n] \times [n]$, or an ordered array (which is up to $O(n)$ preprocessing the same as the adjacency array model). For simplicity, we use the array model as presented above.

For the quantum query complexity of general monotone graph properties, a lower bound of $\Omega(\sqrt{n})$ is known in the matrix model, as shown by Buhrman, Cleve, de Wolf and Zalka [9].[1] We are not aware of any quantum nor classical lower bounds in the array model.

In this paper we show that the quantum query complexity of CONNECTIVITY is $\Theta(n^{3/2})$ in the matrix model and $\Theta(n)$ in the array model. The classical randomized query complexity of CONNECTIVITY in the matrix model is $\Omega(n^2)$

[1] In a previous version of this paper, we said that Buhrman et al. conjectured $\Omega(n)$ for CONNECTIVITY, and since their conjecture concerns arbitrary monotone graph properties in general, we gave a false impression of improving their result. We apologize.

by a sensitivity argument: Distinguishing the graph consisting of two length $n/2$ paths from the graph consisting of those two paths, plus an additional edge connecting them, $\Omega(n^2)$ queries are required.

We study the complexity of three other problems. In STRONG CONNECTIVITY we are given a directed graph and have to decide if there is a directed path between any pair of vertices. In MINIMUM SPANNING TREE we are given a weighted graph and have to compute a spanning tree with minimal total edge weight. In SINGLE SOURCE SHORTEST PATHS we have to compute the shortest paths from a given source vertex to every other vertex. The quantum query complexity of these three problems is $\Omega(n^{3/2})$ in the matrix model and $\Omega(\sqrt{nm})$ in the array model. We give almost tight upper bounds.

Table 1. Quantum query complexity of some graph problems

problem	matrix model	array model
minimum spanning tree	$\Theta(n^{3/2})$	$\Theta(\sqrt{nm})$
connectivity	$\Theta(n^{3/2})$	$\Theta(n)$
strong connectivity	$\Theta(n^{3/2})$	$\Omega(\sqrt{nm})$, $O(\sqrt{nm\log n})$
single src. short. paths	$\Omega(n^{3/2})$, $O(n^{3/2}\log^2 n)$	$\Omega(\sqrt{nm})$, $O(\sqrt{nm}\log^2 n)$

We note that for graphs with a large number of edges ($m = \Theta(n^2)$), the complexities are (almost) the same in the matrix and array model for all problems but CONNECTIVITY. However, the models still differ in that case. For example the test $(u, v) \in E$ costs a single query in the matrix model and $O(\sqrt{\min\{d_u^+, d_v^+\}})$ queries in the array model since we do not assume any order on the arrays f_u and f_v.

The time complexities of the algorithms are the same as their query complexities, up to a $\log n$ factor in the bit computational model. The algorithms given for connectivity and strong connectivity can be altered to also output the (strongly) connected components without increasing the asymptotic complexity. The space requirement is $O(\log n)$ qubits and $O(n \log n)$ classical bits. If we constraint the space (both classical and quantum) to $O(\log n)$ qubits, the problems may be solved by random walks. Quantum random walks has been the subject of several papers [1,10,15], in particular for the *st*-CONNECTIVITY problem [22].

2 Quantum Search

The quantum ingredient to our algorithms is amplitude amplification [7,8]. It is a generalization of Lov Grover's search algorithm [13]. Since it is the most important tool used in our algorithms, we restate the exact results we require. We are given a boolean function F defined on a domain of size n. The function is given as a black box so that the only way we can obtain information about F is via evaluating F on elements in the domain. The search problem considered by Grover is to find an element x for which $F(x) = 1$, provided one exists. We

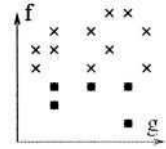
say that x is a *solution* to the search problem, and that x is *good*. We use three generalizations of the search algorithm—all of which we refer to as "the search algorithm".

- If there are t elements mapped to 1 under F, with $t > 0$, the search algorithm returns a solution after an expected number of at most $\frac{9}{2}\sqrt{n/t}$ queries to F. The output of the algorithm is chosen uniformly at random among the t solutions. The algorithm does not require prior knowledge of t [6].
- A second version uses $O(\sqrt{n})$ queries to F in the worst case and outputs a solution with probability at least a constant, provided there is one [6].
- A third version uses $O(\sqrt{n \log 1/\epsilon})$ queries to F and finds a solution with probability at least $1 - \epsilon$, provided there is one [9].

3 Minima Finding

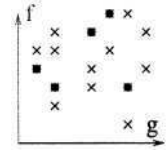
Many graph problems are optimization problems, as are finding a minimum spanning tree, single source shortest paths, and largest connected components. Most quantum algorithms for such optimization problems utilize the search algorithm discussed above. A very basic and abstract optimization problem is as follows. Suppose we are given a function f defined on a domain of size n, and we want to find an index i so that $f(i)$ is a minimum in the image of f. This minimization problem was considered in [11] which gives an optimal quantum algorithm that uses $O(\sqrt{n})$ queries to f and finds such an i with constant probability. For the purposes of this paper, we require the following generalizations.

Problem 1 (Find d smallest values of a function). Let $\mathbb{N}^*$ denote $\mathbb{N} \cup \{\infty\}$. Given function $f : [N] \to \mathbb{N}^*$ and an integer $d \in [N]$, we wish to find d distinct indices mapping to smallest values, i.e. a subset $I \subseteq [N]$ of cardinality d such that for any $j \in [N] \setminus I$ we have that $f(i) \leq f(j)$ for all $i \in I$.



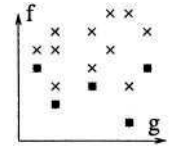
In the rest of this section, we assume $d \leq N/2$. In the following problem we are given a different function $g : [N] \to \mathbb{N}$, such that $g(j)$ defines the *type* of j. Let $e = |\{g(j) : j \in [N]\}|$ be the number of different types.

Problem 2 (Find d elements of different type). Given function g and an integer d' we wish to find integer $d = \min\{d', e\}$ and a subset $I \subseteq [N]$ of cardinality d such that $g(i) \neq g(i')$ for all distinct $i, i' \in I$.



Now we present a generalization of both previous problems.

Problem 3 (Find d smallest values of different type). Given two functions f, g and an integer d' we wish to find integer $d = \min\{d', e\}$ and a subset $I \subseteq [N]$ of cardinality d such that $g(i) \neq g(i')$ for all distinct $i, i' \in I$ and such that for all $j \in [N] \setminus I$ and $i \in I$, if $f(j) < f(i)$ then $f(i') \leq f(j)$ for some $i' \in I$ with $g(i') = g(j)$.



It is clear that Problems 1 and 2 are special cases of Problem 3. In this section, we give an upper bound of $O(\sqrt{dN})$ for Problem 3. In Section 7, we then show a lower bound of $\Omega(\sqrt{dN})$ for Problems 1 and 2, implying that all three problems are of complexity $\Theta(\sqrt{dN})$. We prove the upper bound by a simple greedy algorithm. Consider a subset $I \subseteq [N]$ of d indices of different types. We say an index $j \in [N]$ is *good for I* if

1. either $g(j) = g(i)$ and $f(j) < f(i)$ for some $i \in I$,
2. or $g(j) \notin g(I)$ and $f(j) < f(i)$ for some $i \in I$.

In the former case we say j is a good index of *known type,* in the latter that j is a good index of *unknown type.* In each iteration of the greedy algorithm, we find a good index j by the search algorithm and then improve I by replacing some index in I by j.

1. Initially, let $I = \{N+1, \ldots, N+d'\}$ be a set of fictitious indices of unique different types and unique maximal value.
2. Repeat forever
   a) Let t denote the number of good elements for I. (Note: This step is not required, but only included for the purpose of simplifying the analysis of the algorithm.)
   b) Use the first version of the search algorithm to find a good element $j \in [N]$ for I.
   c) Set $I = \text{improve}(I, j)$ where we improve I by replacing with j the element in I that has the same type as j if j is of known type, and by replacing with j some element in I with largest f-value if j is of unknown type.

The next lemma shows we only need an expected number of $O(d)$ iterations of the main loop to eliminate a constant fraction of the remaining good elements.

Lemma 1. *Let $I \subseteq [N]$ be any subset of d' indices of different types with $t > 0$ good elements of e types. After an expected number of $O(d)$ iterations of the main loop there are at most $\frac{3}{4}t$ good elements for I. Here $d = \min\{d', e\}$.*

Proof. For notational simplicity assume f is injective. Set $I_0 = I$ and let $T_0 = T$ be the set of good elements for I. Let T_j denote the set of good elements after j iterations of the main loop, for $j > 0$. Similarly, let I_j denote the selected index-set after j iterations, for $j > 0$. Set $t_k = |T_k|$. In particular $I_0 = I$ and $t_0 = t$. Let y_{mid} denote the $\lfloor t/2 \rfloor^{\text{th}}$ smallest of the t elements according to f. For any subset $S \subseteq [N + d']$, let $\text{low}(S)$ denote the number of elements in S that are no bigger than y_{mid} according to f.

Note that initially

- $\text{low}(T_0) = \lfloor t/2 \rfloor$ and
- $\text{low}(I_0) < d$.

By the nature of the greedy algorithm, $\text{low}(T_{k+1}) \leq \text{low}(T_k)$ and $\text{low}(I_{k+1}) \geq \text{low}(I_k)$ for any $k \geq 0$. Note that

- if $\text{low}(T_k) < \frac{t}{4}$, then we have eliminated at least a fraction of $\frac{1}{4}$ of the initially t good elements for I, and similarly,
- if $\text{low}(I_k) = d$, then we have eliminated at least a fraction of $\frac{1}{2}$ of the initially t good elements for I.

We claim that in each iteration of the main loop, as long as $\text{low}(T_k) \geq \frac{t}{4}$, with probability at least $\frac{1}{32}$, at least one of the following two events happens

- $\text{low}(T_{k+1}) \leq \text{low}(T_k)\left(1 - \frac{1}{32d}\right)$,
- $\text{low}(I_{k+1}) = \text{low}(I_k) + 1$.

Assume $\text{low}(T_k) \geq \frac{t}{4}$, since otherwise we are done. Consider the element j picked in Step 2b. First suppose the majority of the $\text{low}(T_k)$ indices are of unknown type with respect to I_k. Then, with probability at least $\frac{1}{8}$, index j is among these, in which case $\text{low}(I_{k+1}) = \text{low}(I_k) + 1$.

Now suppose the majority of the $\text{low}(T_k)$ indices are of known type with respect to I_k. Then, with probability at least $\frac{1}{8}$, index j is among these. Conditioned on this happens, with probability at least $\frac{1}{2}$, there are at least $\frac{\text{low}(T_k)}{4d}$ good elements for I_k of the same type as j. With probability at least $\frac{1}{2}$, at least half of these are not good for I_{k+1}. Thus, with probability at least $\frac{1}{32}$, we have eliminated at least $\frac{t}{32d}$ of the remaining elements in T_j.

This proves the claim. It follows that after an expected number of $O(d)$ iterations of the main loop, we have eliminated at least a fraction of $\frac{1}{4}$ of the initially t good elements. □

The above lemma implies that, for $t > 2d$, after an expected number of $O(d\sqrt{N/t})$ applications of function f, the number of good elements is at most $\frac{t}{2}$. Hence, for any $t > 2d$, the expected number applications of function f required till we have that $t \leq 2d$ for the first time is in the order of

$$d\left(\sqrt{\frac{N}{d}} + \sqrt{\frac{N}{2d}} + \sqrt{\frac{N}{4d}} + \sqrt{\frac{N}{8d}} + \cdots\right) \in O(\sqrt{dN}).$$

Once $t \leq 2d$ for the first time, the expected number of applications of f required before $t = 0$ for the first time is in the order of $\sum_{j=1}^{2d} \sqrt{N/j}$ which is in $O(\sqrt{dN})$.

Corollary 1. *In the greedy algorithm given above, after an expected number of* $O(\sqrt{dN})$ *applications of function* f*, there are no good elements for I, that is,* $t = 0$.

The next theorem follows immediately.

Theorem 1. *The problem* FIND d SMALLEST VALUES OF DIFFERENT TYPE *has bounded error quantum query complexity* $O(\sqrt{dN})$.

We would like to mention that this implies a bounded error algorithm for finding the element of rank d in a table of size N using $O(\sqrt{dN})$ queries. It is of different nature than an algorithm given by Nayak and Wu [19], and later improved by Nayak [18].

4 Minimum Spanning Tree

In this section we consider undirected graphs with weighted edges. In MINIMUM SPANNING TREE we wish to compute a cycle-free edge set of maximal cardinality that has minimum total weight. To be precise if the graph is not connected this is actually a spanning forest.

Classically, there are a number of different approaches to finding minimum spanning trees efficiently, including the algorithms of Borůvka [5,20], Kruskal [17], and Prim [21]. To construct an efficient quantum algorithm, we use Borůvka's algorithm since it is of a highly parallel nature. This allows us to use the minima finding algorithms given in Section 3.

Borůvka's algorithm consists of at most $\log n$ iterations. In brief, initially it starts with a collection of n spanning trees, each tree containing a single vertex. In each iteration, it finds a minimum weight edge out of each tree in the collection, adds the edges to the trees, and merges them into larger and fewer trees. After at most $\log n$ iterations, there is only one tree left, which is a minimum spanning tree. The correctness of Borůvka's algorithm rests on the following simple fact about spanning trees.

Fact 2 *Let $U \subset V$ be a set of vertices of a connected graph $G = (V, E)$ and let e be a minimum weight edge of $(U \times \overline{U}) \cap E$. Then there is a minimum spanning tree containing e.*

In our quantum version of Borůvka's algorithm, we make a few adjustments to keep the overall error probability small without sacrificing in the number of queries. We adjust it slightly so that the ℓ^{th} iteration errs with probability at most $\frac{1}{2^{\ell+2}}$, ensuring that the overall error is at most $\frac{1}{4}$. This increases the cost of the ℓ^{th} iteration by a factor of $O(\sqrt{\ell})$, but since the cost of the first few iterations dominates, this is asymptotically negligible. The details follow.

1. Let $T_1, T_2, \ldots, T_k$ be a spanning forest. Initially, $k = n$ and each tree T_j contains a single vertex.
2. Set $\ell = 0$.
3. Repeat until there is only a single spanning tree (i.e., $k = 1$).
   a) Increment ℓ.
   b) Find edges $e_1, e_2, \ldots, e_k$ satisfying that e_j is a minimum weight edge leaving T_j. Interrupt when the total number of queries is $c\sqrt{km(\ell+2)}$ for some appropriate constant c.
   c) Add the edges e'_j to the trees, merging them into larger trees.
4. Return the spanning tree T_1.

To find the minimum edges $e_1, \ldots, e_k$ in Step 3b, we use the following functions. In the array model, any edge (u, v) is coded twice, u appears as neighbor of v, but v also appears as neighbor of u. Enumerate the directed edges from 0 to $2m-1$. Let $f : [2m] \to \mathbb{N}^*$ denote the function that maps every directed edge (u, v) to its weight if u and v belong to different trees of the current spanning

forest and to ∞ otherwise. Let $g : [2m] \to [k]$ denote the function that maps every directed edge (u, v) to the index j of the tree T_j containing u. We may then apply the algorithm for FINDING k SMALLEST VALUES OF DIFFERENT TYPE. Interrupting this algorithm after $c\sqrt{km}$ queries introduces an error probability at most 1/2. To reduce it to $1/2^{\ell+2}$ we could simply repeat it $\ell + 2$ times and keep only the minimal values among the $\ell + 2$ outputs, or even simpler and better, run the algorithm only once and interrupt it after $(\ell + 2)c\sqrt{km}$ queries.

Theorem 3. *Given an undirected graph with weighted edges, the algorithm above outputs a spanning tree that is minimum with probability at least* $\frac{1}{4}$. *The algorithm uses* $O(\sqrt{nm})$ *queries in the array model and* $O(n^{3/2})$ *queries in the matrix model.*

Proof. To simplify the proof, consider the matrix model an instance of the array model with $m = n(n-1)$ edges.

At the beginning of the ℓ^{th} iteration of the main loop, the number of trees k is at most $n/2^{\ell-1}$, and thus it uses at most $c\sqrt{nm(\ell+2)/2^{\ell-1}}$ queries. Summing over all iterations, the total number of queries is at most $\sum_{\ell\geq 1} c\sqrt{nm(\ell+2)/2^{\ell-1}}$, which is in $O(\sqrt{nm})$.

The ℓ^{th} iteration introduces an error with probability at most $\frac{1}{2^{\ell+2}}$. The overall error probability is thus upper bounded by $\sum_{\ell\geq 1} \frac{1}{2^{\ell+2}} \leq \frac{1}{4}$. □

Apart of an additional $\log n$ factor in the bit computational model, the time complexity is the same as the query complexity by using an appropriate data structure. Each vertex holds a pointer to another vertex in the same component, except for a unique vertex per component that holds a null pointer. This vertex is called the canonical representative of the component. To decide if two vertices are in the same component, we need only to determine the canonical representative of each vertex by pointer chasing. To merge two components, we change the pointer of one of the canonical representative to point to the other.

Using pointer chasing, the time complexity of the ℓ^{th} iteration is a factor of ℓ larger than its query complexity. However, as in the case of the error reduction, this is insignificant: $\sum_{\ell\geq 1} \ell c\sqrt{nm(\ell+2)/2^{\ell-1}}$ is also in $O(\sqrt{nm})$. Thus in the algebraic computational model, the time complexity is asymptotically the same as the query complexity.

5 Connectivity and Strong Connectivity

A special case of MINIMUM SPANNING TREE when all edge weights are equal, is GRAPH CONNECTIVITY. The input is an *undirected* graph and the output is a spanning tree, provided the graph is connected.

Surprisingly for the array model, there is a quantum algorithm that uses only $O(n)$ queries. It starts with a collection of connected components which were obtained classically, and which have the property that the total degree of each is not too large. Then it runs simply repeatedly the quantum search procedure to find edges connecting two components. We omit the proofs here.

Lemma 2. *Given an undirected graph G in the array model, we can in $O(n)$ classical queries construct a set of connected components $\{C_1, \ldots, C_k\}$ for some integer k, so that for each component C, its total degree $m_C = \sum_{i \in C} d_i$ is no more than $|C|^2$.*

Theorem 4. *There is a quantum algorithm which given an undirected graph G in the array model outputs after $O(n)$ queries with probability at least $\frac{9}{10}$ a spanning tree for G, provided G is connected.*

For directed graphs, we proceed slightly differently. Assume the input graph is strong connected. First with bounded error we construct a depth-first tree rooted in some arbitrary vertex, with edges oriented towards the leaves, using $O(n^{3/2}\sqrt{\log n})$ queries in the matrix model and $O(\sqrt{nm \log n})$ queries in the array model.

For the matrix model we then compute another depth-first tree with the same root but edges oriented towards the root, by applying the same algorithm on the transposed adjacency matrix. For the array model, we number the vertices in order of depth-first traversal, and search for every node, except the root, the edge with minimal index target. Adding these edges to the depth-first tree make it strong connected. Again, we omit the proofs here.

Theorem 5. *The bounded error quantum query complexity of* STRONG CONNECTIVITY *is $O(n^{3/2}\sqrt{\log n})$ in the matrix model and $O(\sqrt{nm \log n})$ in the array model.*

6 Single Source Shortest Paths

Let G be a directed graph with non-negative edge weights and a fixed vertex v_0. We want to compute for every vertex v a shortest path from v_0 to v. It may happen that it is not unique. Using for example the lexicographical ordering on vertex sequences, we choose to compute a single canonical shortest path. From now on assume that different paths have different lengths. As a result, the union over all vertices v of the shortest paths from v_0 to v is a *shortest path tree*. Let $\nu(u, v)$ be the weight of edge (u, v) and $\delta(v_0, v)$ the shortest path length from v_0 to v.

Classically SINGLE SOURCE SHORTEST PATH may be solved by Dijkstra's algorithm. It maintains a subtree $\mathcal{T}$ with the "shortest path subtree" invariant: for any vertex $v \in \mathcal{T}$, the shortest path from v_0 to v uses only vertices from $\mathcal{T}$. An edge (u, v) is called a *border edge* (of $\mathcal{T}$) if $u \in \mathcal{T}$ and $v \notin \mathcal{T}$, and u is called the source vertex, v the target vertex. The *cost* of (u, v) is $\delta(v_0, u) + \nu(u, v)$. Dijkstra's algorithm starts with $\mathcal{T} = \{v_0\}$ and iteratively adds the cheapest border edge to it.

Our improvement lays in the selection of the cheapest border edge. We give the algorithm for the array model. Setting $m = n^2$ implies the required bound for the matrix model.

Theorem 6. *The bounded error query complexity of single source shortest path in the array model is* $O(\sqrt{nm}\log^2 n)$.

Proof. As in Dijkstra's algorithm we construct iteratively a tree T, such that for every vertex $v \in T$, the shortest path from v_0 to v is in T. We also maintain a partition of the vertices covered by T, into a set sequence. Its length is denoted by l.

1. $T = \{v_0\}$, $l = 1$, $P_1 = \{v_0\}$
2. Repeat until T covers the graph
   a) For P_l compute up to $|P_l|$ cheapest border edges with disjoint target vertices. For this purpose set $N = \sum_{v \in P_l} d(v)$, and number all edges with source in P_l from 1 to N. Define the functions $f : [N] \to \mathbb{N}^*$ and $g : [N] \to V$, where $g(i)$ is target vertex of the i^{th} edge and $f(i)$ is its weight if $g(i) \notin T$ and ∞ otherwise. Apply the algorithm of section 3 on f and g with $d = |P_l|$ to find the d lowest cost edges with distinct target vertices. Let A_l be the resulting edge set.
   b) Let (u, v) be the minimal weighted edge of $A_1 \cup \ldots \cup A_l$ with $v \notin P_1 \cup \ldots P_l$. Set $T = T \cup \{(u, v)\}$, $P_{l+1} = \{v\}$ and $l = l + 1$.
   c) As long as $l \geq 2$ and $|P_{l-1}| = |P_l|$, merge P_l into P_{l-1}, and set $l = l - 1$.

All steps but 2(b) constructed a vertex set sequence $P_1, \ldots, P_l$, the cardinality of each being a power of 2, and of strictly decreasing sizes. Therefore each set P_i is strictly larger than the union of all the following sets, since $\sum_{i=0}^{k-1} 2^i = 2^k - 1$. If A_i contained $|P_i|$ edges, than at least one of them has its target vertex outside of $P_1 \ldots, P_l$. Let (u, v) be the cheapest border edge of T. Let P_i be the vertex set containing u. Then A_i must contain this edge, and step 2(b) selects it.

Only step 2(a) generates queries to the graph. What is the total number of queries related to sets P_i of some size s. There are at most n/s sets of this size s. Therefore total work is order of $\sum_{j=1}^{n/s} \sqrt{sm_j}$, where m_j is the number of edges with source in the j^{th} vertex set. We have $\sum_{j=1}^{n/s} m_j = m$. This worst case is when $m_i = sm/n$. In that case the total work is $O(\sqrt{nm})$ for the fixed size s. There are $\log n$ different set sizes in the algorithm. Moreover each of the of the $O(n \log n)$ calls to the minimum finding procedure should succeed with probability $1 - 1/2n \log n$ at least, requiring $O(\log n)$ repetitions. □

7 The Lower Bounds

Theorem 7. *The bounded error quantum query complexity of*

1. *the problems* FIND d SMALLEST VALUES OF A FUNCTION, FIND d ELEMENTS OF DIFFERENT TYPE *and* FIND d SMALLEST VALUES OF DIFFERENT TYPE *are* $\Omega(\sqrt{dN})$,
2. MINIMUM SPANNING TREE *and* SINGLE SHORTEST PATHS *are* $\Omega(\sqrt{nm})$ *in the array model,*
3. CONNECTIVITY *is* $\Omega(n)$ *in the array model and* $\Omega(n^{3/2})$ *in the matrix model,*

4. STRONG CONNECTIVITY *is* $\Omega(\sqrt{nm})$ *in the array model and* $\Omega(n^{3/2})$ *in the matrix model,*

Most proofs are based on Ambainis' standard quantum adversary method [2]. Independent of us similar proofs where found in [4] for other graph problems. We only describe in detail the proof for CONNECTIVITY in the matrix model.

Proof. You are given a symmetric matrix $M \in \{0,1\}^{n \times n}$ with the promise that it is the adjacency matrix of a graph with exactly one or two cycles, and have to find out which is the case.

Let X be the set of all adjacency matrices of a unique cycle, and Y the set of all adjacency matrices with exactly two cycles each of length between $n/3$ and $2n/3$. We define the relation $R \subseteq X \times Y$ as $M \; R \; M'$ if there exist $a, b, c, d \in [n]$ such that the only difference between M and M' is that $(a,b),(c,d)$ are edges in M but not in M' and $(a,c),(b,d)$ are edges on M' but not in M. See figure 1. The definition of Y implies that in M the distance from a to c is between $n/3$ and $2n/3$.

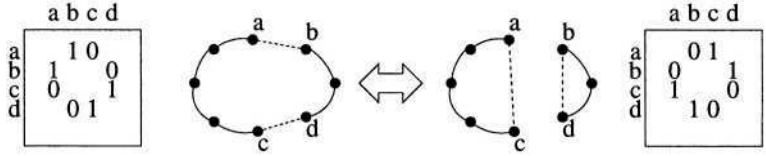


Fig. 1. Illustration of the relation

Referring to the notation of [2, theorem 6] we have $m = \Theta(n^2)$ since there are $n-1$ choices for the first edge and $n/3$ choices for the second edge. Also $m' = \Theta(n^2)$ since from each cycle one edge must be picked, and cycle length is at least $n/3$.

We have $l_{M,(i,j)} = 4$ if $M_{i,j} = 0$ since in M' we have the additional edge (i,j) and the endpoints of the second edge must be neighbors of i and j respectively. Moreover $l_{M,(i,j)} = \Theta(n)$ if $M_{i,j} = 1$ since then (i,j) is one of the edges to be removed and there remains $n/3$ choices for the second edge.

The values $l'_{M',(i,j)}$ are similar, so in the product one factor will always be constant while the other is linear giving $l_{M,(i,j)} l'_{M',(i,j)} = \Theta(n)$ and the theorem follows. □

Acknowledgments. For helpful discussions or comments we are grateful to Miklos Santha, Katalin Friedl, Oded Regev, Ronald de Wolf and Andris Ambainis. We thank one anonymous referee for pointing out a mistake in Theorem 6.

References

1. A. Aharonov, A. Ambainis, J. Kempe and U. Vazirani. *Quantum walks on graphs.* In Proceedings of 33th Annual ACM Symposium on Theory of Computing (STOC), pages 50–59, 2001.
2. A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences,* **64**, pages 750–767, 2002.
3. C.H. Bennett, E. Bernstein, G. Brassard and U. Vazirani, Strengths and weaknesses of quantum computing. *SIAM Journal on Computing,* **26**(5), pages 1510–1523, 1997.
4. A. Berzina, A. Dubrovsky, R. Freivalds, L. Lace and O. Scegulnaja, *Quantum Query Complexity for Some Graph Problems,* In Proceedings of the 30th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM), pages 140–150, 2004.
5. O. Borůvka. O jistem problemu minimaln im, Prace Mor. Prrodove Spol. v Brne (Acta Societ. Scient. Natur. Moravicae), **3**, pages 37–58, 1926.
6. M. Boyer, G. Brassard, P. Høyer and A. Tapp. Tight bounds on quantum searching. *Fortschritte Der Physik,* **46**(4-5), pages 493–505, 1998.
7. G. Brassard and P. Høyer. *An exact quantum polynomial-time algorithm for Simon's problem.* In Proceedings of Fifth Israeli Symposium on Theory of Computing and Systems (ISTCS), pages 12–23, 1997.
8. G. Brassard, P. Høyer, M. Mosca and A. Tapp. *Quantum amplitude amplification and estimation.* In Quantum Computation and Quantum Information: A Millennium Volume, AMS Contemporary Mathematics Series.
9. H. Buhrman, R. Cleve, R. de Wolf and Ch. Zalka, *Bounds for Small-Error and Zero-Error Quantum Algorithms.* In 40th IEEE Symposium on Foundations of Computer Science (FOCS), pages 358–368, 1999.
10. A.M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann and D.A. Spielman. *Exponential algorithmic speedup by quantum walk,* In Proceedings of 35th Annual ACM Symposium on Theory of Computing (STOC), pages 59–68, 2003.
11. C. Dürr and P. Høyer. *A quantum algorithm for finding the minimum,* quant-ph/9607014, 1996.
12. Fahri, E., Goldstone, J., Gutmann, S. and Sipser,M. *A limit on the speed of quantum computation in determining parity,* quant-ph/9802045, 1998.
13. L. Grover. *A fast mechanical algorithm for database search.* In Proceedings of 28th Annual ACM Symposium on Theory of Computing (STOC), pages 212–219, 1996.
14. M. R. Henzinger and M. L. Fredman. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica,* **22**, pages 351–362, 1998.
15. J. Kempe. *Quantum random walks—an introductory overview,* Contemporary Physics, **44**(4), pages 307–327 2003.
16. D. Kozen. *The Design and Analysis of Algorithms.* Springer–Verlag, 1991.
17. J. B. Kruskal Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society,* 1956.
18. A. Nayak, *Lower Bounds for Quantum Computation and Communication,* PhD from the University of California, Berkeley, 1999.
19. A. Nayak and F. Wu. *The quantum query complexity of approximating the median and related statistics.* In Proceedings of 31th Annual ACM Symposium on Theory of Computing (STOC), pages 384–393, 1999.
20. J. Nesetril, E. Milková, and H. Nesetrilová. Otakar Borůvka on Minimum Spanning Tree Problem: translation of both the 1926 papers, comments, history. *Discrete Mathematics,* **233**, pages 3–36 2001.

21. R. Prim. Shortest connecting networks and some generalizations. *Bell Syst. Tech. J.*, 1957.
22. J. Watrous. Quantum simulations of classical random walks and undirected graph connectivity. *Journal of Computer and System Sciences,* **62**(2), pages 376–391, 2001.

A Domain Theoretic Account of Picard's Theorem

A. Edalat[1] and D. Pattinson[2]

[1] Department of Computing, Imperial College London, UK
[2] Institut für Informatik, LMU München, Germany

Abstract. We present a domain-theoretic version of Picard's theorem for solving classical initial value problems in $\mathbb{R}^n$. For the case of vector fields that satisfy a Lipschitz condition, we construct an iterative algorithm that gives two sequences of piecewise linear maps with rational coefficients, which converge, respectively from below and above, exponentially fast to the unique solution of the initial value problem. We provide a detailed analysis of the speed of convergence and the complexity of computing the iterates. The algorithm uses proper data types based on rational arithmetic, where no rounding of real numbers is required. Thus, we obtain an implementation framework to solve initial value problems, which is sound and, in contrast to techniques based on interval analysis, also complete: the unique solution can be actually computed within any degree of required accuracy.

1 Introduction

We consider the initial value problem (IVP) given by the system of differential equations

$$\dot{y}_i(x) = v_i(y_1, \ldots, y_n), \quad y_i(0) = 0 \qquad (i = 1, \ldots, n) \tag{1}$$

where the vector field $v : O \to \mathbb{R}^n$ is continuous in a neighbourhood $O \subseteq \mathbb{R}^n$ of the origin, and we look for a differentiable function $y = (y_1, \ldots, y_n) : [-a, a] \to \mathbb{R}^n$, defined in a neighbourhood of $0 \in \mathbb{R}$, which satisfies (1). By a theorem of Peano there is always a solution [9, page 19]. Uniqueness of the solution is guaranteed, by Picard's theorem, if v satisfies a Lipschitz condition. The question of computability and the complexity of the initial value problem has been studied in different contexts in computable analysis [12,3,8,14,19,17,6].

On the algorithmic and more practical side, standard numerical packages for solving IVP's try to compute an approximation to a solution with a specified degree of accuracy. Although these packages are usually robust, their methods are not guaranteed to be correct and it is easy to find examples where they output inaccurate results [13].

Interval analysis [16] provides a method to give upper and lower bounds for the unique solution in the Lipschitz case with a prescribed tolerance, and has been developed and implemented for analytic vector fields [18,1]. In this approach, arithmetic operations are performed on intervals, and outward rounding is applied if the resulting interval endpoints are not machine representable. While this strategy guarantees soundness, i.e. containment of the exact result in the computed interval, one has in general no control over the rounding, which can produce unduly large intervals. As a consequence, for an implementation of the framework for solving IVP based on interval analysis, one cannot

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 494–505, 2004.

in general guarantee completeness, that is, actual convergence to the solution. For the same reason, one has no control over the speed of convergence.

Domain theory [4] presents an alternative technique, based on proper data types, to produce a provably correct solution with any given degree of accuracy. Using the domain of Scott continuous interval valued functions on a compact interval, we define here a domain theoretic Picard operator, whose least fixed point contains any solution of the IVP. When the vector field is Lipschitz, the solution is unique and we construct an iterative algorithm that gives two sequences of piecewise linear maps with rational coefficients, which converge, respectively from below and above, exponentially fast to the unique solution of the initial value problem. Since the data types for representing the piecewise linear maps with rational coefficients are directly representable on a digital computer, no rounding of real numbers is required. As a consequence, the implementation of the domain theoretic approach is also complete, that is, we can guarantee the convergence of the approximating iterates to the solution of the IVP also for the implementation. To our knowledge, this property is not present in any other approach to validated solutions of differential equations. Furthermore, as a result of the data types we use, we can give estimates for the speed of convergence of the approximating iterates, which are still valid for an actual implementation of our algorithm.

This simplifies the earlier treatment [10], which used a domain for C^1 functions [11] and, at each stage of iteration, required a new approximation of the derivative of the solution. The new treatment is much more similar to the classical theorem in that it gives rise, in the Lipschitz case, to fast convergence of the approximations to the solution.

We discuss two different bases to represent approximations to the solutions of the IVP, namely the piecewise linear and the piecewise constant functions with rational coefficients. Using piecewise linear functions, there is no need to compute rectangular enclosures of the solution, and we therefore avoid the wrapping effect, a well known phenomenon in interval analysis. This comes at the expense of an increase in the size of the representation of the approximations to the solution. Using the base consisting of piecewise constant functions, we show that the order of the speed of convergence to the solution remains unchanged, while the space complexity for the representation of the iterates is much reduced.

A prototypical implementation using the GNU multi precision library [2] shows that the resulting algorithms are actually feasible in practice, and we plan to refine the implementation and compare it in scope and performance with existing interval analysis packages like AWA [1]. Of course we have to bear in mind that floating point arithmetic used by interval software is executed on highly optimised processors, whereas the rational arithmetic needed for our implementation is performed by software.

2 Preliminaries and Notation

For the remainder of the paper, we fix a continuous vector field

$$v = (v_1, \ldots, v_n) : [-K, K]^n \to [-M, M]^n$$

which is defined in a compact rectangle containing the origin and consider the IVP given by Equation (1). Note that any continuous function on a compact rectangle is bounded, hence we can assume, without loss of generality, that v takes values in $[-M, M]^n$.

We construct solutions $y : [-a, a] \to \mathbb{R}^n$ of Equation (1) where $a > 0$ satisfies $aM \leq K$. This will guarantee that the expression $v(y)$ is well defined, since M is a bound for the derivative of y. We consider the *n-dimensional* Euclidean space $\mathbb{R}^n$ equipped with the maximum norm $\|x\| = \max\{|x_1|, \ldots, |x_n|\}$, as this simplifies dealing with the Lipschitz conditions, which we introduce later. Approximations of real numbers live in the interval domain

$$\mathbb{IR} = \{[a, b] \mid a, b \in \mathbb{R}, a \leq b\} \cup \{\mathbb{R}\} \text{ with } [a, b] \sqsubseteq [c, d] \Leftrightarrow [c, d] \subseteq [a, b]$$

ordered by reverse inclusion; the way below relation is given by $[a, b] \ll [c, d]$ iff $[c, d] \subseteq (a, b)$. For $n \geq 1$, the domain $\mathbb{IR}^n$ is isomorphic to the domain of *n-dimensional* rectangles $\{\alpha_1 \times \cdots \times \alpha_n \mid \alpha_i \in \mathbb{IR} \text{ for all } 1 \leq i \leq n\}$, and we do not distinguish between these two presentations. For a rectangle $R \subseteq \mathbb{R}^n$, the subset $\{S \in \mathbb{IR}^n \mid S \subseteq R \text{ is a rectangle}\}$ of rectangles contained in R is a sub-domain of $\mathbb{IR}^n$, which is denoted by $\mathbb{I}R$. The powers $\mathbb{IR}^n$ of the interval domain and the sub-domain $\mathbb{I}R$, for a rectangle R, are continuous Scott domains. If $\alpha^-, \alpha^+ \in \mathbb{R}^n$ with $\alpha_i^- \leq \alpha_i^+$ for all $1 \leq i \leq n$, we write $[\alpha^-, \alpha^+]$ for the rectangle $[\alpha_1^-, \alpha_1^+] \times \cdots \times [\alpha_n^-, \alpha_n^+]$. Similarly, if $f : X \to \mathbb{IR}^n$ is a function, we write $f = [f^-, f^+]$ if $f(x) = [f^-(x), f^+(x)]$ for all $x \in X$.

The link between ordinary and interval valued function is provided by the notion of *extension*. If $R \subseteq \mathbb{R}^n$ is a rectangle, we say that $F : \mathbb{I}R \to \mathbb{IR}^n$ is an extension of $f : R \to \mathbb{R}^n$ if

$$F(\{x_1\}, \ldots, \{x_n\}) = \{f(x_1, \ldots, x_n)\}$$

for all $x \in R$. Note that every continuous function $f : R \to \mathbb{R}^n$ has a *canonical* extension F defined by

$$F = (F_1, \ldots, F_n) : \mathbb{I}R \to \mathbb{IR}^n \text{ with } F_i(S) = [\inf_{x \in S} f_i(x), \sup_{x \in S} f_i(x)],$$

where $S \in \mathbb{I}R$ is a rectangle, which is maximal in the set of interval valued functions extending f. It is easy to see that F is continuous wrt. the Scott topology on $\mathbb{I}R$ and $\mathbb{IR}^n$ if f is continuous wrt. the Euclidean topology.

We consider the following spaces for approximating the vector field and the solutions to the IVP:

- $\mathcal{S} = [-a, a] \to \mathbb{I}[-K, K]^n$, the set of continuous functions wrt. the Euclidean topology on $[-a, a]$ and the Scott topology on $\mathbb{I}[-K, K]^n$
- $\mathcal{V} = \mathbb{I}[-K, K]^n \to \mathbb{I}[-M, M]^n$, the set of continuous functions wrt. the Scott topology on $\mathbb{I}[-K, K]^n$ and $\mathbb{I}[-M, M]^n$.

In order to measure the speed of convergence, as well as for technical convenience in the formulation of some of our results, we introduce the following notation, where X is an arbitrary set:

- For a rectangle $\alpha = [\alpha^-, \alpha^+]$, $w(\alpha) = \|\alpha^+ - \alpha^-\|$ denotes the *width* of α. Similarly, if $f : X \to \mathbb{IR}^n$ is a function, $w(f) = \sup_{x \in X} w(f(x))$ is the *width* of f.
- For $u, u' : X \to \mathbb{R}$ with $u'(x) \sqsubseteq u(x)$ for all $x \in X$, the *width of u' relative to u* is defined as $w_u(u') = \sup_{x \in X} \|u'^+(x) - u^+(x) + u^-(x) - u'^-(x)\|$.

Considering u' as approximation to u, the relative width $w_u(u')$ can be understood as measuring the quality of the approximation.

3 The Picard Operator in Domain Theory

In the classical proof of Picard's theorem on the existence and uniqueness of the solution of the initial value problem (1) one defines an integral operator on $C^0[-a, a]$ by

$$y \mapsto \lambda x. \int_0^x v(y(t))dt$$

(with the integral understood componentwise), which can be shown to be a contraction for sufficiently small a provided v satisfies a Lipschitz condition [15]. An application of Banach's theorem then yields a solution of the initial value problem. We now define the domain-theoretic Picard operator for arbitrary continuous vector fields $u : \mathbf{I}[-K, K]^n \to \mathbf{I}[-M, M]^n$ and focus on the special case where u is an extension of a classical function later. As in the classical proof, the Picard operator is an integral operator, and we therefore introduce the integral of interval-valued functions.

Definition 1. *Suppose* $f = [f^-, f^+] : [-a, a] \to \mathbb{R}$ *is Scott continuous. For* $x \in [-a, a]$ *we let*

$$\int_0^x f(t)dt = [\int_0^x f^{-\sigma}(t)dt, \int_0^x f^{\sigma}(t)dt]$$

where $\sigma = \mathrm{sgn}(x)$ *is the sign of* x. *If* $f = (f_1, \ldots, f_n) : [-a, a] \to \mathbb{R}^n$, *we let* $\int_0^x f(t)dt = (\int_0^x f_1(t)dt, \ldots, \int_0^x f_n(t)dt)$.

Note that, if we integrate in the positive x-direction, then f^- contributes to the lower function associated with the integral of f and f^+ contributes to the upper function. If we integrate in the negative x-direction, the roles of f^- and f^+ are swapped. The following shows that our definition is meaningful:

Lemma 1. *Suppose* $f : [-a, a] \to \mathbb{R}$ *is Scott continuous.*

(i) f^- *and* f^+ *are Lebesgue integrable*
(ii) $\int_0^x f(t)dt \in \mathbb{R}$ *for all* $x \in [-a, a]$.

Proof. For Scott continuous f, the functions f^-, f^+ are lower (resp. upper) semi continuous, hence Lebesgue integrable. If $\sigma = \mathrm{sgn}(x)$, then $\sigma f^{-\sigma} \leq \sigma f^{\sigma}$ and $\int_0^x f^{-\sigma}(t)dt \leq \int_0^x f^{\sigma}(t)dt$ follows from the definition of the ordinary integral. Finally $\int_0^x f(t)dt$ is either compact or the whole of $\mathbb{R}$, since $f^+(t) = \infty$ iff $f^-(t) = -\infty$, for all $t \in [-a, a]$.

The following lemma shows that integration is compatible with taking suprema.

Lemma 2. *Let* $f : [-a, a] \to \mathbb{R}^n$.

(i) The function $\lambda x. \int_0^x f(t)dt$ *is Scott continuous.*
(ii) The function $\int : f \mapsto \lambda x. \int_0^x f(t)dt$ *is Scott continuous.*

Proof. We assume $n = 1$ from which the general case follows. If $g(x) = \int_0^x f(t)dt$, then g^-, g^+ are continuous, hence g is Scott continuous. The second statement follows from the monotone convergence theorem.

The domain theoretic Picard operator can now be defined as follows:

Definition 2. *Suppose* $u \in \mathcal{V}$. *The* domain theoretic Picard operator $P_u : \mathcal{S} \to \mathcal{S}$ *is defined by* $P_u(y) = \lambda x. \int_0^x u(y(t))dt$.

Lemma 3. P_u *is well defined and continuous.*

Proof. That $P_u(y) \in \mathcal{S}$ follows from our assumption $aM \leq K$. Lemma 2 shows that $P_u(y)$, for $y \in \mathcal{S}$, and P_u itself are continuous.

In the classical proof of Picard's theorem, one constructs solutions of IVP's as fixpoint of the (classical) Picard operator. The domain theoretic proof replaces Banach's theorem with Klenee's theorem in the construction of a fixed point of the (domain theoretic) Picard operator. Unlike the classical case, where one chooses an arbitrary initial approximation, we choose the function $y_0 = \lambda t.[-K, K]^n$ with the least possible information as initial approximation.

Theorem 4. *Let* $u \in \mathcal{V}$ *and* $y_{k+1} = P_u(y_k)$. *Then* $y = \bigsqcup_{k \in \mathbb{N}} y_k$ *satisfies* $P_u(y) = y$.

Proof. Follows immediately from the Kleene's Theorem, see e.g. [4, Theorem 2.1.19].

The bridge between the solution of the domain-theoretic fixpoint equation and the classical initial value problem is established in the following proposition, where $\mathbf{I}f : [-a, a] \to \mathrm{I}[-K, K]^n$ denotes the function $\lambda x.\{f(x)\}$, for $f : [-a, a] \to [-K, K]^n$.

Proposition 5. *Suppose* u *is an extension of* v *and* y *is the least fixpoint of* P_u.

(i) If $f : [-a, a] \to \mathrm{I}[-K, K]^n$ *solves (1) then If* $\sqsubseteq y$.
(ii) If y *has width* 0, *then* $y^- = y^+$ *solves (1).*

Proof. For the first statement, note that If is a fixed point of P_u and y is the least such. The second statement follows from the fundamental theorem of calculus; note that $y^- = y^+$ implies the continuity of both.

The previous proposition can be read as a soundness result: Every solution of the IVP is contained in the least fixpoint of the domain theoretic Picard operator.

4 The Lipschitz Case

We can ensure the uniqueness of the solution of the IVP by requiring that the vector field satisfies an interval version of the Lipschitz property. Recall that for metric spaces (M, d) and (M', d'), a function $f : M \to M'$ is Lipschitz, if there is $L \geq 0$ such that $d'(f(x), f(z)) \leq L \cdot d(x, z)$ for all $x, z \in M$. The following definition translates this property into an interval setting.

Definition 3 (Lipschitz Condition). *Suppose* $u : \mathrm{I}[-K, K]^n \to \mathrm{I}[-M, M]^n$. *Then* u *is interval Lipschitz if there is some* $L \geq 0$ *such that* $w(u(\alpha)) \leq L \cdot w(\alpha)$ *for all* $\alpha \in \mathrm{I}[-K, K]^n$. *In this case,* L *is called a* Lipschitz constant *for* u.

The following Proposition describes the relationship between the classical notion and its interval version.

Proposition 6. *For* $v : [-K, K]^n \to [-M, M]^n$, *the following are equivalent:*

(i) v *is Lipschitz*
(ii) The canonical extension of v *satisfies an interval Lipschitz condition*
(iii) v *has an interval Lipschitz extension.*

Note that every interval Lipschitz function induces a total and continuous classical function.

Corollary 7. *Suppose* u *is interval Lipschitz. Then* $w(u(\alpha)) = 0$ *whenever* $w(\alpha) = 0$, *and the induced real valued function* $\bar{u}$, *given by* $\bar{u}(x) = z$ *iff* $u(\{x\}) = \{z\}$ *is continuous.*

In order to guarantee that the sequence of approximations to the solution of the IVP does converge to a width-zero function, we make the following assumption.

For the remainder of the paper, u denotes an extension of v, which satisfies an interval Lipschitz condition with Lipschitz constant L such that $aL < 1$. For later use, we fix $c \in \mathbb{R}$ with $aL < c < 1$.

In case this assumption is not valid, i.e. $aL > 1$, we pick $a' < a$ such that $a'L < 1$ and divide the interval $[-a, a]$ into subintervals of length $< a'$. Replacing a by a' allows us to compute solutions on each subinterval. As we will show in the full version, we can use a glueing process to obtain a solution defined on the whole of $[-a, a]$; this is as in the classical theory [9, page 13].

Assuming the Lipschitz condition, we have the following estimate, which guarantees that the least fixed point of P_u is of width 0:

Lemma 8. $w(P_u(y)) \leq aL \cdot w(y)$ *for all* $y \in \mathcal{S}$.

The above estimate allows us to show that – in the Lipschitz case – the least fixed point of the domain-theoretic Picard operator has width 0, i.e. solves the initial value problem, as shown in Proposition 5.

Proposition 9. *Let* $y_{k+1} = P_u(y_k)$ *for* $k \in \mathbb{N}$. *Then* $w(y_k) \leq c^k w(y_0)$. *In particular,* $y = \bigsqcup_{k \in \mathbb{N}} y_k$ *satisfies* $P_u(y) = y$ *and* $w(y) = 0$.

Proof. Follows immediately from $aL < c < 1$ by induction.

In order to be able to compute the integrals, we now consider approximations to u; the basic idea is that every continuous vector field can be approximated by a sequence of step functions (i.e. functions taking only finitely many values), which allows us to compute the integrals involved in calculating the approximations to the solution effectively. The key property which enables us to use approximations also to the vector field is the continuity of the mapping $u \mapsto P_u$.

Lemma 10. *The map* $P : \mathcal{V} \to \mathcal{S} \to \mathcal{S}$, $u \mapsto P_u$, *is continuous.*

Proof. Follows from continuity of u and the monotone convergence theorem.

This continuity property allows us to compute solutions to the classical initial value problem by means of a converging sequence of approximations of u.

Theorem 11. *Suppose* $u = \bigsqcup_{k\in\mathbb{N}} u_k$ *and* $y_{k+1} = P_{u_k}(y_k)$ *for* $k \in \mathbb{N}$. *Then* $y = \bigsqcup_{k\in\mathbb{N}} y_k$ *satisfies* $y = P_u(y)$ *and* $w(y) = 0$.

Proof. Follows from Theorem 4 and continuity of $u \mapsto P_u$ by the interchange-of-suprema law (see e.g. [4, Proposition 2.1.12]).

We have seen that the Lipschitz condition on the vector field ensures that the approximations of the solution converge exponentially fast (Proposition 9). In presence of approximations of the vector field, the speed of convergence will also depend on how fast the vector field is approximated. The following estimate allows to describe the speed of convergence of the iterates if the vector field is approximated by an increasing chain of vector fields.

Lemma 12. *Suppose* $u' \sqsubseteq u$ *and* $y \in \mathcal{S}$. *Then* $w(P_{u'}(y)) \leq aL \cdot w(y) + a \cdot w_u(u')$.

As a corollary we deduce that the approximations converge exponentially fast, if the approximations of the vector field do so too.

Proposition 13. *Suppose* $u = \bigsqcup_{k\in\mathbb{N}} u_k$ *and* $y_{k+1} = P_{u_k}(y_k)$. *Then* $w(y_k) \leq c^k \cdot w(y_0)$ *provided* $w_u(u_k) \leq c^k \cdot 2M(c - aL)$.

Given a representation of u in terms of step functions, Theorem 11 gives rise to an algorithm for computing the solution of the initial value problem. Our next goal is to show that this algorithm can be restricted to bases of the respective domains, showing that it can be implemented without loss of accuracy. We then give an estimate of the algebraic complexity of the algorithm.

5 An Implementation Framework for Solving IVP's

We now show that the algorithm contained in Theorem 11 is indeed implementable by showing that the computations can be carried out in the bases of the domains. In fact, we demonstrate that every increasing chain of (interval valued) vector fields $(u_k)_{k\in\mathbb{N}}$, where each u_k is a base element of $\mathcal{V}$, gives rise to a sequence of base elements of $\mathcal{S}$, which approximate the solution and converge to it.

In view of the algorithm contained in Theorem 11, we consider simple step functions as base of $\mathcal{V}$ and piecewise linear function as base of $\mathcal{S}$. Note that in this setup, the domain-theoretic Picard operator computes integrals of piecewise constant functions, hence produces piecewise linear functions.

We begin by introducing the bases which we are going to work with.

Definition 4. *Let* $D \subseteq \mathbb{R}$ *and assume that* $-a = a_0 < \cdots < a_k = a$ *with* $a_0, \ldots, a_k \in D$, $\beta_0, \ldots, \beta_k \in \mathrm{I}[-K, K]^n_D$ *and* $\gamma_1, \ldots, \gamma_k \in \mathrm{I}[-M, M]^n_D$, *where* R_D *denotes the set of rectangles, which are contained in R and whose endpoints lie in D. We consider the following classes of functions:*

(i) The class $\mathcal{S}_D^L$ *of piecewise D-linear functions* $[-a,a] \to \mathrm{I}[-K,K]^n$,

$$f = (a_0, \ldots, a_k) \searrow^L (\beta_0, \ldots, \beta_k)$$

where $f(x)^{\pm} = \beta_{j-1}^{\pm} + \frac{x - a_{j-1}}{a_j - a_{j-1}}(\beta_j^{\pm} - \beta_{j-1}^{\pm})$ *for* $x \in [a_{j-1}, a_j]$. *Every component of a D-linear function is piecewise linear and takes values in D at* $a_0, a_1 \ldots, a_k$.

(ii) The set $\mathcal{S}_D^C$ *of piecewise D-constant functions* $[-a,a] \to \mathrm{I}[-K,K]^n$,

$$f = (a_0, \ldots, a_k) \searrow^C (\beta_1, \ldots, \beta_n), x \mapsto \begin{cases} \beta_i & x \in [a_{i-1}, a_i)^o \\ \beta_{i-1} \sqcap \beta_i & x = a_i \text{ and } 1 < i < k \end{cases}$$

where $\sqcap$ *denotes the greatest lower bound. The components of a D-constant function assume constant values in D, which only change at* $a_0, a_1, \ldots, a_k$.

(iii) The set $\mathcal{V}_D$ *of finite sups of step functions* $\mathrm{I}[-K,K]^n \to \mathrm{I}[-M,M]^n$,

$$f = \bigsqcup_{1 \le j \le k} \beta_j \searrow \gamma_j : x \mapsto \bigsqcup \{\gamma_j \mid 1 \le j \le k, \beta_j \ll x\}.$$

(iv) For any f *as above, we put* $\mathcal{N}(f) = k$ *and call it the* complexity of representation *of* f.

Since we will not consider different representations for the same functions, we allow ourselves to blur the distinction between a function and its representation as step function. Note that any computable vector field u can be approximated by a sequence of basis elements $(u_k)_{k \in \mathbb{N}}$ in $\mathcal{V}_{\mathbb{Q}}$, and such approximating sequences can be constructed from a library of elementary functions.

If D is dense in $\mathbb{R}$, it is well known that the sets defined above are bases of their respective superspaces:

Proposition 14. *Suppose* $D \subseteq \mathbb{R}$ *is dense and* $-a, a \in D$.

(i) $\mathcal{V}_D$ *is a base of* $\mathcal{V}$.
(ii) $\mathcal{S}_D^C$ *and* $\mathcal{S}_D^L$ *are bases of* $\mathcal{S}$.

We can now show that the Picard operator P_u associated with a simple step function u restricts to an endofunction on the set of basis elements of the space of linear step functions $\mathcal{S}_D^L$.

Lemma 15. *Suppose* $D \subseteq \mathbb{R}$ *is a subfield,* $u \in \mathcal{V}_D$ *and* $y \in \mathcal{S}_D^L$. *Then, there is* $f \in \mathcal{S}_D^C$ *with* $\mathcal{N}(f) \le 3\mathcal{N}(y)\mathcal{N}(u)$ *and* $u \circ y(x) = f(x)$ *for all but finitely many* $x \in [-a,a]$. *Moreover,* f *can be computed in time* $\mathcal{O}(\mathcal{N}(u)\mathcal{N}(y))$.

Now that we have a basis representation of $u \circ y$, it's easy to obtain a basis representation of $P_u(y)$ by integration. Note that computing integrals can be performed over a base defined over a subring of $\mathbb{R}$; we will make use of this fact later.

Lemma 16. *Suppose* $D \subseteq \mathbb{R}$ *is a subring and let* $g(x) = \int_0^x f(x)dx$ *for* $f \in \mathcal{S}_D^C$. *Then* $g \in \mathcal{S}_D^L$ *and* $\mathcal{N}(g) = \mathcal{N}(f)$. *Furthermore,* g *can be computed in* $\mathcal{O}(\mathcal{N}(f))$ *steps.*

Summing up, we have the following estimate on the algorithm induced by Theorem 11 if we compute over the base of piecewise linear functions.

Proposition 17. *Suppose* $D \subseteq \mathbb{R}$ *is a subfield,* $u \in \mathcal{V}_D$ *and* $y \in \mathcal{S}_D^L$.

(i) $P_u(y) \in \mathcal{S}_D^L$
(ii) $P_u(y)$ *can be computed in time* $\mathcal{O}(\mathcal{N}(u)\mathcal{N}(y))$.
(iii) $\mathcal{N}(P_u(y)) \in \mathcal{O}(\mathcal{N}(u)\mathcal{N}(y))$.

We can now summarise our results as follows:

Theorem 18. *Suppose* $D \subseteq \mathbb{R}$ *is a subfield and* $u = \bigsqcup_{k\in\mathbb{N}} u_k$ *with* $u_k \in \mathcal{V}_D$. *If* $y_{k+1} = P_{u_k}(y_k)$, *then*

(i) $y_k \in \mathcal{S}_D^L$ *for all* $k \in \mathbb{N}$
(ii) $y = \bigsqcup_{k\in\mathbb{N}} y_k$ *has width* 0 *and* $y^- = y^+$ *solves the IVP (1).*
(iii) $w(y_k) \in \mathcal{O}(c^k)$ *if* $w_u(u_k) \in \mathcal{O}(c^k)$.

Since the elements of $\mathcal{S}_D^L$ for $D = \mathbb{Q}$, the set of rational numbers, can be represented faithfully on a digital computer, the theorem – together with Proposition 5 – guarantees soundness and completeness also for implementations of the domain theoretic method. We also provide a guarantee on the speed of convergence, since the condition $w_u(u_k) \in \mathcal{O}(c^k)$ can always be ensured by the library used to construct the sequence (u_k) of approximations to the vector field.

Also, computing over the base of piecewise linear functions eliminates the need of computing rectangular enclosures at every step of the computation. This avoids the well-known wrapping effect of interval analysis, but it comes at the cost of a high complexity of the representation of the iterates. The next section presents an alternative, which uses piecewise constant functions only.

6 Computing with Piecewise Constant Functions

We have seen that the time needed to compute $P_u(y)$ is quadratic in terms of the complexity of the representation of u and y. However, the complexity of the representation of $P_u(y)$ is also quadratic in general. This implies that

$$\mathcal{N}(y_{k+1}) \in \mathcal{O}(\mathcal{N}(u_0)\dots\mathcal{N}(u_k)),$$

if $u = \bigsqcup_{k\in\mathbb{N}} u_k$ and $y_{k+1} = P_{u_k}(y_k)$.

The blow up of the complexity of the representation of the iterates is due to the fact that each interval on which y is linear is subdivided when computing $u \circ y$, since we have to intersect linear functions associated with y with constant functions induced by u, as illustrated by the left diagram in Figure 1.

This can be avoided if we work with piecewise constant functions only. The key idea is to transform the linear step function $P_u(y)$ into a simple step function before computing the next iterate: on every interval, replace the upper (linear) function by its maximum and the lower function by its minimum. We now develop the technical apparatus which is needed to show that the approximations so obtained still converge to the solution. Technically, this is achieved by making the partitions of the interval $[-a, a]$ explicit.

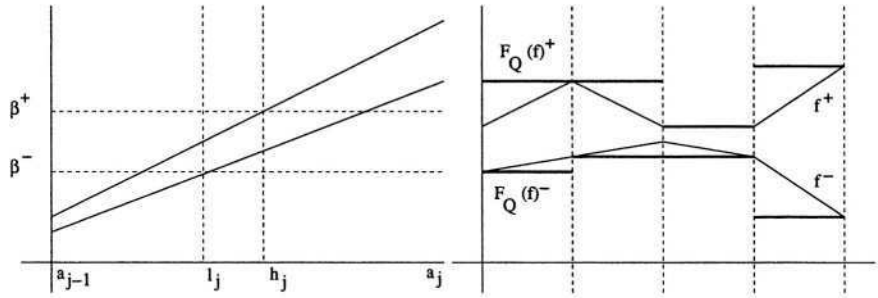


Fig. 1. Subdivision of Intervals (left) and Flattening (right)

Definition 5 (Partitions).

(i) A partition *of* $[-a, a]$ *is a finite sequence* $(q_0, \ldots, q_k)$ *of real numbers such that* $-a = q_0 < \cdots < q_k = a$*; the set of partitions of* $[-a, a]$ *is denoted by* $\mathcal{P}$*. If* $D \subseteq \mathbb{R}$ *then* $\mathcal{P}_D \subset \mathcal{P}$ *is the subset of partitions of* $[-a, a]$ *whose points lie in D.*

(ii) The norm $|Q|$ *of a partition* $Q = (q_0, \ldots, q_k)$ *is given by* $|Q| = \max_{1 \le i \le k} q_i - q_{i-1}$.

(iii) A partition $Q = (q_0, \ldots, q_k)$ refines *a partition* $R = (r_0, \ldots, r_l)$ *if* $\{r_0, \ldots, r_l\} \subseteq \{q_0, \ldots, q_k\}$*; this is denoted by* $R \le Q$.

We are now ready for the definition of the flattening functional, which transforms piecewise linear functions to piecewise constant functions.

Definition 6. *Suppose* $Q \in \mathcal{P}$*. The* flattening functional $F_Q : \mathcal{S} \to \mathcal{S}$ *associated with* Q *is defined by*

$$F_Q(f) = (q_0, \ldots, q_k) \searrow^C (\gamma_1, \ldots, \gamma_k)$$

where $\gamma_i = \bigsqcap \{f(x) \mid x \in [q_{i-1}, q_i]\}$ *for* $1 \le i \le k$.

Note that, geometrically speaking, F_Q computes an enclosure of semi continuous functions into rectangles, as illustrated by the right diagram in Figure 1.

Lemma 19. F_Q *is well defined, that is,* $F_Q(f)$ *is continuous, if* f *is continuous.*

In order to reduce the complexity of the representations of the iterates, we want to apply the flattening functional at every step of the computation. The following lemma is the stepping stone in proving that this does not affect convergence to the solution.

Lemma 20. *Suppose* $(Q_k)_{k \in \mathbf{N}}$ *is an increasing sequence of partitions with* $\lim_{k \to \infty} |Q_k| = 0$*. Then* $\bigsqcup_{k \in \mathbf{N}} F_{Q_k} = \mathrm{id}$.

Proof. This follows from the fact that for every upper semi continuous function $f : [-a, a] \to \mathbb{R}$ and every decreasing chain $\alpha_0 \subseteq \alpha_1 \subseteq \ldots$ of compact intervals containing x with $w(\alpha_k) \to 0$ as $k \to \infty$ one has $f(x) = \inf_{k \in \mathbf{N}} \sup\{f(x) \mid x \in \alpha_k\}$, and the dual statement for lower semi continuous functions.

The last lemma puts us in the position to show that the application of the flattening functional at every stage of the construction does not affect the convergence of the iterates to the solution.

Theorem 21. *Suppose* $u = \bigsqcup_{k\in\mathbb{N}} u_k$, $(Q_k)_{k\in\mathbb{N}}$ *is an increasing sequence of partitions with* $\lim_{k\to\infty} |Q_k| = 0$ *and* $y_{k+1} = F_{Q_k}(P_{u_k}(y_k))$. *Then* $y = \bigsqcup_{k\in\mathbb{N}} y_k$ *satisfies* $y = P_u(y)$ *and* $w(y) = 0$.

Proof. Follows from the interchange-of-suprema law (see e.g. [4, Proposition 2.1.12]), the previous lemma and Theorem 11.

We now show that the speed of convergence is essentially unaffected if we apply the flattening functional at every stage of the computation. This result hinges on the following estimate:

Lemma 22. *Suppose* $u' \in \mathcal{V}$ *with* $u' \sqsubseteq u$ *and* $Q \in \mathcal{P}$, $y \in D$. *Then* $w(F_Q(P_{u'}(y))) \leq aL \cdot w(y) + a \cdot w_u(u') + 2\frac{K}{a}|Q|$.

This lemma implies that flattening does not affect the speed of convergence.

Proposition 23. *Suppose* $u = \bigsqcup_{k\in\mathbb{N}} u_k$ *with* $w_u(u_k) \leq c^k \cdot M(c-aL)$ *and* $(Q_k)_{k\in\mathbb{N}}$ *is an increasing sequence of partitions with* $|Q_k| \leq c^k \cdot \frac{a}{2}(c-aL)$. *Then* $w(y_k) \leq c^k w(y_0)$ *if* $y_{k+1} = F_{Q_k}(P_{u_k}(y_k))$ *for all* $k \geq 0$.

We now show that the application of the flattening functional at every step avoids the blow up of the size of the iterates. As a consequence, the algorithm with flattening can be implemented using a base of functions defined over a dense subring of $\mathbb{R}$, such as the dyadic numbers.

Lemma 24. *Suppose* $D \subseteq \mathbb{R}$ *is a subring and* $Q \in \mathcal{P}_D$. *Then* F_Q *restricts to a mapping* $\mathcal{S}_D^L \to \mathcal{S}_D^C$.

The complexity of the algorithm underlying Theorem 21 over the bases $\mathcal{V}_D$ and $\mathcal{S}_D^C$ can now be summarised as follows, where $\mathcal{N}(Q) = k$ for a partition $Q = (q_0, \ldots, q_k)$.

Lemma 25. *Suppose* $D \subseteq \mathbb{R}$ *is a subring,* $y \in \mathcal{S}_D^C$ *and* $u \in \mathcal{V}_D$.

(i) $F_Q(P_u(y)) \in \mathcal{S}_D^C$ *and* $\mathcal{N}(F_Q(P_u(y)) = \mathcal{N}(Q)$
(ii) $F_Q(P_u(y))$ *can be computed in time* $\mathcal{O}(\max(\mathcal{N}(u) \cdot \mathcal{N}(y), \mathcal{N}(Q))$.

We can now summarise our results concerning soundness and completeness of the algorithm with flattening as follows:

Theorem 26. *Suppose* $D \subseteq \mathbb{R}$ *is a subring and* $u = \bigsqcup_{k\in\mathbb{N}} u_k$ *with* $u_k \in \mathcal{V}_D$. *Furthermore, assume* $(Q_k)_{k\in\mathbb{N}}$ *is an increasing sequence of partitions with* $\lim_{k\to\infty} |Q_k| = 0$ *and* $y_{k+1} = F_{Q_k}(P_{u_k})(y_k)$.

(i) $y_k \in \mathcal{S}_D^C$ *for all* $k \in \mathbb{N}$ *and* $\mathcal{N}(y_k) = \mathcal{N}(Q_k)$.
(ii) $y = \bigsqcup_{k\in\mathbb{N}} y_k$ *has width* 0 *and* $y^- = y^+$ *solves the IVP (1)*
(iii) $w(y_k) \in \mathcal{O}(c^n)$, *if both* $w_u(u_k)$ *and* $|Q_k| \in \mathcal{O}(c^k)$.

Note that, for a subring $R \subseteq \mathbb{Q}$ of the rational numbers, the elements of $\mathcal{V}_D$ and $\mathcal{S}_D^C$ can be faithfully represented on a digital computer. Hence we can guarantee both soundness and completeness also for an implementation of the domain theoretic approach where furthermore the size of the iterates are bounded above by the size of the partitions.

Acknowledgements. This work has been supported by EPSRC in the UK and the EU project "APPSEM-II".

References

1. AWA. A software package for validated solution of ordinary differential equations. www.cs.utep.edu/interval-comp/intsoft.html
2. *The GNU multi precision library.* www.swox.com/gmp/.
3. O. Aberth. Computable analysis and differential equations. In *Intuitionism and Proof Theory, Studies in Logic and the Foundations of Mathematics,* pages 47–52. North-Holland, 1970. Proc. of the Summer Conf. at Buffalo N.Y. 1968.
4. S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science,* volume 3. Clarendon Press, 1994.
5. J. P. Aubin and A. Cellina. *Differential Inclusions.* Springer, 1984.
6. V. Brattka. Computability of Banach space principles. Informatik Berichte 286, FernUniversität Hagen, Fachbereich Informatik, June 2001.
7. F. H. Clarke, Yu. S. Ledyaev, R. J. Stern, and P. R. Wolenski. *Nonsmooth Analysis and Control Theory.* Springer, 1998.
8. J. P. Cleave. The primitive recursive analysis of ordinary differential equations and the complexity of their solutions. *Journal of Computer and Systems Sciences,* 3:447–455,1969.
9. E. A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations.* McGraw-Hill, 1955.
10. A. Edalat, **M. Krznarić,** and A. Lieutier. Domain-theoretic solution of differential equations (scalar fields). In *Proceedings of MFPS XIX,* volume 83 of *Elect. Notes in Theoret. Comput. Sci.,* 2004. Full Paper in www.doc.ic.ac.uk/~ae/papers/scalar.ps.
11. A. Edalat and A. Lieutier. Domain theory and differential calculus (Functions of one variable). In *Seventh Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 2002. Full paper in www.doc.ic.ac.uk/~ae/papers/diffcal.ps.
12. A. Grzegorczyk. Computable functionals. *Fund. Math.,* 42:168–202,1955.
13. A. Iserles. *Numerical Analysis of Differential Equations.* Cambridge Texts in Applied Mathematics. CUP, 1996.
14. Ker-I Ko. On the computational complexity of ordinary differential equations. *Inform. Contr.,* 58:157–194, 1983.
15. A. N. Kolmogorov and S. V. Fomin. *Introductory Real Analysis.* Dover, 1975.
16. R.E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, 1966.
17. N. Th. Müller and B. Moiske. Solving initial value problems in polynomial time. In *In Proceedings of the 22th JAIIO - Panel'93,* pages 283–293, Buenos Aires, 1993.
18. N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation,* 105:21–68,1999.
19. M. B. Pour-El and J. I. Richards. *Computability in Analysis and Physics.* Springer-Verlag, 1988.

Interactive Observability in Ludics

Claudia Faggian

University of Padova, Italy
claudia@math.unipd.it

Abstract. In Ludics [7] a proof/program (called design) can be thought of as a black box, to be studied by making it interact with other designs. We explore what can be recognized interactively in this setting, developing two approaches, which we respectively qualify as dynamic and static. The former consists in studying the geometrical properties of the paths induced by the interaction (normalization), much in the style of the Geometry of Interaction. The latter analyzes statically the properties of a design.

Context. Calculi for sequential or concurrent computation (λ-calculus, π-calculus) allow for formal methods to prove properties of programs/processes. A typical approach is to define equivalence relations on terms, to characterize when two terms are the same *from the point of view of the observer.* Such techniques go back to the works on *lambda*-calculus and PCF and in the context of process calculi have given rise to a whole range of equivalence relations. Still, one can wonder whether the study of equivalence properties could not be simplified by reconsidering the design of the calculus, and tailoring it to have a good theory of interaction. The point of view that interaction should come before syntax is the base of Ludics.

Ludics. Ludics [7] is a recent proposal of analysis of interaction in a proof-theoretical setting. The whole theory is built on the analysis of the interaction between a design (proof/program) and counter-designs (again proofs/programs), where designs are rather simple, combinatorial structures which abstract away from the syntax of proofs. The emphasis on the symmetry between observed system and observer gives rise to interesting geometrical properties, on which to found the study of equivalences.

Ludics encompasses and builds on ideas from proof theory, proof search and different approaches to semantics. In particular, on the semantical side, Ludics is close to Game Semantics (see [5]). On the syntactical side, designs can be understood as proofs in the logic programming approach (as in the work by Andreoli) or as lambda-terms in the form of abstract Böhm trees [1]. For these reasons, Ludics provides a clean, general setting in which interaction and equivalence can be studied at a foundational level using geometrical intuitions.

Contents and relevance to Ludics. The meaning of a proof/program (design) is given by its behaviour, that is by the way it interacts with the other proofs/programs. A design can be seen as a "black box": what we can know of it is only what

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 506–518, 2004.

we can observe by testing it against other designs. Indeed, the *separation theorem* (the analogue of Böhm's theorem) states that if two designs react the same way to all tests, then we cannot distinguish between them. They must actually be the same syntactical object. As all properties of proofs are not only tested but also determined interactively, inside the system itself, it appears important to the theory to understand what can be observed interactively. This is the main object of our paper. It had not been expected in the theory of Ludics to discover that there are properties which cannot be detected in an interactive way: an example is the use of weakening in a proof ([7, p. 392], and [4]).

We provide a characterization of *what can be observed interactively,* for untyped designs (for the typed case see [3]). The structures we work with are finite trees of addresses (or pointers) with certain properties. We characterize the designs that can be observed at each single test *(primitive observables).*

To this purpose we establish combinatorial methods, which we believe are of interest in themselves. We develop two approaches which we respectively qualify as (i) "dynamic" and (ii) "static". (i) consists in studying the geometrical properties of the paths induced by normalization, much in the style of Geometry of Interaction ([2]). (ii) analyzes statically the "spatial" properties of a design.

The geometrical properties we study provide a better understanding of the computational structures and the dynamics of their interaction. In particular, we point out a deep interleaving between two key relations: "time" (sequentiality) and "space" (internal dependency). This understanding will be a base and a guide for further developments.

1 Background, Definitions, and Questions

Let us first introduce informally a few key notions.

Daimon and para-proofs. Ludics provides a general setting in which to any proof of A we can oppose (via cut-elimination) a proof of $A^{\perp}$. To obtain this, it introduces a new rule, called *daimon,* which allows us to justify (assume) any conclusion. Since all proofs are read bottom-up, a daimon stops the flow of computation: it is a sort of error, or a "quit".

Addresses. Proofs do not manipulate formulas, but their *addresses,* which could be thought of as names, channels, or as the address in the memory where the formula is stored. If we give to a formula address ξ , its (immediate) subformulas will receive address $\xi i, \xi j$, etc.

Actions. An elementary operation is called *action* (a move in Game Semantics). An action should be thought of as a cluster of operations which can be performed at the same time. This has a precise proof-theoretical meaning in the calculus which underlies Ludics, i.e. 2nd order multiplicative-additive Linear Logic. Multiplicative and additive connectives of linear logic separate into two families: positives $(\otimes, \oplus)$ and negatives $(\parr, \&)$. A cluster of connectives of the same polarity can be decomposed in a single step, and can be written as a single connective, which is called a *synthetic connective.*

An action has an address (a name) and a polarity (positive or negative): the actions corresponding to A and $A^{\perp}$ will have the same address, but opposite polarity.

Designs. In Ludics, the computational structures (proofs/programs/strategies) are *trees of actions,* which are called *designs.* Designs capture the geometrical structure of sequent calculus derivations, providing a more abstract syntax which is actually very close to that of proof-nets. A proof-theoretical intuition for designs is provided in [4]. Here we are more concerned with the general combinatorial aspects, and will simply think of a design as a tree of actions with certain properties (see later).

Interaction: normalization and slices. Interaction between designs is trough normalization. Normalization of designs is the analogue of normalization on proof-nets, or cut-elimination on sequents.

To understand proof-theory but also interaction in Ludics, it is important to understand the notion of slice. This notion has been introduced as part of the theory of proof-nets of Linear Logic, to deal with the additives. A &-rule can be seen as composed of two unary rules: $\&_1$ and $\&_2$. A usual binary & is the "superimposition" of two unary $\&_i$. To get an idea, think of a sequent-calculus derivation; if for any &-rule we select one of the premises, we obtain a derivation where all &-rules are unary. This is called a *slice.*

It should already be clear that the interaction (normalization) between proofs (or designs...) only involves a part of them. This part is necessarily a slice, as for any application of & there is a $\oplus$ selecting one of the premisses.

It has been en idea of Linear Logic since long that slices are the perfect syntax for additive proof-nets: a proof should be seen as the superimposition of all its slices. The advantage is that normalization of slices enjoys the same pleasant properties as MLL (multiplicative linear logic) proof-nets. The difficulty is how to "superimpose" the slices.

Addresses and sequentialization make this possible in Ludics. A design can conveniently be presented as a set of slices. In fact, normalization on designs works very well by slices: **(i)** normalize all possible slices **(ii)** put the normal forms together. The result is a set of slices, which is a design.

Interactive types. We said that proofs work with addresses rather than with formulas. The use of addresses makes the designs and the calculus *untyped,* as we abstract from the type annotation. *Types* are recovered in Ludics as sets of proof/programs which behave the same way in reaction to the same set of tests: these interactive types are called behaviours.

Definitions. An *address* ξ is a sequence of natural numbers. The empty sequence is indicated by <>. We say that σ is a *subaddress* of ξ if ξ is prefix of σ (written $\xi \sqsubseteq \sigma$); ξi is an immediate subaddress of ξ. If we think of ξ as the address of a formula A, we think of a subaddress of ξ as the address of a subformula of A.

An *action* is either the symbol † or a pair $\kappa = (\xi, I)$ given by an address ξ and a set I of indices which define the possible subaddresses. We say that κ

creates the addresses ξi, for all $i \in I$. To an action in a design is also associated a polarity, positive (κ^+) or negative (κ^-).

Base. A base provides the addresses which will be used in a design (the conclusion of the proof, the specification of the process). To the base is associated a polarity. Here we only consider designs with a single address as base. This captures the most interesting case, does not imply any loss of generality, and will simplify the presentation.

Designs and slices A design is given by a base and a structure which can be presented in several ways. In particular, a design can be seen as a set of slices, with some coherence conditions. Here we are only interested in slices, as the normal form of cut designs is given by the set of normal form of all the possible slices. Any slice in the normal form comes from the normalization of slices in the original designs.

A *slice* is given by a base and a tree of actions satisfying the conditions below. We think of the tree as oriented from the root upwards. If the action κ_1 is *before* κ_2, we write $\kappa_1 < \kappa_2$. A sequence of actions starting from the root is called a *chronicle.* The path leading from the root to an action κ is *the chronicle of* κ.

Root. The root is an action on the address given by the base.
Polarity. The polarities of the actions alternate. The root has the same polarity as the base.
Sub-address. Given an action κ which is not †, its address either belongs to the base or it is a σi such that in the tree there is an action (σ, I), $i \in I$ and $(\sigma, I) < (\sigma i, K)$. This mirrors the *sub-formula property.*
Branching. The tree only branches on positive actions.
Negative addresses. The addresses used immediately after a positive action of address ξ are immediate sub-addresses of ξ.
Leaves. All maximal actions are positive. † can only appear as a leaf.
Linearity. Each address only appears once.

A few *simplifications* are possible. As we focus on the case of base with a single address, and this is forced to be the initial address used by the slice, we identify the base with the root of the designs. All addresses in the tree will be subaddresses of the root.

As in a slice all addresses are distinct (by linearity condition), *each action is uniquely determined by its address.* For this reason, we will identify the action $\kappa = (\sigma, I)$ with its address σ.

The leaves condition allows us to deal with daimon implicitly: we can assume that any maximal action which is negative is followed by a †.

Unless needed, we will not make explicit the polarity of the actions in a design. However, in all pictures, when representing a slice, we indicate the polaritieswe by *circling the positive nodes.*

Sequential and prefix order. In a slice we are given *two partial orders,* corresponding to two kinds of information on the actions: (1.) a time relation (*sequential order*) $\kappa_1 \leq \kappa_2$, and (2.) a space relation *(prefix order),* corresponding

to the relation of being sub-address $\xi \sqsubseteq \xi'$. The sub-address condition can be reformulated as $\sigma \sqsubseteq \xi \Rightarrow \sigma \leq \xi$.

A contribution of this paper is to evidence how the relation between these two orders forces structure and remarkable properties, on which normalization ultimately relies.

Normalization and Orthogonality. As in the λ-calculus, normalization between designs is deterministic but not necessarily converging. The most important use of normalization in Ludics is to oppose a proof of A to a proof of $A^{\perp}$ (a design $\mathfrak{D}$ of base ξ^{+} to a design $\mathfrak{E}$ of base ξ^{-}). In the closed case of normalization (that is A against $A^{\perp}$), the result can only be an empty conclusion (the empty base). There are only two possible outcomes: either normalization fails (diverges) or it converges, producing as result a daimon †, as daimon is the only rule that can "prove" everything, and in particular an empty conclusion.

If normalizing $\mathfrak{D}$ against $\mathfrak{E}$ produces a †, we say that $\mathfrak{D}$ and $\mathfrak{E}$ are *orthogonal:* $\mathfrak{D} \perp \mathfrak{E}$. Orthogonality is a key notion in Ludics. A *type* (called *behaviour*) is a set $\mathbf{G}$ of designs equal to its biorthogonal $\mathbf{G} = \mathbf{G}^{\perp\perp}$.

Normalization on slices coincides with identifying any action with its opposite.

A convenient, operational way to describe the interaction between two slices is given by the following procedure (we think of a token traveling on the slices to be cut, see [4]):

– We *start* on the root of the positive slice.

– From a *positive proper action* κ^{+} we move to the corresponding negative action κ^{-} (changing slice). If κ^{-} does not belong to the cut slices, the process fails.

–From a *negative action* κ^{-} we move upwards to the unique action which follows κ in the same slice.

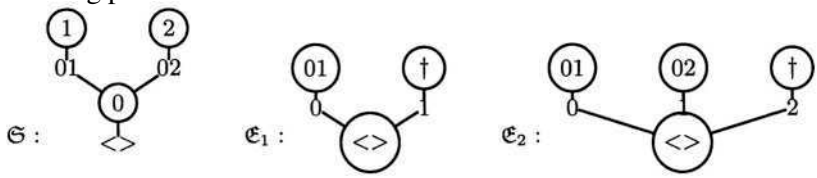
– On the special action †, the process successfully terminates.

Normalization between two slices $\mathfrak{S}$ and $\mathfrak{T}$ establishes a linear order among the actions which are used. The sequence of visited actions, which represents the trace of the *interaction* between $\mathfrak{S}$ and $\mathfrak{T}$, is indicated by $[\mathfrak{S} \rightleftharpoons \mathfrak{T}]$ and called *dispute* (a play in Game Semantics). This sequence induces a path on each of the two cut slices. When we speak of *visiting a slice* (or a design), we always mean visiting it by normalizing. We also call the sequence of visited actions a *normalization path.* Notice as each slice determines a traversal strategy for the actions of its opponent.

What can be observed interactively? Given a slice, can we build a counter-design which is able to completely explore it? Even if we only consider finite slices, the answer is no, as shown by the following example[1], which we have

[1] Remember that we circle the positive nodes.

discussed in [4]. Let us try to build a counter-design to explore the slice $\mathfrak{S}$ in the following picture.



Normalization must start with <>, use 0, and then, depending on the counter-design, it will choose one of the branches, going either to 01 or 02. The two choices are symmetrical, so let us take 01. At 1 we are forced to stop, because there is no way to move to the other branch. In fact the counter-design we have implicitly built is $\mathfrak{E}_1$ (picture above), corresponding to the path $<>,0,01,1,$ while the path we would like to have is $p = <>, 0, 01, 1, 02, 2.$ The tree of actions that would realize this path is actually $\mathfrak{E}_2$ (in the picture above), where we dispose the action in a configuration that via normalization would produce p. However, *this is not a design,* because it does not satisfy the sub-address condition $(0 \not< 02)$.

A consequence, which is relevant to the theory of Ludics, is for example that we *cannot interactively detect* the use of *weakening,* not even in a slice (see [7] or [4] for details).

What is the relation between slices and prefix trees? Given a slice $\mathfrak{S}$, we can associate to it *the prefix tree* of the addresses which appear in $\mathfrak{S}$, with the relation $\sqsubseteq$. Doing this, we forget some information about sequentiality. For example, to the design consisting of the single chronicle $<>^-, 1^+, 1.0, 2^+, 2.0, \dagger$ corresponds the prefix tree in Fig.1, where we forget that the action of address 1 is performed before the action of address 2. The prefix tree of a slice $\mathfrak{S}$ is the addresses analogue of the "sub-formula tree." We indicate it by $\mathbf{T}(\mathfrak{S})$.

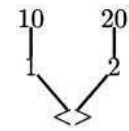
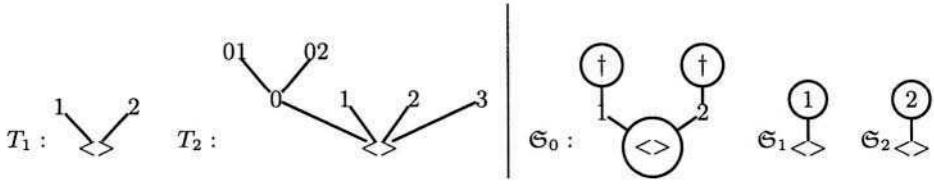


Fig. 1.

Could we deal with slices "modulo sequentiality"? The question is whether given a prefix tree, we can associate to it a slice using the same addresses[2]. If this is possible, in general there will be several ways to do so, corresponding to several ways to sequentialize (add a sequential order to) the actions.

Given a prefix tree T, *a slice* associated to it is any slice $\mathfrak{S}$ such that $\mathbf{T}(\mathfrak{S}) = T$. It is not always possible to associate a slice to a prefix tree. To give an *example,* let us consider the prefix tree T_1 and T_2 in the picture below.



[2] We deal with daimon implicitly: any maximal branch terminating on a negative action is completed by a daimon.

To T_1 we can associate a positive design ($\mathfrak{S}_0$ in the same picture), but there is no way to associate to T_1 a negative design. We can build $\mathfrak{S}_1$ or $\mathfrak{S}_2$, but there is no space to add a second positive action (when we sequentialize, each negative action has space to allow only one positive action). As for T_2, to such a tree we cannot associate neither a positive nor a negative slice (the reader should try).

Plan of the paper. The above questions appear as aspects of the same problem. Indeed, to be able to visit all the actions of a slice $\mathfrak{S}$ via normalization means that we have a counter-design which is also a slice, and whose actions are the same as those of $\mathfrak{S}$ with opposite polarity. This is the same as being able to sequentialize those actions into a slice.

It is fairly easy to find a sufficient condition on prefix trees to guarantee we can associate to it a slice. To establish that it is also necessary is harder. It will follow from a study of the properties of normalization paths.

The key point is that if we want that both the design and the counter-design are able to develop the dialogue (interact) on all the addresses, we need a balance between positive and negative addresses. In the following section we will make this precise, establishing three equivalent conditions.

2 Observability Conditions

Polarized trees. Given a tree (and in particular a prefix tree), we can associate a polarity to the nodes. A *polarized tree T* is a tree whose nodes are alternatively labeled with opposite polarities. A polarized tree is positive or negative according to the polarity of the root. We indicate by $N(T)$ the number of negative nodes in T, and by $P(T)$ the number of positive nodes. We indicate by $N_k(T)$ the number of negative nodes of arity k and by $P_k(T)$ the number of positive nodes of arity k. Hence, in particular, $N_0(T)$ is the number of negative leaves. We will omit to explicitly mention T when not ambiguous.

Lemma 1. *Let T be a polarized tree. If the root is negative, the two following conditions are equivalent:*
(i) $N(T) \geq P(T)$ *and (ii)* $N_0 \geq N_2 + 2N_3 + ... + (k-1)N_k + ...$

A similar result holds if the root is positive.

A polarized prefix tree is "almost" a slice, except for the fact that it branches also on negative addresses. Not any prefix tree can be associated to a slice. However, under certain conditions (Proposition 1), we can produce a "sequentialization"of a prefix tree into a slice. We proceed as follows: for each n-ary negative node (with $n \geq 2$), we prune the exceeding $n-1$ positive subtrees, and *graft* them on top of a negative leaf. Condition $(*)$ in Proposition 1 ultimately guarantees that there are "enough leaves" to do so. The crucial point is that the operations we perform preserve the sub-address condition, which is an invariant of any transformation we perform (notice that the condition is satisfied by any prefix tree).

Proposition 1. *Let T be a polarized prefix tree. If T satisfies the condition*

$$(*)\ N(T') \geq P(T') \text{ for any negative subtree } T'$$

then there is a slice $\mathfrak{S}$ with the same polarity as T, such that $\mathbf{T}(\mathfrak{S}) = T$. That is, T is the prefix tree of $\mathfrak{S}$.

We are ready to give two (equivalent) conditions which guarantee that a slice can be visited in a single test. We can already show that they are sufficient to guarantee observability. The fact that they are also necessary will follow from Proposition 3 in the next Section.

Parity. Let $\mathfrak{S}$ be a slice and $\mathbf{T}(\mathfrak{S})$ its prefix tree. $\mathfrak{S}$ satisfies the *parity condition* if in all the *positive* subtrees T' of $\mathbf{T}(\mathfrak{S})$ $P(T') \geq N(T')$.
It is immediate that $P - N \leq 1$.

We can reformulate the previous condition to consider only the positive nodes. We can only look at (positive) leaves and (positive) nodes that are n-ary, for $n \geq 2$.

Leaves. Let $\mathfrak{S}$ be a slice and $\mathbf{T}(\mathfrak{S})$ the corresponding prefix tree. $\mathfrak{S}$ satisfies the *leaves condition* if all the positive subtrees T' of $\mathbf{T}(\mathfrak{S})$ satisfy the following expression: $P_0(T') \geq \sum(i-1)P_i(T')$.

These two conditions guarantee that the addresses composing the slice can be rearranged in a counter-design. We call them *observability conditions.*

Lemma 2 (Girard). *Given two slices $\mathfrak{S}$ and $\mathfrak{T}$ whose sets of actions are dual, then the normalization uses all the actions (Section 3.2 in [7]).*

Proposition 2. *Let $\mathfrak{S}$ be a slice. If $\mathfrak{S}$ satisfies the observability conditions then $\mathfrak{S}$ admits a complete visit by normalization. That is, there is a counter-design $\mathfrak{E}$, such that $[\mathfrak{S} \rightleftharpoons \mathfrak{E}]$ uses all the actions of $\mathfrak{S}$.*

Proof. The parity condition ensures that we can exhibit a slice $\mathfrak{T}$ with the same actions as $\mathfrak{S}$ but opposite polarity. This implies that the normalization between $\mathfrak{S}$ and $\mathfrak{T}$ converges, and uses all of the actions, by Lemma 2.

Normalization paths. Let us study the geometrical properties of a path generated by normalization. Proposition 3, which is the key result of the paper, establishes a relation between the sequential order and the prefix order.

Notations. Given two addresses α, β, we indicate by $\alpha \sqcap \beta$ their longest common prefix. In the same way, since a chronicle is a sequence of addresses, given two chronicles we can consider their longest common prefix. Given a slice $\mathfrak{S}$ and two actions κ_1, κ_2 which are sequentially incomparable (that is neither $\kappa_1 < \kappa_2$ nor $\kappa_2 < \kappa_1$), we indicate by $\kappa_1 \bigwedge_{\mathfrak{S}} \kappa_2$ the last action of the longest common prefix of their chronicles (the chronicle of an action κ is the path from the root to κ).

It is convenient to fix a notation for the *subtrees:* if ξ occurs as a node in the tree T (in the slice $\mathfrak{S}$), then we indicate by T_ξ ($\mathfrak{S}_\xi$) the subtree induced by T (by $\mathfrak{S}$) above ξ.

It is immediate by the branching condition that given a slice $\mathfrak{S}$ and two actions κ_1, κ_2 which are sequentially incomparable, $\kappa_1 \bigwedge_{\mathfrak{S}} \kappa_2$ is a positive node ξ. Therefore, there exist i, j such that $\kappa_1 \in \mathfrak{S}_{\xi i}$, $\kappa_2 \in \mathfrak{S}_{\xi j}, i \neq j$.

Lemma 3. *(i) Given in a slice $\mathfrak{S}$ two actions of addresses σ, τ such that $\sigma < \tau$ (they belong to the same chronicle), either $\sigma \sqsubseteq \tau$ or $\sigma \sqcap \tau$ is a negative address. (ii) Assume that σ, τ are sequentially incomparable in $\mathfrak{S}$. If $\sigma \sqcap \tau$ is positive, then $\sigma \bigwedge_{\mathfrak{S}} \tau = \sigma \sqcap \tau$.*

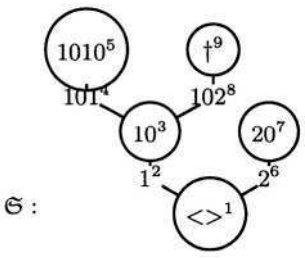


Fig. 2.

Consider the path induced by the closed normalization of two slices $\mathfrak{S}, \mathfrak{T}$. The sequence of actions $p = [\mathfrak{S} \rightleftharpoons \mathfrak{T}]$ induces a path on each slice.

Let us consider a positive action ξ in the slice $\mathfrak{S}$. Each of the negative subaddress ξi which immediately follow ξ induces a subtree $\mathfrak{S}_{\xi_i}$. In general the normalization path does not traverse the slice completing each subtree before entering another subtree (as it would in a preorder traversal). When moving from $\mathfrak{S}_{\xi_i}$ to $\mathfrak{S}_{\xi_j}$ the path will exit $\mathfrak{S}_{\xi_i}$ after a positive action α, then possibly move around outside $\mathfrak{S}_\xi$, and then enter $\mathfrak{S}_{\xi_j}$ on a negative action β.

To fix ideas, let us give a concrete example in Figure 2, where the supscripts on the nodes of $\mathfrak{S}$ indicate the order of visit induced by normalization against the slice $\mathfrak{T}$ consisting only of the linearly ordered actions:

$$<>^-, 1^+, 10^-, 101^+, 1010^-, 2^+, 20^-, 102^+.$$

The path p induced by $\mathfrak{S}$ and $\mathfrak{T}$ is exactly : $\langle <>, 1, 10, 101, 1010, 2, 20, 102 \rangle$. On $\mathfrak{S}$ p exits the subtree $\mathfrak{S}_{10}$ (the subtree induced by the node 10) in 1010, continues outside $\mathfrak{S}_{10}$ and then enters it again on the node 102. Proposition 3 says that the path must leave the subtree $\mathfrak{S}_{\xi i}$ and enter the subtree $\mathfrak{S}_{\xi j}$ on a sub-address of ξ. That is $\alpha \bigwedge_{\mathfrak{T}} \beta = \alpha \sqcap \beta$.

Given a path $p = p_1\ \alpha\ p_2\ \beta\ p_3$ we write $\alpha \ll_p \beta$; we indicate p_2 as $]\alpha, \beta[$. We indicate by $d(\alpha, \beta)$ the number of actions between α and β in p. Such a number is necessarily even if α is positive and β is negative as in the sequel.

Proposition 3. *Let $p = [\mathfrak{S} \rightleftharpoons \mathfrak{E}]$, where $\mathfrak{S}, \mathfrak{E}$ are slices, and let α, β be any two actions such that $\alpha \ll_p \beta$. Assume that in one of the two slices (let us indicate it by $\mathfrak{T}$) (i) α, β belong to distinct chronicles of $\mathfrak{T}$, and (ii) no action κ such that $\alpha \ll_p \kappa \ll_p \beta$ belongs to the subtree induced by $\alpha \bigwedge_{\mathfrak{T}} \beta$. Then $\alpha \sqcap \beta = \alpha \bigwedge_{\mathfrak{T}} \beta$.*

Condition (ii) means that the path p *exits* that subtree $\mathfrak{T}_{\alpha \bigwedge_{\mathfrak{T}} \beta}$ in α and *enters* in β. Notice that only one of the two slices $\mathfrak{S}, \mathfrak{E}$ can satisfy the conditions, because α must be positive and β negative.

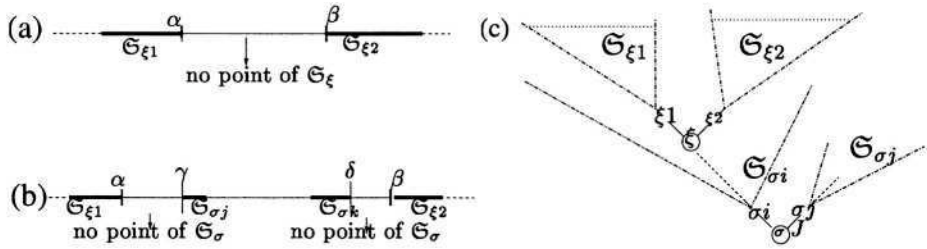


Fig. 3.

Proof. It is enough to show that $\alpha \sqcap \beta$ is always positive in the slice $\mathfrak{T}$ in which we calculate $\alpha \bigwedge_{\mathfrak{T}} \beta$. Then $\alpha \sqcap \beta = \alpha \bigwedge_{\mathfrak{T}} \beta$ by Lemma 3. The proof is by induction on $d(\alpha, \beta)$.

Let $d(\alpha, \beta) = 0$. In the slice $\mathfrak{T}$ (which is either $\mathfrak{S}$ or $\mathfrak{E}$), when p moves from α to β it changes branch. This means that in the counter-design we must have a chronicle $\mathfrak{c}\alpha^- \beta^+$. By Lemma 3, $\alpha \sqcap \beta$ is negative in the counter-design, and then positive in $\mathfrak{T}$, where we calculate $\alpha \bigwedge_{\mathfrak{T}} \beta$.

Let $d(\alpha, \beta) = n \geq 2$. Assume that $\mathfrak{T}$ is $\mathfrak{S}$, and let $\xi = \alpha \bigwedge_{\mathfrak{S}} \beta$. Condition (ii) writes as : $]\alpha, \beta[\cap \mathfrak{S}_\xi = \emptyset$. We are in the situation illustrated by Fig.3(a).

Notice that for any action $\kappa \in]\alpha, \beta[$: (i) $\kappa \notin \mathfrak{S}_\xi$, by hypothesis, and (ii) $\kappa \not< \xi$. In fact all the actions below ξ in $\mathfrak{S}$ are visited before ξ, which in turn is visited before α (because $\tau <_S \sigma$ implies $\tau \ll_p \sigma$); (iii) τ belongs (at least) to one of the subtrees induced by the nodes $\sigma_i <_{\mathfrak{S}} \xi$ (the actions between the root of $\mathfrak{S}$ and ξ).

Let us consider the positive actions $\sigma_i <_{\mathfrak{S}} \xi$ (those below ξ in $\mathfrak{S}$). Let σ be the maximal node such that $\sigma <_{\mathfrak{S}} \xi$ and $\mathfrak{S}_\sigma \cap]\alpha, \beta[\neq \emptyset$. Figure 3 (c) illustrates this. Notice that: (i) $\mathfrak{S}_\xi \subseteq \mathfrak{S}_{\sigma i}$, for some σi sub-address of σ, and (ii) $\mathfrak{S}_{\sigma i} \cap]\alpha, \beta[= \emptyset$, by maximality of σ. Let us call γ the first point of $]\alpha, \beta[$ that belongs to $\mathfrak{S}_\sigma$, and let δ be the last point of $\mathfrak{S}_\sigma$ appearing in $]\alpha, \beta[$. We are then in the situation illustrated by Figure 3 (b).

Since $\alpha \in \mathfrak{S}_{\sigma i}$, $\gamma \in \mathfrak{S}_{\sigma j}$, and $\mathfrak{S}_\sigma \cap]\alpha, \gamma[= \emptyset$, we can apply the inductive hypothesis, obtaining that $\alpha = \sigma i *$, $\gamma = \sigma j *$. In a similar way, we have that $\delta = \sigma k *$ and $\beta = \sigma i *$. We know then that $\sigma i \sqsubseteq \alpha \sqcap \beta$. If we now assume that $\alpha \sqcap \beta$ is negative in $\mathfrak{S}$, then in $\mathfrak{E}$ $\alpha \sqcap \beta$ is positive and, by the proof of Lemma 3 i, α, β belong to distinct subtrees of $\alpha \sqcap \beta$. This means in particular that inside $[\alpha, \beta]$ the normalization path moves from one subtree to the other. By the inductive hypothesis, the exit and the enter points should be sub-addresses of $\alpha \sqcap \beta$ and then of σi. But, $\mathfrak{S}_{\sigma i} \cap]\alpha, \beta[= \emptyset$, giving a contradiction. So $\alpha \sqcap \beta$ is positive in $\mathfrak{S}$ and we are done by Lemma 3,ii.

Proposition 3 allows us to write a third condition. This is more technical, but we will be able to show that it is necessary to observability. The condition really says that we can visit the slice in preorder, persistently completing the visit of a subtree before starting visiting a new one. Conversely, if a slice admits a visit, it admits a visit in preorder.

A *preorder traversal* of an n-ary tree T first visits the root and then visits in preorder each of the subtrees. Consider an n-ary tree T with root ξ and $n \neq 0$; we indicate by $T_1, ...T_n$ its *immediate* subtrees. For each T_i, the last visited node σ is necessarily a leaf, and either it terminates the visit of T itself, or after it we start the visit of another subtree T_j. If this is the case, we call σ a *return point* for the visit of T, and for the root of T. Each leaf but the last one visited is necessarily a return point for exactly one of the internal nodes.

Therefore, any visit in preorder of an n-ary tree T induces a (partial) function G from the leaves to the internal nodes, defined by $\xi = G(\sigma)$ if σ is a return point for ξ. The visit satisfies the following property: *(**) "each internal node has exactly one immediate subtree that does not contain a return point for that node."* Observe that the function G is defined on all the leaves but one.

Conversely, given a tree we can consider a function G from the leaves into the internal nodes which satisfies the property *(**)* above (necessarily, all return points of a node ξ belong to distinct subtrees of ξ). For each node ξ of the tree, let us choose an order between its return points; this then induces a visit in preorder of the tree.

Return. A finite slice $\mathfrak{S}$ satisfies the *return condition* if we can define a partial function G from the positive leaves to the internal nodes, which satisfies the following two properties. (i) $\xi = G(\sigma)$ is a prefix of the leaf σ (ii) each ξ internal node has exactly one immediate subtree which does not contain a leaf σ s.t. $G(\sigma) = \xi$.

We call the leaf σ s.t. $\xi = G(\sigma)$ *return point* for ξ. Observe that the return point of ξ in the sub-tree $\mathfrak{S}_{\xi i}$ is forced to be a sub-address of ξi.

Proposition 4. *If a slice admits a visit by normalization, then it satisfies the return condition.*

Proof. Let $\mathfrak{S}$ be a slice, and ξ a positive node of arity $n \geq 2$. Each of the n subtrees has as root a subaddress ξi; we indicate the subtree by $\mathfrak{S}_{\xi i}$. Since $\mathfrak{S}$ admits a visit, the normalization path must complete the visit of all the subtrees $\mathfrak{S}_{\xi i}$. We can order them accordingly to the order in which their visit is completed. Let $\mathfrak{S}_{\xi j}$ be any of the first $n-1$ subtrees, and α be its last visited action, which must be a leaf. Since there is at least one subtree of ξ whose visit is still to be completed, we are sure that after α the normalization path will enter $\mathfrak{S}_{\xi}$ again. By Proposition 3, α is a sub-address of ξ. Hence we can choose it as the return point for ξ in the subtree $\mathfrak{S}_{\xi j}$.

The return condition really means that we can visit the slice in preorder, *for a suitable ordering of the subtrees,* persistently completing the visit of a subtree before starting a new one.

Proposition 5. *If a slice $\mathfrak{S}$ satisfies the return condition, any ordering of the return points induces a preorder traversal of $\mathfrak{S}$ that can be realized by normalization of $\mathfrak{S}$ with a counter-design.*

Lemma 4. Return *condition* $\Rightarrow$ Leaves *condition.*

Putting it all together, we have that $\mathfrak{S}$ *admits a visit* by normalization $\Rightarrow$ *Return* $\Rightarrow$*Leaves* $\Rightarrow$ *Parity* $\Rightarrow$ $\mathfrak{S}$ *admits a visit* by normalization.

Proposition 6. *The three conditions of* parity, leaves and return *are equivalent and characterize the slices which admit a visit by normalization*

We also have enough information to characterize the prefix trees to which we can associate two slices $\mathfrak{S}, \mathfrak{T}$, such that $\mathfrak{S} \perp \mathfrak{T}$.

Proposition 7. *To a prefix tree T we can associate both a positive slice* $\mathfrak{S}$ *and a negative slice* $\mathfrak{E}$ *s.t.* $|\mathfrak{S}| = |\mathfrak{E}| = T$ *iff, as soon as we fix a polarization:*

(i) in all the negative subtrees T', $N(T') \geq P(T')$, *and*

(ii) in all the positive subtrees T'', $P(T') \geq N(T')$. We can check that the conditions hold *for all the subtrees* with a single *postorder* traversal of *T*.

It is immediate that for all subtrees T' we have $|N(T') - P(T')| \leq 1$.

3 Conclusions and Further Work

The question we have addressed could be put in a more general perspective: "given a piece of code P, and an enviroment in which it is executed, how many lines of P are actually run or visited during its execution?" It would be interesting to study possible relations with static analysis or dead code elimination.

Ideally, in Ludics we should be able to determine, test and express interactively all properties we require of designs. For this reason it is important to know what can be observed at each single test. The designs that can be visited in a single run of normalization represent the *primitive units of observability.*

We also have *a notion of slice which abstracts over sequentiality details.* We expect this to be a first step towards a more asynchronous or "concurrent" notion of interaction. The geometrical properties we have highlighted are a base and a guide for further developments also in this perspective.

We believe that a contribution of this paper is to evidence how the relation between the two orders on a slice (sequential and spatial order) forces structure and remarkable properties. Normalization ultimately relies on such properties, which will be important if we want to generalize the structures.

Ludics, as introduced in [7], accounts for sequential computation. However the sequentiality assumptions are not essential to the theory, and it seems natural to extend it to a concurrent setting. Progress on this issue is reported in [6].

Acknowledgments. I wish to thank Jean-Yves Girard for his insightful advice and Pierre-Louis Curien for his many suggestions and deep comments.

References

1. P.-L. Curien. Abstract Böhm trees. *Mathematical Structures in Computer Science,* 8(6), 1998.

2. V. Danos and L. Regnier. Proof-nets and the Hilbert space. In R. L. Girard J.-Y., Lafont Y., editor, *Advances in Linear Logic,* number 222 in London Mathematical Society Lecture Notes Series. Cambridge University Press, 1995.
3. C. Faggian. *On the Dynamics of Ludics. A Study of Interaction.* PhD thesis, Université Aix-Marseille II, 2002.
4. C. Faggian. Travelling on designs: ludics dynamics. In *CSL'02,* volume 2471 of *LNCS.* Springer Verlag, 2002.
5. C. Faggian and M. Hyland. Designs, disputes and strategies. In *CSL'02,* volume 2471 of *LNCS.* Springer Verlag, 2002.
6. C. Faggian and F. Maurel. Ludics on graphs, towards concurrency. draft.
7. J.-Y. Girard. Locus solum. *Mathematical Structures in Computer Science,* 11:301–506, 2001.

Easily Refutable Subformulas of Large Random 3CNF Formulas

Uriel Feige and Eran Ofek

Weizmann Institute of Science,
Department of Computer Science and Applied Mathematics,
Rehovot 76100, Israel,
{uriel.feige,eran.ofek}@weizmann.ac.il

Abstract. A simple nonconstructive argument shows that most 3CNF formulas with *cn* clauses (where c is a large enough constant) are not satisfiable. It is an open question whether there is an efficient refutation algorithm that for most formulas with *cn* clauses proves that they are not satisfiable. We present a polynomial time algorithm that for most 3CNF formulas with $cn^{3/2}$ clauses (where c is a large enough constant) finds a subformula with $O(c^2 n)$ clauses and then proves that this subformula is not satisfiable (and hence that the original formula is not satisfiable). Previously, it was only known how to efficiently certify the unsatisfiability of random 3CNF formulas with at least $\mathbf{poly}(\log(n)) \cdot n^{3/2}$ clauses. Our algorithm is simple enough to run in practice. We present some preliminary experimental results.

1 Introduction

A 3CNF formula ϕ over n variables is a set of m clauses, each one contains exactly 3 literals of three different variables. A formula ϕ is satisfiable if there exists an assignment to its n variables such that in each clause there is at least one literal whose value is true. The problem of deciding whether an input 3CNF formula ϕ is satisfiable is NP-hard, but this does not rule out the possibility of designing a good heuristic for it. A heuristic for satisfiability may try to find a satisfying assignment for an input formula ϕ, in case one exists. A refutation heuristic may try to prove that no satisfying assignment exists. In this paper we present an algorithm which tries to refute an input formula ϕ. The algorithm has one sided error, in the sense that it will never say "unsatisfiable" on a satisfiable formula, but for some unsatisfiable formulas it will fail to output "unsatisfiable". It then follows that for a formula ϕ on which the algorithm outputs 'unsatisfiable', its execution on ϕ is a witness for the unsatisfiability of ϕ.

How does one measure the quality of a refutation heuristic? A possible test may be to check how good the heuristic is on a random input. But then, how do we generate a random unsatisfiable formula? To answer this question we review some known properties of random 3CNF formulas. The satisfiability property has the following interesting threshold behavior. Let ϕ be a random 3CNF formula with n variables and *cn* clauses (each new clause is chosen independently and uniformly from the set of all possible clauses). As the parameter c is increased, it becomes less likely that ϕ is satisfiable, as there are more constraints to satisfy. In [6] it is shown that there exists c_n such that

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 519–530, 2004.

for $c < c_n(1-\epsilon)$ almost surely ϕ is satisfiable, and for $c > c_n(1+\epsilon)$, ϕ is almost surely unsatisfiable (for some ϵ which tends to zero as n increases). It is also known that $3.52 < c_n < 4.596$ [12],[10],[11]. We will use random formulas with cn clauses (for $c > c_n(1+\epsilon)$) to measure the performance of a refutation heuristic. Specifically, the refutation heuristic is considered good if for some $c > (1+\epsilon)c_n$ it almost surely proves that a random formula with cn clauses is unsatisfiable.

Notice that for any n, as c is increased (for $c > c_n(1+\epsilon)$), the algorithmic problem of refutation becomes less difficult since we can always ignore a fixed fraction of the clauses. The following question is still open: how small can c be so that there is still an efficient algorithm which almost surely refutes random 3CNF formulas with cn clauses (c may also be an increasing function of n).

A possible approach for refuting a formula ϕ is to find a resolution proof for the unsatisfiability of ϕ. However, Chvatal and Szemeredi [2] proved that a resolution proof of a random 3CNF formula with linear number of clauses is almost surely of exponential size. A result of a similar flavor for denser formulas was given by Ben-Sasson and Wigderson [1] who showed that a random formula with $n^{3/2-\epsilon}$ clauses almost surely requires a resolution proof of size $2^{\Omega(n^{\epsilon/(1-\epsilon)})}$. These lower bounds imply that finding a resolution proof for a random formula is computationally inefficient.

A simple refutation algorithm can be used to refute random instances with $\Omega(n^2)$ clauses. This is done by fixing a variable x, and taking all the clauses which contain it. Fixing x to be true leaves about half of the selected clauses as a random 2CNF formula with $\Omega(n)$ clauses, which can be proved to be unsatisfied by a polynomial running time algorithm. The same can be done when fixing x to be false.

A new approach introduced by Goerdt and Krivelevich in [8], gave a significant improvement and reduced the bound into $(\log n)^7 \cdot n^k$ clauses for efficient refutation of 2kCNF formulas. This approach was later extended in [7],[9] to handle also random 3CNF formulas with $n^{3/2+\epsilon}$, $\mathrm{poly}(\log n)\cdot n^{3/2}$ clauses respectively. In [3], [5] it is shown how to efficiently refute a random 2kCNF instances with at least cn^k clauses.

Further motivation for studying efficient refutability of random 3CNF formulas is given in [4]. There it is shown that if there is no polynomial time refutation heuristic that works for most 3CNF formulas with cn clauses (where c is an arbitrarily large constant) then certain combinatorial optimization problems (like minimum graph bisection, the 2-catalog segmentation problem, and others) have no polynomial time approximation schemes. It is an open question whether it is NP-hard to approximate these problems arbitrarily well.

Our refutation algorithm is based on similar techniques to those that appear in [7], [3], [4], but it has some advantages. Both the algorithms in [7], [9] and our algorithm perform eigenvalue computations on some random matrices derived from the random input formula ϕ. However, our matrices are much smaller (of order n rather than n^2), making the computational task easier. Moreover, the structure of our matrices is simpler, making the analysis of our algorithm simpler, and easier to apply also to formulas with fewer clauses than those in [7],[9]. As a result of this simplicity, we can show that our algorithm refutes formulas with $cn^{3/2}$ clauses, whereas the algorithms given in [7],[9] are claimed only to refute formulas with $\Omega(n^{3/2+\epsilon})$, $\Omega(\mathrm{poly}(\log n)\cdot n^{3/2})$

clauses respectively. An implementation of our algorithm refuted a random formula with 50000 variables and 27335932 clauses (see details in section 4).

In some other respects, our algorithm is more limited then the algorithms in [7], [9]. An algorithm is said to provide *strong refutation* if it shows not only that the input 3CNF formula is not satisfiable, but also that every assignment to the variables fails to satisfy a constant fraction of the clauses. Our refutation algorithm does not provide a strong refutation. It is plausible that the algorithms of [7],[9] (or a variant of them) does provide a strong refutation, though this issue is not explicitly addressed in [7],[9]. The ability to perform strong refutation is an important issue, and its relation to approximability is discussed in [4].

2 Preliminaries

2.1 The Random Model

Definition 1. *In the $F_{n,p}$ model a random 3CNF formula is generated in the following way: each clause out of the $2^3\binom{n}{3}$ possible clauses is chosen independently with probability of p.*

Although we concentrate on a specific random model for generating random formulas, our algorithm succeeds also on other related random models. For example, it is not hard to see that our algorithm works also if we generate a random formula by picking exactly $cn^{3/2}$ clauses independently at random, with (or without) replacement. Details are omitted.

2.2 Efficient Certification of a Property

An important concept which will be frequently used is the concept of *efficient certification.* Let P be some property of graphs/formulas or any other combinatorial object. An algorithm A *certifies* the property P if the following holds:

1. On any input instance ϕ the algorithm returns either 'accept' or 'don't know'.
2. *Soundness:* The algorithm never outputs 'accept' on an instance ϕ which does not have the property P. The algorithm may output 'don't know' on an instance x which has the property P (the algorithm has one sided error).

We will use certification algorithms on random instances of formulas/graphs taken from some probability space C. We shall consider properties that are almost surely true for the random object taken from C. A certification algorithm is *complete* with respect to the probability space C if it almost surely outputs 'accept' on an input taken from C.

The computationally heavy part of our algorithm is certifying that two different graphs derived from the random formula ϕ do not have a large cut. One of these two graphs is random, and the other is a multigraph that is the union of 6 graphs, where each of these graphs by itself is essentially random, but there are correlations among the graphs. A cut in a graph is a partition of its vertices into two sets. The size of the cut is the number of edges with one endpoint in each part. A certification algorithm for verifying

that an input graph with m edges has no cut significantly larger than $m/2$ is implicit in [13]. This algorithm is based on semi-definite programming; if the maximum cut in the input graph is of size at most $m(1/2+\epsilon)$, then the algorithm outputs a certificate that the maximum cut in G is bounded by $m(1/2+\delta(\epsilon))$, where $\delta(\epsilon) \longrightarrow 0$ as $\epsilon \longrightarrow 0$. A computationally simpler algorithm can be applied if the graph is random. In [5] it is shown how to certify that in a random graph taken from $G_{n,d/n}$ the size of the maximum cut is bounded by $dn(1/4+\frac{\theta(1)}{\sqrt{d}})$, thus bounding the maximum cut by $m(1/2+\epsilon)$ when d is large enough. This is done by removing the vertices of highest degree from G, and then computing the most negative eigenvalue of the adjacency matrix of the resulting graph.

2.3 An Overview of Our Refutation Algorithm

Our algorithm builds on ideas taken from earlier work ([8], [7], [4], [5], [3], [9]). This section gives an informal overview of the algorithm at a fairly detailed level. Other sections of this manuscript fill in the formal details.

The input to the algorithm is a random 3CNF formula ϕ with n variables and $m = cn^{3/2}$ clauses, where c is a large enough constant. The algorithm first greedily extracts from ϕ a subformula ϕ'. This is done as follows. We say that two clauses are *matched* if they differ in their first literal, but agree on their second literal and on their third literal. For example, the clauses $(x_1 \vee \bar{x}_2 \vee x_3)$ and $(x_4 \vee \bar{x}_2 \vee x_3)$ are matched. ϕ' is constructed by greedily putting into ϕ' pairs of clauses that form a match, until no further matches are found in ϕ. Let m' be the number of clauses in ϕ'. A simple probabilistic argument shows that we can expect $m' = \Theta(m^2/n^2) = \Theta(c^2 n)$. Moreover, ϕ' is essentially a union of two random (but correlated) formulas ϕ_1 and ϕ_2 (each containing one clause from every pair of clauses that are matched in ϕ'). Our algorithm will now ignore the rest of ϕ, and refute ϕ'. As explained, ϕ' is a union of two random formulas. Here we use an observation that is made in [4], that we shall call the 3XOR principle.

The 3XOR principle. In order to show that a random 3CNF formula is not satisfiable, it suffices to strongly refute it as a 3XOR formula.

Let us explain the terms used in the 3XOR principle. A clause in a 3XOR formula is satisfied if either one or three of its literals are satisfied. A strong refutation algorithm is one that shows that every assignment to the variables leaves at least a constant fraction of the clauses not satisfied (as 3XOR clauses, in our case).

A proof of the 3XOR principle is given in [4]. We sketch it here, and give it in more details in Section 3. Observe that in a random formula every literal is expected to appear the same number of times, and if the number of clauses is large enough, then things behave pretty much like their expectation. As a consequence, every assignment to the variables sets roughly half the occurrences of literals to true, and roughly half to false. Hence every assignment satisfies on average 3/2 literals per clause. Moreover, this property is easily certifiable, by summing up the number of occurrences of the n most popular literals.

Given that every assignment satisfies on average 3/2 literals per clause, let us consider properties of satisfying assignments (if such assignments exist). The good option is that they satisfy one literal in roughly 3/4 of the clauses, three literals in roughly 1/4 of the clauses, and 2 literals in a negligible fraction of the clauses. This keeps the average roughly at 3/2, and indeed nearly satisfies the formula also as a 3XOR formula, as

postulated by the XOR principle. The bad option (which also keeps the average at 3/2) is that the fraction of clauses that are satisfied three times drops significantly below 1/4, implying that significantly more than 3/4 of the clauses are satisfied either once or twice, or in other words, as a 3NAE (3-"not all equal" SAT) formula. But here, let us combine two facts. One is that for a random large enough formula, every assignment satisfies roughly 3/4 of the clauses as a 3NAE formula. The other (to be explained below) is that there are known efficient algorithms for certifying that no assignment satisfies more than $3/4+\epsilon$ fraction of the clauses of a 3NAE formula. Hence for a random 3CNF formula, one can efficiently certify that the bad option mentioned above does not occur.

Having established the 3XOR principle, the next step of our algorithm makes one round of Gaussian elimination. That is, under the assumption that we are looking for near satisfiability as 3XOR (which is simply a linear equation modulo 2), we can add clauses modulo 2. Adding (modulo 2) two matched clauses, the common literals drop out, and we get a clause with only two literals whose XOR is expected to be 0, namely, a 2EQ clause (EQ for equality). For example, from the clauses $(x_1 \oplus \bar{x}_2 \oplus x_3)$ and $(x_4 \oplus \bar{x}_2 \oplus x_3)$ one gets the clause $(x_1 = x_4)$. Doing this for all pairs of matched clauses in ϕ', we get a random 2EQ formula ϕ_{2eq}. If ϕ' was nearly satisfiable as 3XOR, then ϕ_{2eq} must be nearly satisfiable as 2EQ. But if ϕ' is random, then ϕ_{2eq} too is essentially a random 2EQ formula. For such formulas, every assignment satisfies roughly half the clauses. Moreover, there are known algorithms that certify this (to be explained shortly). Hence we can strongly refute ϕ_{2eq} as 2EQ, implying strong refutation of ϕ' as 3XOR, implying strong refutation of ϕ' as 3SAT, implying refutation (though not strong refutation) of ϕ as 3SAT.

Let us briefly explain here the major part that we skipped over in the description of our algorithm, namely, how to certify that a random 2EQ formula is not $1/2+\epsilon$ satisfiable, and how to certify that a random 3NAE formula is not $3/4+\epsilon$ satisfiable. In both cases, we reduce the certification problem to certifying that certain random graphs do not have large cuts, and then use the certification algorithms mentioned in Section 2.2. (The principle of refuting random formulas by reduction to random graph problems was introduced in [8].)

To strongly refute random 2EQ formulas, we negate the first literal in every clause, getting a 2XOR formula. Now we construct a graph whose vertices are the literals, and whose edges are the clauses. A nearly satisfying 2XOR assignment naturally partitions the vertices into two sides (those literals set to true by the assignment versus those that are set to false), giving a cut containing nearly all the edges. On the other hand, if the original 2EQ formula was random, then so is the graph, and it does not have any large cut. As explained in Section 2.2, we can efficiently certify that the graph does not have a large cut, thus strongly refuting the 2XOR formula, and hence also strongly refuting the original 2EQ formula.

To strongly refute random 3NAE formulas, we again consider a max-cut problem on a graph (in fact, a multigraph, as there will be parallel edges) whose vertices are the literals. From each 3NAE clause we derive three edges, one for every pair of literals. For example, from the NAE clause $(x_1, \bar{x}_2, x_3)$ we get the edges $(x_1, \bar{x}_2)$, $(\bar{x}_2, x_3)$ and (x_3, x_1). It is not hard to see that if a $3/4+\epsilon$ fraction of the 3NAE clauses are satisfied as 3NAE, than a $\frac{2}{3}(3/4+\epsilon) = 1/2 + 2\epsilon/3$ fraction of the edges of the graph are cut by the partition induced by the corresponding assignment. Note that in our case (of ϕ' that

is the union of random ϕ_1 and random ϕ_2) this graph is essentially a union of 6 random graphs: 3 graphs derived from the clauses of ϕ_1 (one with edges derived from the first two literals in every clause, one with edges derived from the last two literals, and one from the first and third literal), and 3 graphs derived from ϕ_2. Hence it is not expected to have a cut containing significantly more than half the edges. One may certify that this is indeed the case either by using the algorithm of [13] on the whole graph, or by using the algorithm of [5] on each of the 6 random graphs separately.

Summarizing, our refutation algorithm extracts from ϕ a subformula ϕ' (composed of matched pairs of clauses), checks that in ϕ' almost all literals appear roughly the same number of times, derives from ϕ' certain graphs on $2n$ vertices and certifies that they do not have large cuts (e.g., by computing the most negative eigenvalue of their adjacency matrices). The combination of all this evidence forms a proof that ϕ is not satisfiable. If ϕ is random and large enough ($cn^{3/2}$ clauses), then almost surely the algorithm will indeed manage to collect all the desired evidence.

3 The Refutation Algorithm

The input formula ϕ is taken from $F_{n,p}$ where $p = \frac{c}{n^{3/2}}$. For convenience we will assume that the clauses of ϕ appear in a random order and that the order of literals inside each clause is random (this assumption is reasonable because we can permute the clauses and the literals inside each clause before handling ϕ). We will use $(?, w, l)$ to denote a clause in which the second and the third literals are w, l respectively and the first literal can be any literal. Let ϕ' be a subformula of ϕ constructed in the following way: for every pair of literals (w, l) we count the number of clauses in ϕ of the form $(?, w, l)$. If this number is two or more, then we take into ϕ' the first two appearances of these clauses (preserving the order of appearance). If the number of such clauses is one or less, we don't add anything to ϕ'. From each pair of clauses $(x, w, l), (y, w, l)$ in ϕ' we take the first one to the set ϕ_1 and the second to the set ϕ_2. Each of ϕ_1, ϕ_2 is a random formula, though clauses in ϕ_i are not completely independent of each other: if a clause (x, y, z) appears, then the clause (t, y, z) cannot appear. From now own we will forget ϕ and concentrate in refuting ϕ'. Before specifying the algorithm, we introduce additional notation and definitions which will ease the description of the algorithm.

Definition 2. *Let ϕ' be a 3CNF formula with m pairs of matched clauses. Then the following graphs $G_1^1, G_2^1, G_3^1, G_1^2, G_2^2, G_3^2$ and G_{2eq} are defined as follows.*
G_{2eq}: *A graph whose vertices are the literals and whose edges are the following: each matched pair from ϕ_1, ϕ_2, say $(x, w, l), (y, w, l)$ respectively, induces exactly one edge $(\bar{x}, y)$. Notice that we negate the literal which corresponds to the clause of ϕ_1.*
G_i^1: *A graph whose vertices are the literals and whose edges are the following: for each clause in ϕ_1 we omit the i-th coordinate and get a set of two literals which induces an edge of G_i.*
G_i^2: *This graph is similar to G_i^1 but its edges are induced by ϕ_2.*

Definition 3. *Let ϕ be a formula with n variables and m clauses. The* imbalance *of a variable i (denoted by Im_i) is the difference in absolute value between the number of*

times it appears with positive polarity and the number of times it appears with negative polarity. The normalized imbalance *of* ϕ *is* $\frac{\sum_{i=1}^{n} Im_i}{3m}$.

If the normalized imbalance of ϕ is bounded by δ, then ϕ is δ-*balanced.*

Definition 4. *A 3CNF formula* ϕ *has the* $(1-\epsilon)$ 3*XOR property if for every assignment A satisfying* ϕ *as 3CNF, at least* $1-\epsilon$ *fraction of the clauses are satisfied as 3XOR.*

Definition 5. *A graph is said to have a* δ-*cut if there is a partition of it's vertices into two disjoint sets such that at least* δ-*fraction of the edges cross this partition.*

The number of matched pairs is denoted by m (in section 2.3 we used m to denote the number of clauses in ϕ. From here on, m will denote the number of matched pairs in ϕ'). In lemma 2 it is shown that almost surely the number of matched pairs $m = 8c^2 \pm 10c\sqrt{n}$. The refutation algorithm does the following steps ($\epsilon < 1/22$ is some positive fixed constant):

1. Certify that ϕ' has the $(1-5\epsilon)$ 3XOR property. This is done by certifying the $(1-5\epsilon)$ 3XOR property for ϕ_1, ϕ_2 separately. If both ϕ_1, ϕ_2 have this property then also ϕ' has this property. Specifically, verify the following (for $i = 1, 2$):
   (a) ϕ_i is ϵ-balanced.
   (b) Each of the graphs G_1^i, G_2^i, G_3^i has a maximum cut bounded by $m(1/2+\epsilon)$.
2. Certify that G_{2eq} has a maximum cut with at most $(\frac{1}{2}+\epsilon)m$ edges.

The algorithm rejects if one of the above steps fails. We will show the following facts: (1) a 3CNF formula ϕ_i with m clauses which satisfies conditions (a),(b) from step (1) of the refutation algorithm has the $(1-5\epsilon)$ 3XOR property; this follows from Lemma 1. (2) The random graphs $G_1^1, G_2^1, G_3^1, G_1^2, G_2^2, G_3^2, G_{2eq}$ each has a maximum cut (almost surely) bounded by $m(1/2+\epsilon_1)$ for small $\epsilon_1 > 0$. This is done at Theorem 2. (3) There is an efficient algorithm which certifies that each of $G_1^1, G_2^1, G_3^1, G_1^2, G_2^2, G_3^2, G_{2eq}$ has a maximum cut bounded by $m(1/2+\epsilon)$; this is shown in Theorem 3. (4) The above refutation algorithm almost surely accepts a random formula ϕ (Corollary 2). (5) A satisfiable formula will be rejected by the algorithm (Corollary 1).

Theorem 1. *Let* ϕ' *be a 3CNF formula composed of pairs of matched clauses. Denote by* G_{2eq} *be the graph induced by* ϕ' *as described by the refutation algorithm.* ϕ' *is not satisfiable if all the following conditions hold:*

1. ϕ' *has the* $(1-\gamma)$**3XOR** *property,*
2. G_{2eq} *has no* $(1/2+\epsilon)$-*cut.*
3. $\epsilon + 2\gamma < 1/2$.

Proof. We shall show that if ϕ' is satisfiable and has the $(1-\gamma)$**3XOR** property, then G_{2eq} has a $(1-2\gamma)$-cut. Combined with the fact that G_{2eq} has no $1/2+\epsilon$ cut we derive a contradiction (since $\epsilon + 2\gamma < 1/2$). Consider the cut induced on the vertices of G_{2eq} by a satisfying assignment A (where in one side there are all the literals whose value is true and in the other side there are all the literals whose value is false), $A(x)$ denotes the value of the literal x induced by the assignment A. By the 3XOR property

of ϕ', all but γ fraction of the clauses of ϕ' are satisfied as 3XOR clauses. Almost every pair of matched clauses induces an edge which crosses the cut: Let $(x, w, l), (y, w, l)$ be a pair such that both (x, w, l) and (y, w, l) are satisfied as 3XOR. It holds that: $A(x) + A(y) + 2(A(w) + A(l)) = 0 \bmod 2$. Thus exactly one of the literals $\bar{x}, y$ is true and the other is false (under the assignment A), and the edge $(\bar{x}, y)$ induced by this pair of clauses crosses the cut. It then follows that at least $1 - 2\gamma$ fraction of the edges in G_{2eq} are cut edges. □

Corollary 1 (Soundness). *Let ϕ be a 3CNF formula. If the refutation algorithm accepts ϕ, then ϕ is not satisfiable.*

Proof. Let ϕ be a formula accepted by the refutation algorithm. The algorithm verified that the extracted subformula ϕ' has the $(1 - 5\epsilon)$**XOR** property and that G_{2eq} has no $(1/2 + \epsilon)$-**cut.** Since $\epsilon < 1/22$ it follows that $\epsilon + 2 \cdot 5\epsilon < 1/2$, thus by theorem 1 ϕ' is not satisfiable. □

Lemma 1 (The 3XOR lemma). *Let ϕ be a formula with m clauses and n variables. Let G_1, G_2, G_3 be the following projection graphs: G_i has $2n$ vertices associated with the literals of ϕ. The edges of G_i are derived from ϕ by removing the i-th coordinate from each clause of ϕ. If the following hold:*

1. *ϕ is δ-balanced.*
2. *The maximum cut in each G_i is bounded by $m(1/2 + \varepsilon)$.*

then ϕ has the $(1 - \frac{3}{2}(\delta + 2\varepsilon))$3XOR property.

Proof. The proof of this lemma appears at [4]; also a similar version of this lemma (for denser random 2kCNF formulas) appears at [3]. We repeat the proof for the sake of self-containment.

Let A be a satisfying assignment. We bound from above the number of satisfied appearances of literals. The assignment which maximizes the number of satisfied appearances of literals is the 'majority vote': a variable x_i is assigned 'true' iff it appears more times with positive polarity than with negative polarity. Using this assignment the total number of satisfied appearances is $\frac{3m + \sum_{i=1}^{n} Im_i}{2}$ (where Im_i denotes the imbalance of variable i). It follows that on average each clause is satisfied at most $3/2 + \frac{\sum_{i=1}^{n} Im_i}{2m} = \frac{3}{2}(1 + \delta)$.

We next show that the fraction of clauses satisfied as 3AND is at least $\frac{1}{4} - \frac{3}{2}\varepsilon$. Equivalently it is enough to show that the fraction of clauses satisfied as 3NAE denoted by β is bounded by $\frac{3}{4} + \frac{3}{2}\varepsilon$ (since A is a satisfying assignment). Consider the graph $G_{1+2+3} = G_1 + G_2 + G_3$ induced by taking the union of the edges of G_1, G_2, G_3. We remind the reader that each clause of ϕ contributes a "triangle"of 3 edges to G_{1+2+3} (e.g. the clause $(x \vee \bar{y} \vee z)$ contributes the edges $(x, \bar{y}), (\bar{y}, z), (x, z)$). Consider the cut induced by the satisfying assignment A. Each clause satisfied as 3NAE contributes exactly 2 edges to the cut, thus this cut has at least $2\beta m$ edges. But the edges of G_{1+2+3} are exactly the union of the edges of G_1, G_2, G_3 (each of them has m edges), and each G_i has no $(1/2 + \varepsilon)$-**cut.** Hence also G_{1+2+3} has no $(1/2 + \varepsilon)$-**cut** (at least 1/3 of the

edges of the maximum cut in G_{1+2+3} belong to G_i for some $i \in \{1,2,3\}$). It follows that $2\beta m \leq (1/2+\varepsilon)3m$, giving $\beta \leq \frac{3}{4}+\frac{3}{2}\varepsilon$ as needed.

It remains to show that all but a small fraction of the clauses are satisfied as 3XOR by A. Denote by $\alpha_1, \alpha_2, \alpha_3$ the fraction of clauses which are satisfied exactly once, exactly twice and exactly 3 times respectively ($\sum_{i=1}^{3}\alpha_i = 1$). We already know that $\alpha_3 \geq \frac{1}{4}-\frac{3}{2}\varepsilon$ and that each clause is satisfied at most $\frac{3}{2}+\delta$ times on average, thus: $\frac{3}{2}(1+\delta) \geq 3\cdot\alpha_3 + 2\cdot\alpha_2 + 1\cdot\alpha_1$. Substituting $\alpha_1 = (1-\alpha_3-\alpha 2)$ and α_3 with $\frac{1}{4}-\frac{3}{2}\varepsilon$ yields that $\alpha_2 \leq \frac{3}{2}(\delta+2\varepsilon)$. □

In the following theorems $0 < \epsilon_1 < \epsilon_2 < 1/22$. ϵ_1 can be made arbitrarily small by increasing the density parameter c.

Theorem 2. *Let ϕ be a random CNF formula chosen from $F_{n,c/n^{3/2}}$ (with large enough c). Let ϕ', ϕ_1, ϕ_2 be subformulas derived from ϕ as described by the refutation algorithm. Then with high probability over the choice of ϕ, the subformula ϕ_i is ϵ_1-balanced, and none of the respective graphs G_1^i, G_2^i, G_3^i and G_{2eq} has a $(1/2+\epsilon_1)$-cut.*

It is well known that a random graph has no $(1/2+\epsilon_1)$-cut, and that a random formula is ϵ_1-balanced. The distributions of G_j^i, ϕ_i are "close enough" to the standard models of random graphs/formulas respectively, so that the proof techniques used for the random cases can be applied also in our case. Proof details are omitted due to lack of space.

Theorem 3. *There is a polynomial time algorithm that checks whether a 3CNF formula is ϵ-balanced. There is a polynomial time algorithm that for every graph that does not have a $(1/2+\epsilon_1)$-cut certifies that the size of the maximum cut is at most $(1/2+\epsilon_2)$.*

Proof. Checking that a formula ϕ with m clauses and n variables is ϵ-balanced is done by counting positive and negative appearances for every variable, computing its imbalance and averaging.

An algorithm for certifying a bound on the maximum cut of a graph is given in [13]. Given a graph with m edges whose maximum cut is bounded by $m(1/2+\epsilon_1)$ this algorithm produces a proof that the input graph has no cut of cardinality $m(1/2+\epsilon_2)$. This algorithm has the property that $\epsilon_2 \longrightarrow 0$ as $\epsilon_1 \longrightarrow 0$. □

Corollary 2 (Completeness). *The refutation algorithm almost surely accepts a random formula ϕ taken from $F_{n,c/n^{3/2}}$ for big enough c (c is big enough so that $\epsilon_2 < \frac{1}{22}$).*

Proof. We set the parameter ϵ from the refutation algorithm to be equal to ϵ_2 (by taking the constant c large enough we can make ϵ_1, ϵ_2 arbitrarily small). Let ϕ', ϕ_1, ϕ_2 be the subformulas derived from ϕ as described by the refutation algorithm. By theorem 2 almost surely ϕ_i is ϵ_1-balanced, and none of the graphs $G_1^i, G_2^i, G_3^i, G_{2eq}$ induced by ϕ has a $(1/2+\epsilon_1)$-cut. The refutation algorithm which uses the algorithms from theorem 3 will succeed in verifying that each of the graphs $G_1^i, G_2^i, G_3^i, G_{2eq}$ has no $(1/2+\epsilon)$-cut and that ϕ_i is ϵ-balanced, since $\epsilon_1 < \epsilon_2 = \epsilon$. □

Lemma 2. *Let ϕ' be the formula derived from ϕ by the refutation algorithm (where ϕ is taken from $F_{n,\frac{c}{n^{3/2}}}$). Almost surely the number of matched pairs in ϕ' is $8c^2n \pm 10\sqrt{c^2n}$.*

Proof (sketch). The probability that $(?, w, l)$ appears at least twice in ϕ is $\frac{2c^2}{n}(1 - O(\frac{c}{\sqrt{n}}))$, thus the expected number of matched clauses is roughly $8c^2n$. Using large deviation laws (e.g. the Chernoff's bound) we derive that the number of matched pairs is concentrated around its expectation. □

4 Practical Considerations for the Refutation Algorithm

Recall that our refutation algorithm extracts from ϕ a subformula ϕ' that contains matched pairs of clauses, and then refutes ϕ'. The longer ϕ' is, the easier it is to refute it. For simplicity, we matched a pair of clauses only if they agreed on their last two literals. Moreover, every clause of ϕ participated in at most one pair of matched clauses in ϕ', even though a clause may be eligible to participate in more than one matched pair. In practical implementations, it is advantageous not to have these restrictions, and thus get a longer formula ϕ'. In particular, we may allow the same clause to participate in several pairs of matched clauses, by duplicating it. More importantly, we may match any two clauses that share two variables (regardless of the polarity of the variables, and of their location within the clauses). For example, the two clauses (x, w, l) and $(\bar{w}, l, y)$ can be matched. If ϕ' is satisfied by an assignment A that has the 3XOR property, then one step of Gaussian elimination gives in this case $A(x) + A(w) + A(l) + A(y) + A(\bar{w}) + A(l) = 0 \ mod\ 2$, thus $A(x) + A(y) = 1 \ mod\ 2$. Hence we will associate the edge (x, y) with this pair of matched clauses so that the edge induced by this pair in G_{2eq} will cross the cut which corresponds to the assignment A. Using the principles above, the number of pairs of matched clauses that one expects to extract from a random formula of length $cn^{3/2}$ is roughly $\binom{3cn^{3/2}}{2}/\binom{n}{2} \simeq 9c^2n$.

We used the principles above to implement the algorithm in practice. In the current implementation, the problem of refuting ϕ' is reduced to strong refutation of two 2XOR formulas. We performed 2 eigenvalue computations on matrices of size $n \times n$, whereas the original refutation algorithm performed eigenvalue computations on matrices of size $2n \times 2n$. Our implementation uses the conditions of Theorem 4 to refute ϕ'. Before stating Theorem 4 we need the following definition.

Definition 6. *Let ϕ be a 2XOR formula with m clauses and n variables. A_ϕ is the following symmetric matrix associated with ϕ. Initially A_ϕ is the zero matrix. For each clause of the forms $(x, \bar{y})$ or $(\bar{y}, x)$ we add $+1$ to positions $A(x, y), A(y, x)$. For each clause of the forms (x, y) or $(\bar{x}, \bar{y})$ we add -1 to positions $A(x, y), A(y, x)$.*

A similar matrix can be defined for a 2EQ formula, just by reducing the 2EQ formula into a 2XOR formula.

Theorem 4. *Let ϕ' be a 3CNF formula with m pairs of matched clauses and n variables. Let ϕ_{2xor} be the 2XOR formula induced by replacing each 3CNF clause of ϕ' by 3 2XOR clauses (one for every two literals). Let ϕ_{2eq} be the 2EQ formula induced by adding pairs of matched clauses mod 2. If the following hold then ϕ' is not satisfiable:*
(1) ϕ' is δ-balanced.
(2) The largest eigenvalues of matrices $A_{\phi_{3xor}}, A_{\phi_{2eq}}$ are bounded by $\lambda_{3xor}, \lambda_{2eq}$.
(3) $3\delta + \frac{n}{4m}(\lambda_{3xor} + \lambda_{2eq}) < 1/2$.

Lemma 3 (2XOR strong refutation). *Let ϕ be a 2XORformula with m clauses. If λ is the maximum eigenvalue of A_ϕ then ϕ is at most $(1/2+\epsilon)$ satisfiable,for $\epsilon = n\lambda/4m$.*

Proof (sketch). Take the most satisfying assignment T and use the Rayleigh quotient of its corresponding $\{\pm 1\}$ vector (in index i we put 1 iff $T(i) = true$) to lower bound λ. □

Lemma 4 (3XOR property certification). *Let ϕ be a 3CNF formula which is δ-balanced with m clauses and n variables. Assume that the 2XOR formula induced by replacing each 3CNF clause by 3 2XOR clauses is at most $(1/2+\gamma)$ satisfiable. Then ϕ has the $(1-3/2(\delta+2\gamma))$3XOR property.*

Proof. The proof is very similar to the proof of Lemma 1, details omitted.

Proof (of Theorem 4). Assume that ϕ' is satisfiable as 3CNF and show that properties (1),(2) contradict property (3). By Lemma 3 ϕ_{3xor} and ϕ_{2eq} are at most $(1/2+e_{3xor}), (1/2+\epsilon_{2eq})$ satisfied respectively $(e_{3xor} = n\lambda_{3xor}/(4{\cdot}6m), e_{2eq} = n\lambda_{2eq}/4m)$. Using Lemma 4 we conclude that ϕ' is $(1-3/2(\delta+2e_{3xor}))$ satisfied as 3XOR. Each pair of matched clauses of ϕ' for which both clauses are satisfied as 3XOR yields a satisfied 2EQ clause of ϕ_{2eq}. As ϕ_{2eq} is at most $(1/2+\epsilon_{2eq})$ satisfied, we conclude that $1-3(\delta+2e_{3xor}) \leq 1/2+e_{2eq}$ implying $3(\delta+2e_{3xor})+e_{2eq} \geq 1/2$. Substituting e_{3xor}, e_{2eq} with the bounds derived from Lemma 3 we conclude that $3\delta + \frac{n}{4m}(\lambda_{3xor}+\lambda_{2eq}) \geq 1/2$ which contradicts property (3). □

We generated several random formulas with $n = 5\cdot 10^4$ variables and $27335932 = \lceil 2.445\cdot n^{3/2}\rceil$ clauses. Our algorithm refuted all of them (our current implementation fails to refute formulas of significantly lower clause density). We give more detailed results for one specific (though typical) run. Our algorithm extracted a subformula ϕ' with $m = 2689832$ pairs of matched clauses. Table 1 summarizes the values computed by the algorithm along with a heuristic estimation of what we could have expected them to be. Let us explain the heuristic bounds used in the table. To estimate the largest eigenvalue of a symmetric matrix we use the formula $2\sqrt{d}$ where d is the average l_1 norm of each of the rows. This bound is known to be true for various random graph models, but apparently is too optimistic for G_{3xor}. To estimate the imbalance δ we assume that each variable appears exactly $6m/n$ times, each time with random polarity. The difference between the number of positive and negative appearances behaves like the distance from 0 when performing a random walk of length $6m/n$ on $\mathbb{Z}$ (starting from 0). The expected square of the distance is $6m/n$, and $\sqrt{6m/n}$ is an upper bound on the expected distance. We estimated the expected normalized imbalance as $n\sqrt{6m/n}/6m = \sqrt{\frac{n}{6m}}$.

Table 1. Results for a random formula with $5\cdot 10^4$ variables and 27335932 clauses.

	m	λ_{2eq}	λ_{3xor}	δ	Bound
Algorithm	2689832	20.8961	54.6503	0.048662	$0.49706 < 1/2$
Heuristic bound: (using formula)	2690112 $\approx 9c^2n$	20.7454 $2\sqrt{9c^2}$	50.8157 $2\sqrt{54c^2}$	0.05565 $\sqrt{\frac{n}{6m}}$	$0.49946 < 1/2$ $\frac{n}{4m}(\lambda_{2eq}+\lambda_{3xor})+3\delta$

A few words about the implementation of our algorithm. The part of extracting the subformula ϕ' was implemented in C. The other parts (computing the imbalance and the eigenvalues) were implemented in Matlab. To save memory we used Matlab's sparse matrix objects. The heavy part of the algorithm is computing the largest eigenvalues of the two matrices A_{3xor}, A_{2eq}. This part took 63 minutes on an Intel Xeon CPU 1700MHz with 256K cache and 2Gbyte memory (Linux).

It may be interesting to see if other refutation algorithms (such as the ones based on resolution or on OBDDs) can handle random formulas with as many variables as those handled by our algorithm. We have not made a serious attempt to check this.

Acknowledgements. This research was supported by a grant from the G.I.F., the German-Israeli Foundation for Scientific Research and Development. We thank Amin Coja Oghlan for useful discussions.

References

1. E. Ben-Sasson and A. Wigderson. Short proofs are narrow-resolution made simple. *Journal* of the *ACM*(*JACM*), 48(2):149–169, 2001.
2. V. Chvatal and E. Szemeredi. Many hard examples for resolution. *Journal of the ACM,* 35(4):759–768, Oct 1988.
3. A. Coja-Oghlan, A. Goerdt, A. Lanka, and F. Schadlich. Certifying unsatisfiability of random 2k-sat formulas using approximation techniques. In *Proc. of the 14th International Symposium on Fundamentals of Computation Theory,* 2003.
4. U. Feige. Relations between average case complexity and approximation complexity. In *Proc. of the 34th Annual ACM Symposium on Theory of Computing,* pages 534–543, 2002.
5. U. Feige and E. Ofek. Spectral techniques applied to sparse random graphs. Technical report, Weizmann Institute of Science, 2003.
6. E. Friedgut and J. Bourgain. Sharp thresholds of graph properties, and the k-sat problem. *JAMS: Journal of the American Mathematical Society,* 12(4): 1017–1054, 1999.
7. J. Friedman, A. Goerdt, and M. Krivelevich. Recognizing more unsatisfiable random 3-sat instances efficiently. Technical report, 2003.
8. A. Goerdt and M. Krivelevich. Efficient recognition of random unsatisfiable k-SAT instances by spectral methods. In *STACS: Annual Symposium on Theoretical Aspects of Computer Science,* pages 294 –304, 2001.
9. A. Goerdt and A. Lanka. Recognizing more random 3-sat instances efficiently. Manuscript, 2003.
10. M. Hajiaghayi and G.B. Sorkin. The satisfiability threshold for random 3-SAT is at least 3.52. http://arxiv.org/abs/math.CO/0310193, 2003.
11. S. Janson, Y. C. Stamatiou, and M. Vamvakari. Bounding the unsatisfiability threshold of random 3-sat. *Random Structures and Algorithms,* 17(2):103–116, 2000.
12. A.C. Kaporis, L.M. Kirousis, and E.G. Lalas. Selecting complementary pairs of literals. In Proc. LICS'03 Workshop on Typical Case Complexity and Phase Transitions, June 2003.
13. U. Zwick. Outward rotations: a tool for rounding solutions of semidefinite programming relaxations, with applications to MAX CUT and other problems. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing,* pages 679–687, 1999.

On Graph Problems in a Semi-streaming Model*

Joan Feigenbaum[1]**, Sampath Kannan[2]***, Andrew McGregor[2†], Siddharth Suri[2‡], and Jian Zhang[1§]

[1] Yale University, New Haven, CT 06520, USA
{feigenbaum-joan,zhang-jian}@cs.yale.edu
[2] University of Pennsylvania, Philadelphia, PA 19104, USA
{kannan,andrewm,ssuri}@cis.upenn.edu

Abstract. We formalize a potentially rich new streaming model, the *semi-streaming model,* that we believe is necessary for the fruitful study of efficient algorithms for solving problems on massive graphs whose edge sets cannot be stored in memory. In this model, the input graph, $G = (V, E)$, is presented as a stream of edges (in adversarial order), and the storage space of an algorithm is bounded by $O(n \cdot \text{polylog } n)$, where $n = |V|$. We are particularly interested in algorithms that use only one pass over the input, but, for problems where this is provably insufficient, we also look at algorithms using constant or, in some cases, logarithmically many passes. In the course of this general study, we give semi-streaming constant approximation algorithms for the unweighted and weighted matching problems, along with a further algorithm improvement for the bipartite case. We also exhibit $\log n / \log \log n$ semi-streaming approximations to the diameter and the problem of computing the distance between specified vertices in a weighted graph. These are complemented by $\Omega(\log^{(1-\epsilon)} n)$ lower bounds.

1 Introduction

Streaming [14,3,10] is an important model for computation on massive data sets. Recently, there has been a large body of work on designing algorithms in this model [11,3,10,15,13,12]. Yet, the problems considered fall into a small number of categories, such as computing statistics, norms, and histograms. Very few graph problems [5] have been considered in the streaming model.

The difficulty of graph problems in the streaming model arises from the memory limitation of the model combined with input-access constraints. We can view the amount of memory used by algorithms with sequential (one-way) input

* This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795.
** Supported in part by ONR and NSF.
*** Supported in part by ARO grant DAAD 19-01-1-0473 and NSF grant CCR-0105337.
† Supported in part by NIH.
‡ Supported in part by NIH grant T32 HG000046-05.
§ Supported by ONR and NSF.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 531–543, 2004.

access as a spectrum. At one end of the spectrum, we have dynamic algorithms [9] that may use memory enough for the whole input. At the other end, we have streaming algorithms that use only polylog space. At one extreme, there is a lot of work on dynamic graph problems; on the other, general graph problems are considered hard in the (poly)log-space streaming model. Recently, it has been suggested by Muthukrishnan [19] that the middle ground, where the algorithms can use $O(n \cdot \text{polylog } n)$ bits of space is an interesting and open area. This is the area that we explore.

Besides taking a middle position in the memory-size spectrum, the semi-streaming model allows multiple passes over the input stream. In certain applications with massive data sets, a small number of sequential passes over the data would be much more efficient than many random accesses to the data. Only a few works [8,4] have considered the multiple-pass model and a lot remains to be done.

Massive graphs arise naturally in many real world scenarios. Two examples are the *call graph,* where nodes correspond to telephone numbers and edges to calls between numbers that call each other during some time interval, and the *web graph,* where nodes are web pages, and the edges are links between pages. The streaming model is necessary for the study of the efficient processing of such massive graphs. In [1], the authors introduce the semi-external model for computations on massive graphs, *i.e.,* one in which the vertex set can be stored in memory, but the edge set cannot. However, this work addresses the problems in an external memory model in which random access to the edges, although expensive, is allowed. This is a major difference between their model and ours. Indeed, the authors of [18] argue that one of the major drawbacks of standard graph algorithms, when applied to massive graphs such as the web, is their need to have random access to the edge set. Furthermore, there are situations in which the graph is revealed in a streaming fashion, such as a web crawler exploring the web graph.

We consider a set of classical graph problems in this semi-streaming model. We show that, although the computing power of this model is still limited, there are semi-streaming algorithms for a variety of graph problems. Our main result is a semi-streaming algorithm that computes a $(2/3-\epsilon)$-approximation in $O(\frac{\log 1/\epsilon}{\epsilon})$ passes for unweighted bipartite graph matching. We also provide a one-pass semi-streaming algorithm for 1/6-approximating the maximum weighted graph matching. We also provide $\log n/\log\log n$ approximations for diameter and shortest paths in weighted graphs which we complement with $\Omega(\log^{(1-\epsilon)} n)$ lower bounds for these problems in unweighted graphs.

2 Preliminaries

Unless stated otherwise, we denote by *G(V, E)* a graph G with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E = \{e_1, e_2, \ldots, e_m\}$. Note that n is the number of vertices and m the number of edges.

Definition 1. *A **graph stream** is a sequence of edges* $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$, *where* $e_{i_j} \in E$ *and* $i_1, i_2, \ldots, i_m$ *is an arbitrary permutation of* $[m] = \{1, 2, \ldots, m\}$.

While an algorithm goes through the stream, the graph is revealed one edge at a time. This definition generalizes the streams of graphs in which the adjacency matrix or the adjacency list is presented as a stream. In a stream in the adjacency-matrix or adjacency-list models, the edges incident to each vertex are grouped together. We need the more general model to account for graphs such as call graphs where the edges might generated in any order.

The efficiency of a graph algorithm in the semi-streaming model is measured by the space it uses, the time it requires to process each edge, and the number of passes it makes over the graph stream.

Definition 2. *A **semi-streaming graph algorithm** computes over a graph stream using* $S(n, m)$ *bits of space. The algorithm may access the input stream in a sequential order(one-way) for* $P(n, m)$ *passes and use* $T(n, m)$ *time to process each edge. It is required that* $S(n, m)$ *be* $O(n \cdot \text{polylog}\ (n))$ *bits.*

To see the limitation of the (poly) log-space streaming model for graph problems, consider the following simple problem. Given a graph, determining whether there is a length-2 path between two vertices, x and y, is equivalent to deciding whether two vertex sets, the neighborhood of x and the neighborhood of y, have a nonempty intersection. Because set disjointness has linear-space communication complexity [17], the length-2 path problem is impossible in the (poly)log-space streaming model. See [4] for a more comprehensive treatment of finding common neighborhoods in the streaming model.

3 Graph Matching

3.1 Unweighted Bipartite Matching

In this subsection, we present an algorithm for approximating unweighted bipartite matching. First, a bipartition can be found by the following algorithm. Note that the labeling of the vertices keeps track of the connected components in the graph seen so far, and the signs keep a partition for each connected component.

Algorithm 1 (Bipartition). *As edges stream in, we use a disjoint set data structure to maintain the connected components of the graph so far. We associate a sign with each vertex such that no edge joins two vertices of the same sign. If this condition ever fails and cannot be corrected by flipping the sign of a vertex and the vertices in its connected component, then we output that the graph is non-bipartite.*

The disjoint set data structure with union by rank and path compression can be augmented to maintain the signs without increasing the amortized time, $\alpha(m, n)$ needed per edge.

Given a matching M, we call a vertex *free* if it doesn't appear as the end point of any edge in M. It is easy to see that a maximal matching (thus a 1/2-approximation) for a graph can be constructed by a semi-streaming algorithm in one pass over the graph stream: when going through the stream, the algorithm adds an edge to the current matching M if both ends of the edge are free *w.r.t.* M. (This constructs a maximal matching for an arbitrary graph, not just for bipartite graphs.)

Consider a matching M for a bipartite graph $G = (L \cup R, E)$. A length-3 augmenting path for an edge $e = (u, v) \in M$, $u \in L$ and $v \in R$, is a quadruple (w_l, u, v, w_r) such that $(u, w_l), (w_r, v) \in E$, and w_l and w_r are free vertices. We call w_l and w_r the *wing-tips* of the augmenting path, (u, w_l) the *left wing* and (w_r, v) the *right wing*. A set of *simultaneously augmentable length-3 augmenting paths* is a set of length-3 augmenting paths that are vertex disjoint.

We now provide an algorithm that will be used as a subroutine in our main unweighted bipartite matching algorithm. Given a bipartite graph and a matching of the graph, this algorithm finds a set of simultaneously augmentable length-3 augmenting paths.

Algorithm 2 (Find Augmenting Paths). *The input to the algorithm is a graph $G = (L \cup R, E)$, a matching M for G and a parameter $0 < \delta < 1$.*

1. *In one pass, find a maximal set of disjoint left wings. If the number of left wings found is $\leq \delta M$, terminate.*
2. *In a second pass, for the edges in M with left wings, find a maximal set of disjoint right wings.*
3. *In a third pass we identify the set of vertices that*
   a) *Are endpoints of a matched edge that got a left wing.*
   b) *Are the wing tips of a matched edge that got* both *wings.*
   c) *Are endpoints of a matched edge that is no longer 3 augmentable.*

   We remember these vertices and in subsequent passes, we ignore any edge incident on one of these vertices.
4. *Repeat.*

Our main unweighted bipartite matching algorithm increases the size of a matching by repeatedly finding a set of simultaneously augmentable length-3 augmenting paths and augmenting the matching using these paths.

Algorithm 3 (Unweighted Bipartite Matching). *The input to the algorithm is a bipartite graph $G = (L \cup R, E)$ and a parameter $0 < \epsilon < 1/3$.*

1. *In one pass, find a maximal matching M and the bipartition of G.*
2. *For $k = 1, 2, \ldots, \lceil \frac{\log 6\epsilon}{\log 8/9} \rceil$ Do:*
   a) *Run the algorithm 2 with G, M and $\delta = \frac{\epsilon}{2-3\epsilon}$.*
   b) *For each $e = (u, v) \in M$ for which an augmenting path (w_l, u, v, w_r) is found by algorithm 2, remove (u, v) from M and add (u, w_l) and (w_r, v) to M.*

We now establish a relationship between the size of a maximal set of simultaneously augmentable length-3 augmenting paths and the size of a maximum such set.

Lemma 1. *The size of a maximal set of simultaneously augmentable length-3 augmenting paths is at least 1/3 of the size of a maximum set of simultaneously augmentable length-3 augmenting paths.*

Proof. Let AP_{max} be some maximal set of simultaneously length-3 augmentable paths. Note that each path that we find destroys at most 3 paths that AP_{max} might have used, one involving each of the wing tips that we use and a third path involving the matched edge we used. Thus we have a 1/3-approximation.

Lemma 2. *Let X be a maximum-sized set of simultaneously augmentable length-3 augmenting paths for a maximal matching M. Let* $\alpha = \frac{|X|}{|M|}$ *and* OPT *a maximum matching.* $|M|(1+\alpha) \geq 2/3\ |\text{OPT}|$.

Proof. See for example, [16], page 156.

Lemma 3. *Algorithm 2 finds* $\frac{\alpha|M|-2\delta|M|}{3}$ *simultaneously augmentable length-3 augmenting paths in* $3/\delta$ *passes.*

Proof. Let L(M) be the set of the end vertices of the edges in M that are in L and $V_L(M) = \{v \in R \mid v$ is free *w.r.t.* M and $\exists u \in L(M)$ *s.t.* $(u,v) \in E\}$. We call one repetition of step 1–4 in the algorithm a phase. The number of phases is at most $1/\delta$ because at least $\delta|M|$ edges in M are removed at each phase.

When the algorithm terminates, the number of left wings found is at most $\delta|M|$. Note that the set of left wings found form a maximal matching between the remaining vertices in $L(M)$ and $V_L(M)$. Hence there are fewer than $2\delta|M|$ disjoint left wings that could have been found at this phase. Consequently, there are fewer than $2\delta|M|$ simultaneously augmentable length-3 augmenting paths in the remaining graph, which we denote G'.

Let $G'' = G \setminus G'$. Note that a maximum set of simultaneously augmentable length-3 augmenting paths in G'' would have a size at least $\alpha|M| - 2\delta|M|$. Also note that the set of length-3 augmenting paths found by the algorithm form a maximal set *w.r.t.* G''. By Lemma 1, the size of such a set is at least $\frac{\alpha|M|-2\delta|M|}{3}$.

Theorem 1. *For any* $0 < \epsilon < 1/3$ *and a bipartite graph, algorithm 3 finds a* $2/3-\epsilon$ *approximation of maximum matching in* $O\left(\frac{\log 1/\epsilon}{\epsilon}\right)$ *passes. The algorithm processes each edge in* $O(1)$ *time in each pass except the first pass, in which the bipartition is found. The amortized per-edge processing time is* $\alpha(m,n)$ *for finding the bipartition. The storage space required by the algorithm is* $O(n \log n)$.

Proof. It is easy to see the bounds for the per-edge processing time and storage space. We now show the correctness of the algorithm. Let OPT be the size of the maximum matching. At the ith phase, let M_i be the matching found by the algorithm and X_i a maximum-sized set of simultaneously augmentable length-3 augmenting paths for the matching M_i. Let $\alpha_i = \frac{|X_i|}{|M_i|}$ and $s_i = \frac{|M_i|}{\text{OPT}}$.

Note that we only need to consider the case where $\alpha_i > \frac{3\epsilon}{2-3\epsilon}$. Otherwise, by Lemma 2, the matching M_i is already a $\frac{2}{3}\frac{1}{1+\alpha_i} \geq \frac{2}{3} - \epsilon$ approximation. Assuming $\alpha_i > \frac{3\epsilon}{2-3\epsilon}$ for all stage i, let $\delta = \frac{\epsilon}{2-3\epsilon}$. Then $\delta \leq \frac{\alpha_i}{3}$ for all α_i. By Lemma 3, the number of simultaneously augmentable length-3 augmenting paths found by Algorithm 2 is then $\frac{\alpha_i|M_i| - 2\delta|M_i|}{3} \geq \frac{\alpha_i|M_i|}{9}$.

Because M_0 is a maximal matching, $s_0 \geq 1/2$. At any stage, by Lemma 2, $|M_i| + \alpha_i|M_i| \geq 2/3 \cdot \text{OPT}$. This gives:

$$s_i + \alpha_i s_i \geq 2/3 \tag{1}$$

By Lemma 3, $|M_{i+1}| = |M_i| \cdot (1 + \frac{\alpha_i - 2\delta}{3}) \geq |M_i| \cdot (1 + \alpha_i/9)$. This gives:

$$s_{i+1} \geq s_i + \alpha_i s_i/9 \tag{2}$$

Putting together inequalities 1 and 2, we have: $s_{i+1} \geq 8/9 \cdot s_i + 2/27$. Solving this recurrence gives $s_i \geq 2/3 - 1/6(8/9)^i$. Note that, the algorithm runs in $k = \left\lceil \frac{\log 6\epsilon}{\log 8/9} \right\rceil$ stages. Thus $\frac{|M_k|}{\text{OPT}} = s_k \geq 2/3 - \epsilon$.

At each stage of algorithm 3, we run algorithm 2 as a subroutine. There are $k = \left\lceil \frac{\log 6\epsilon}{\log 8/9} \right\rceil$ stages and each stage requires $\frac{6-9\epsilon}{\epsilon}$ passes. The total number of passes is then:

$$\left\lceil \frac{\log 6\epsilon}{\log 8/9} \right\rceil \frac{6 - 9\epsilon}{\epsilon} = O\left(\frac{\log 1/\epsilon}{\epsilon}\right)$$

3.2 Weighted Matching

In the weighted matching problem, every edge e has a weight $w(e)$. We seek the matching M for which $\sum_{e \in M} w(e)$ is maximized. This is also a well studied problem when we do not restrict ourselves to the streaming model.

At least one existing algorithm [21] can easily be adapted to work in our model. For any $\epsilon > 0$, the streaming version finds a weighted matching that is at least $1/(2+\epsilon)$ the optimal size using $O(\log_{1+\epsilon/3} n)$ passes and $O(n \log n)$ storage. The algorithm works by geometrically grouping the weights into $\lceil \log_{1+\epsilon/3}(\lceil 3/\epsilon + 1.5 \rceil) n \rceil$ groups and then, for each group, starting at those with the largest weights, finding maximal matchings. Each maximal matching can be found with one pass. Further details and the proof of correctness can be found in [21].

We propose a new algorithm that uses only one pass yet still manages to find a matching which is at least $\frac{1}{6}$ of the optimal size.

Algorithm 4 (Weighted Matching). *We maintain a matching M at all times. When we see a new edge e, we compare $w(e)$ with $w(C)$, the sum of the weights of the edges of $C = \{e' | e' \in M$ and e' and e share an end point$\}$.*

- *If $w(e) > 2w(C)$, we update $M \leftarrow M \cup \{e\} \setminus C$.*
- *If $w(e) \leq 2w(C)$, we ignore e and wait for the next edge.*

Theorem 2. *In 1 pass and $O(n \log n)$ storage,we can construct a weighted matching that is at least $\frac{1}{6}$ the of the optimal size.*

Proof. For any set of edges S, let $w(S) = \sum_{e \in S} w(e)$. We say that an edge is *born* if it is ever part of M. We say that an edge is *killed* if it was born but subsequently removed from M by a newer heavier edge. This new edge *murdered* the killed edge. We say an edge is a *survivor* if it is born and never killed. Let the set of survivors be S. The weight of the matching we find is therefore $w(S)$.

For each survivor e, let the *Trail of the Dead* leading to this edge be $T(e) = C_1 \cup C_2 \cup \ldots$ where $C_0 = \{e\}$, $C_1 = \{$the edges murdered by $e\}$, and $C_i = \cup_{e' \in C_{i-1}} \{$the edges murdered by $e'\}$

Claim: $w(T(e)) \leq w(e)$

Proof of claim: For each murdering edge e, $w(e)$ is at least twice the cost of murdered edges, and an edge has at most one murderer. Hence, for all i, $w(C_i) \geq 2w(C_{i+1})$ therefore

$$2w(T(e)) = \sum_{i \geq 1} 2w(C_i) \leq \sum_{i \geq 0} w(C_i) = w(T(e)) + w(e)$$

The claim follows.

Now consider the optimal solution that includes edges $\text{OPT} = \{o_1, o_2, \ldots\}$. We are going to charge the costs of edges in OPT to the survivors and their trails of the dead, $\cup_{e \in S} T(e) \cup \{e\}$. We hold an edge e in this set *accountable to* $o \in \text{OPT}$ if either $e = o$ or if o wasn't born because e was in M when o arrived. Note that, in the second case, it is possible for two edges to be accountable to o. If only one edge is accountable for o then we charge $w(o)$ to e. If two edges e_1 and e_2 are accountable for o, then we charge $\frac{w(o)w(e_1)}{w(e_1)+w(e_2)}$ to e_1 and $\frac{w(o)w(e_2)}{w(e_1)+w(e_2)}$ to e_2. In either case, the amount charged by o to any edge e is at most $2w(e)$.

We now redistribute these charges as follows: (for distinct u_1, u_2, u_3) if $e = (u_1, v)$ gets charged by $o = (u_2, v)$, and e subsequently gets killed by $e' = (u_3, v)$, we transfer the charge from e to e'. Note that we maintain the property that the amount charged by o to any edge e is at most $2w(e)$ because $w(e') \geq w(e)$. What this redistribution of charges achieves is that now every edge in a trail of the dead is only charged by one edge in OPT. Survivors can, however, be charged by two edges in OPT. We charge $w(\text{OPT})$ to the survivors and their trails of the dead, and hence

$$w(\text{OPT}) \leq \sum_{e \in S} 2w(T(e)) + 4w(e)$$

By Claim 1,

$$\sum_{e \in S} 2w(T(e)) + 4w(e) \leq 6w(S)$$

and the theorem follows.

3.3 Lower Bounds

In contrast to the above results, it is worth noting that even to check whether an existing matching is maximum in 1 pass requires $\Omega(m)$ space. First, we prove that testing s,t connectivity in a directed graph requires $\Omega(m)$ space.

Lemma 4. *Testing for $s-t$ connectivity in a directed graph $G = (V, E)$ requires $\Omega(m)$ bits of space, where $|E| = m$.*

Proof. Consider the family $\mathcal{F}$ of graphs $G = (L \cup R \cup \{s,t\}, E)$, where the induced graph on $L \cup R$ is an arbitrary bipartite graph with $|L| = |R| = n$ and $m \leq n^2/2$ edges and all the edges are directed from L to R. Say the stream gives all the edges between L and R first then one edge of the form (s,l) and then (r,t), where $l \in L$ and $r \in R$. At the point at which all the edges from L to R have appeared any correct algorithm must have a different memory configuration for each graph in $\mathcal{F}$ since there are continuations that will result in different answers for any two graphs in $\mathcal{F}$. Thus the number of bits of space required is $\Omega(log_2|\mathcal{F}|$ which is easily seen to be $\Omega(m)$.

Theorem 3. *Consider a bipartite graph $G = (L \cup R, E)$. Testing whether there exists an augmenting path from $s \in R$ to $t \in L$ requires $\Omega(m)$ bits of storage.*

Proof. Let the storage required be $S(n)$. Let $G = (\{v_1, \ldots, v_n\}, E)$ be a directed graph and assume, without loss of generality, that $s = v_1$ and $t = v_n$. We will use the "test for an augmenting path" algorithm to answer the s,t directed-connectivity problem. We construct an undirected bipartite graph G' with nodes v_s, v_t such that there exists an augmenting path from v_s to v_t in G' if and only if there exists a directed path from s to t in G.

For each node v_i in G, create two nodes v_{il} and v_{ir} in G'. In addition, add nodes v_s and v_t to G'. Let the edges of G' be $E' = \{(v_{il}, v_{ir}) : i \in [n]\} \cup \{(v_{ir}, v_{jl}) : (v_i, v_j) \in E\} \cup \{(v_s, v_{1l})\} \cup \{(v_t, v_{nr})\}$. Let the existing matching be $M = \{(v_{il}, v_{ir}) : i \in [n]\}$. There is an augmenting path in G' if and only if there is a path from s to t in G.

4 Distances, Girth, and Other Problems

In this section, we consider the problems of computing shortest-path distances, diameter, and girth on graph streams. We briefly mention several other graph-stream problems at the end of this section.

First we show that, in the semi-streaming model, computing exact shortest-path distances, even certain approximation, is not possible in one pass. In a graph G, we say an edge (u, v) is *k-critical* if the shortest path from u to v in $G \setminus (u, v)$ has length $\geq k$.

Lemma 5. *For $1 > \epsilon > 0$ and sufficiently large n there exists a graph $G = (V, E)$ with $|V| = n$, $|E| = 2^{\log^\epsilon n} n/4$ such that the majority of edges are $\frac{\log^{1-\epsilon} n}{2}$-critical and the majority of the subgraphs in the set*

$$\{G' : G' \text{ formed from } G \text{ by deleting a subset of the } \tfrac{\log^{1-\epsilon} n}{2}\text{-critical edges}\}$$

have diameter less or equal to $4 \log^{1-\epsilon} n$.

Proof (Sketch). We show existence by considering a random graph $G \in \mathcal{G}_{n,p}$ where n is very large and $p = 2^{\log^\epsilon n}/n$. Clearly the number of edges in G is at least $(2^{\log^\epsilon n} n)/4$ with high probability.

Claim 1: w.h.p. the majority of edges are k-critical where $k = \frac{\log^{1-\epsilon} n}{2}$.

Proof of Claim 1: By the Chernoff bound, with high probability, no vertex has degree greater than $2 \cdot 2^{\log^\epsilon n}$. We henceforth assume this to be the case. Consider an edge (u, v). Let $\Gamma_i(v)$ be the vertices up to a distance i from v in $G \setminus (u, v)$ and then

$$|\Gamma_k(v)| \leq \sum_{0 \leq i \leq k} (2 \cdot 2^{\log^\epsilon n})^i \leq (2 \cdot 2^{\log^\epsilon n})^{k+1} \leq 2^{2(\log n)/3}$$

Hence the probability that (u, v) was k-critical is $1 - 2^{2(\log n)/3 - \log n} \to 1$. Thus, by the Chernoff bound, the majority of edges are k-critical with high probability.

Claim 2: Consider $G \in \mathcal{G}_{n,p/2}$ w.h.p. the diameter of G is $< D = 4 \log^{1-\epsilon} n$.

Proof of Claim 2: Pick a node $v \in G$. Let $S_i = \Gamma_i(v) \setminus \Gamma_{i-1}(v)$. By the Chernoff bound, with high probability, for all i such that $|\Gamma_i(v)| < n/2$ we have $|S_{i+1}| > |S_i| 2^{\log^\epsilon n}/4$. Consider the first value of i for which $|\Gamma_i| \geq n/2$. By the above bound $S_i \geq n/4$ and with high probability $|\Gamma_{i+1}| = n$. Hence the distance between v and every other vertex is $\leq \frac{\log n}{\log^\epsilon n - 2} < 2 \log^{1-\epsilon} n$ and so the diameter is $< 4 \log^{1-\epsilon} n$.

Let $G \in \mathcal{G}_{n,p}$ and G' be a random subgraph of G. Picking a random subgraph of a graph picked from $\mathcal{G}_{n,p}$ is the same as picking a graph from $\mathcal{G}_{n,p/2}$. Consider the events

$$A = \{G \in \mathcal{G}_{n,p} : \text{majority of } E(G) \text{ are } k\text{-critical}\}$$
$$B = \{G \in \mathcal{G}_{n,p/2} : \text{diameter of } G \text{ is } < D\}$$
$$\text{for each graph } H,\ B_H = \{\text{subgraphs } H' \text{ of } H : \text{diameter of } H' \text{ is } < D\}$$

From claim 2 we know $\mathbb{P}(B)$ is large and from claim 1 we know $\mathbb{P}(\neg A)$ is small. Thus $\mathbb{P}(A \cap B) \geq \mathbb{P}(B) - \mathbb{P}(\neg A) > 1/2$. $\mathbb{P}(A \cap B) = \sum_G I[A] \mathbb{P}(G) \mathbb{P}(B_G)$ and so there exists a graph $G \in A$ such that $\mathbb{P}(B_G) \geq 1/2$, i.e. the majority of subgraphs of G have diameter $< D$. Now, undeleting edges that weren't k-critical can only decrease the diameter and

hence there exists a least one graph G such that majority of $\{G' : G'$ formed from G by deleting a subset of k-critical edges$\}$ have diameter $< D$.

Theorem 4. *For $1 > \epsilon > 0$, it is impossible in one pass to approximate the diameter of an unweighted graph within a factor of $o(\log^{1-\epsilon} n)$ in the semi-streaming model.*

Proof. Let $k = \frac{\log^{1-\epsilon} n}{2}$ and $D = 8k$. Consider a graph G on $\eta = \frac{n-2D}{D}$ vertices with the properties in Lemma 5. Let the subgraphs of this G with diameter $< D$ be $\mathcal{F}(G)$. Observe that

$$|\mathcal{F}(G)| \geq 2^{2^{\log^{\epsilon} n} n/8} = 2^{\omega(n \cdot \text{polylog } n)}$$

and hence, for any algorithm there exists two graphs $G', G'' \in \mathcal{F}(G)$ that are indistinguishable from information stored by the algorithm. Consequently when we stream in one of G', G'' there exists an edge $e \in E(G') \backslash E(G'')$ whose existence in the streamed graph is undetermined. We stream in D graphs $G_1, \ldots, G_D$ from $\mathcal{F}(G)$ such that there exists an edge (t_i, s_i) in each G_i whose existence is undetermined. Finally we stream in edges (t_i, s_{i+1}) for $i = 2, \ldots D-1$ and two disjoint length D paths, one with end points s and t_1 and the other with end points s_D and t. Because of these two paths, the diameter is realized by the shortest path between s and t. Our construction gives a graph that has diameter 4D – 1 while the algorithm can not guarantee that the diameter is less than $3D - 1 + Dk$. Hence the best approximation ratio is $\Omega(k)$.

We next show that the shortest-path distances can be approximated in one pass in the semi-streaming model. The approximation uses *graph spanners.* A subgraph $G'(V, E_s)$ is a t-spanner of graph $G(V, E)$ if, between any pair of vertices, the distance in G' is at most t times the distance in G.

For an unweighted graph, a $\log n / \log \log n$-spanner S can be constructed in one pass in the semi-streaming model using a simple algorithm similar to the one in [2]. Because a graph whose girth is larger than k can only have $\lceil n^{1+2/(k-1)} \rceil$ edges [6], the algorithm constructs S by adding the edges in the stream to S, if such an edge does not cause a cycle of length less than $\log n / \log \log n$ in the spanner S constructed so far. For a weighted graph, however, the construction in [2] requires to sort the edges according to their weights, which is difficult in the semi-streaming model. So, instead of sorting, we use a geometric grouping technique to extend the spanner construction for unweighted graphs to a construction for weighted graphs. This technique is similar to the one used in [7]. Let ω_{min} be the minimum weight and let ω_{max} be the maximum weight. We divide the range $[\omega_{min}, \omega_{max}]$ into intervals of the form $[(1+\epsilon)^i \omega_{min}, (1+\epsilon)^{i+1} \omega_{min})$ and round all the weights in the interval $[(1+\epsilon)^i \omega_{min}, (1+\epsilon)^{i+1} \omega_{min})$ down to $(1+\epsilon)^i \omega_{min}$. For each induced graph $G^i = (V, E^i)$, where E^i is the set of edges in E whose weight is in the interval $[(1+\epsilon)^i \omega_{min}, (1+\epsilon)^{i+1} \omega_{min})$, a spanner can be constructed in parallel using the above construction for unweighted graphs. The union of the spanners for all the G^i, $i \in \{0, 1, \ldots, \log_{(1+\epsilon)} \frac{\omega_{max}}{\omega_{min}} - 1\}$, forms

a spanner for the graph G. Note that this can be done without prior knowledge of ω_{min} and ω_{max}.

Theorem 5. *For $\epsilon > 0$, and a weighted undirected graph on n vertices, whose maximum edge weight, ω_{max}, and minimum edge weight, ω_{min}, satisfy $\log \frac{\omega_{max}}{\omega_{min}} = \text{polylog } n$, there is a semi-streaming algorithm that constructs a $(1+\epsilon)\log n$-spanner of the graph in one pass. The algorithm uses $O(\log_{1+\epsilon} \frac{\omega_{max}}{\omega_{min}} \cdot n \log n)$ bits of space and the worst case processing time for each edge is $O(\log_{1+\epsilon} \frac{\omega_{max}}{\omega_{min}} \cdot n)$.*

Once we have the spanner, the distance between any pair of vertices can be approximated by computing their distance in the spanner. The diameter of the graph can be approximated by the spanner diameter too. Note that, if the girth of an unweighted graph is larger than k, it can be determined exactly in a k-spanner of the graph. The construction of the $\log n/\log\log n$-spanner thus provides a $\log n/\log\log n$-approximation for the girth.

We end this section by briefly mentioning some graph problems that are simple in the semi-streaming model but may be impossible in a (poly)log-space streaming setting.

A minimum spanning tree can be constructed in one pass and $O(\log n)$ time per edge using a simple adaptation of an existing on-line algorithm [20]. Planarity testing is impossible in the (poly)log-space streaming model, because deciding the existence of a K_5 minor of a graph would require $O(n)$ bits of space. Because a planar graph would have at most $3n-6$ edges, using $O(n)$ storage, many existing algorithms can be adapted to the semi-streaming model.

The following is an algorithm for finding articulation points in the semi-streaming model. It uses one disjoint set data structure, SF, for keeping track of the connected components of the spanning forest, T. It also uses one disjoint set data structure per vertex v, in order to store v's neighbors.

Algorithm 5 (Articulation Points).

$T = (V, \emptyset)$
For each $v \in V$: *SF.makeset(v)*
For each input edge (u, v):
  if SF.find-set(u) = SF.find-set(v) then:
    find the path, $u = a_0, a_1, \ldots, a_k = v$ *from* u *to* v *in* T
    For each $a_i, 0 < i < k$, a_i*.union*(a_{i-1}, a_{i+1}).
  else:
    SF.union(u,v)
    $T = T \cup \{(u, v)\}$
    u.makeset(v)
    v.makeset(u)
For each $v \in V$:
  if the neighbors of v *w.r.t.* T *lie in at least two different sets*
    then output v *as an articulation point.*

If u is an articulation point there exists two neighbors v and w of u in T such that any path from v to w passes through u. In this case, in the disjoint set structure for u the components containing v and w will never be unioned.

5 Conclusion

We considered a set of classical graph problems in the semi-streaming model. We showed that although exact answers to most of these problems are still impossible, certain approximations are possible. More research is needed for a complete understanding of the model. Particularly, the efficiency of an algorithm in the semi-streaming model is measured by $S(m,n)$, $P(m,n)$ and $T(m,n)$ as in definition 2. Together with the approximation factor, an algorithm thus has 4 parameters. It would be interesting to develop a better understanding of the tradeoffs among these parameters.

References

1. J. Abello, A.L. Buchsbaum, J.R. Westbrook. A Functional Approach to External Graph Algorithms. *Algorithmica,* 32(3): 437–458, 2002.
2. I. Althöfer, G. Das, D. Dobkin, and D. Joseph. Generating sparse spanners for weighted graphs. In *Proc. 2nd Scandinavian Workshop on Algorithm Theory (SWAT'90),* LNCS 447, 26–37, 1990.
3. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences,* 58(1):137–147, Feb. 1999.
4. A.L. Buchsbaum, R. Giancarlo and J.R. Westbrook. On finding common neighborhoods in massive graphs. *Theoretical Computer Science,* 299 (1-3):707–718, 2003.
5. Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms,* pages 623–632, 2002.
6. B. Bollobás. *Extremal Graph Theory.* Academic Press, New York, 1978.
7. E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. on Computing,* 28:210-236, 1998.
8. P. Drineas and R. Kannan. Pass Efficient Algorithm for approximating large matrices. In *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms,* pages 223-232, 2003.
9. D. Eppstein, Z. Galil, and G.F. Italiano. Dynamic graph algorithms. *CRC Handbook of Algorithms and Theory of Computation,* Chapter 8, 1999.
10. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1 difference algorithm for massive data streams. *SIAM Journal on Computing,* 32(1):131–151, 2002.
11. P. Flajolet and G.N. Martin. Probabilistic counting. In *Proc. 24th IEEE Symposium on Foundation of Computer Science,* pages 76–82, 1983.
12. A.C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proc. 34th ACM Symposium on Theory of Computing,* pages 389–398, 2002.

13. S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proc. 33th ACM Symposium on Theory of Computing,* pages 471–475, 2001.
14. M. Rauch Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Report 1998-001, DEC Systems Research Center,* 1998.
15. P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. 41th IEEE Symposium on Foundations of Computer Science,* pages 189–197, 2000.
16. M. Karpinski and W. Rytter *Fast Parallel Algorithms for Graph Matching Problems, Oxford Lecture Series in Math. and its Appl.* Oxford University Press, 1998.
17. B. Kalyanasundaram and G. Schnitger. The Probabilistic Communication Complexity of Set Intersection. *SIAM Journal on Discrete Math.,* 5:545–557, 1990.
18. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Extracting large-scale knowledge bases from the web. In *Proceedings of the 25th VLDB Conference,* pages 639–650, 1999.
19. S. Muthukrishnan. Data streams: Algorithms and applications. 2003. *Available at* "http://athos.rutgers.edu/~muthu/stream-1-1.ps"
20. R. Tarjan. Data Structures and Network Algorithms. *SIAM,* Philadelphia, 1983.
21. R. Uehara and Z. Chen. Parallel approximation algorithms for maximum weighted matching in general graphs. *Information Processing Letters,* 76(1–2):13–17, 2000.

Linear Tolls Suffice: New Bounds and Algorithms for Tolls in Single Source Networks

Lisa Fleischer[1,2]*

[1] T. J. Watson Research, IBM, Yorktown Heights, NY
lkf@watson.ibm.com
[2] Carnegie Mellon University, Pittsburgh, PA 15213
lkf@andrew.cmu.edu

Abstract. We show that tolls that are *linear* in the latency of the maximum latency path are necessary and sufficient to induce heterogeneous network users to independently choose routes that lead to traffic with minimum average latency. This improves upon the earlier bound of $O(n^3 l_{\max})$ given by Cole, Dodis, and Roughgarden in STOC 03. (Here, n is the number of vertices in the network; and $l_{\max}$ is the maximum latency of any edge.) Our proof is also simpler, relating the Nash flow to the optimal flow as flows rather than cuts.

We model the set of users as the set [0,1] ordered by their increasing willingness to pay tolls to reduce latency — their *valuation of time*. Cole, et al. give an algorithm that computes optimal tolls for a bounded number of agent valuations, under the very strong assumption that they know which path each user type takes in the Nash flow imposed by these (unknown) tolls. We show that in series parallel graphs, the set of paths travelled by users in any Nash flow with optimal tolls is *independent* of the distribution of valuations of time of the users. In particular, for any continuum of users (not restricted to a finite number of valuation classes) in series parallel graphs, we show how to compute these paths without knowing α.

We give a simple example to demonstrate that if the graph is not series parallel, then the set of paths travelled by users in the Nash flow depends critically on the distribution of users' valuations of time.

1 Introduction

In a (transit/Internet/telecommunications) traffic network, the *latency* of a link is the time required to travel from one end of the link to the opposite end. In a simple model of traffic, the latency of an edge is a nonnegative, nondecreasing function of the flow on the edge: Given graph $G=(V,E)$, with $n = |V|$, the latency of edge $e \in E$ is a function $l_e : R^+ \cup \{0\} \to R^+ \cup \{0\}$. We consider such

* This research is supported in part by NSF grant CCR-0049071.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 544–554, 2004.

a model in this paper and look at how to induce selfish users of the network to follow a traffic pattern that minimizes the average latency experienced by the users. Such a traffic pattern is called a *system optimal flow*. If we assume that the total flow volume from s to t is 1, then a system optimal flow is equivalently expressed as an s-t flow f of value 1 that minimizes $\sum_{e \in E} l_e(f_e) f_e$.

A selfish user traveling from s to t chooses a path P that minimizes the latency experienced on the path: given that all other network traffic is fixed as f, the traveler minimizes $\sum_{e \in P} l_e(f_e)$. This model is introduced in [12]. If each user succeeds in doing this, the resulting traffic pattern is called a *Nash flow,* since it is a Nash equilibrium for the routing game where each player is a user with action space the set of all s-t paths. The Nash flow may be far from a system optimal flow [8,10].

Tolls are a well-known method to induce homogeneous users to choose paths that minimize the average latency while the users selfishly choose paths that minimize individual latency plus toll. For *marginal cost tolls:* $\tau_e = l'_e(f_e) f_e$, the *Nash flow with tolls* τ is a system optimal flow. (See for example [2,9].)

What happens if the users are heterogeneous? To model this, consider for each agent a there is some multiplier $\alpha(a)$ that represents a's valuation of time. User a seeks a path P that minimizes $\sum_{e \in P} \alpha(a) l_e(f_e) + \tau_e$. [1] Early work considers when users pay different tolls on the same edge, according to their multiplier α [6,11]. This is unsatisfying, and also hard to enforce, as it requires knowing individual users' α values, as opposed to the simply a distribution of α-values of the range of users.

Instead, a natural question is, given a distribution α, find a unique toll for each edge that induce users to choose a system optimal flow. We call such tolls *optimal tolls*. Cole, Dodis, and Roughgarden [5] show that optimal tolls exist. Their proof is nonconstructive, and they bound the size of the tolls necessary by $\alpha_{\max} l_{\max} n^3$, where $\alpha_{\max} = \max_a \alpha(a)$ and $l_{\max} = \max_e l_e(1)$. The proof uses Brouwer's fixed point theorem, and a complicated argument about cuts in the network.

We show that *linear tolls suffice:* the optimal toll on each edge need be no more than the latency of the maximum latency path in the minimum average latency flow times the maximum valuation of time on that edge. This quantity is always less than $\alpha_{\max} l_{\max} n$. This bound is also tight: there are instances that require tolls that are linear in the size of the maximum latency path in the network. Our proof is also simpler, as it relates the Nash flow to the system optimal flow directly as flows, rather than indirectly through cuts. This linear bound also holds in the multiple source, single sink setting.

We consider the set of users as the set [0,1] ordered by their increasing willingness to pay tolls to reduce latency. Thus $\alpha : [0,1] \to R^+$ is a nondecreasing function. For the case that α is a step function, Cole, Dodis, and Roughgarden show that optimal tolls can be computed by solving a linear program [5], under the following very strong assumption: The flow of users with valuation α_i in

[1] Cole, et al. use $l_e(f_e) + \beta(a)\tau_e$ to evaluate edge e. By taking $\alpha(a) = \frac{1}{\beta(a)}$, our notation is equivalent to theirs.

the Nash flow with the optimal tolls is known, even though the optimal tolls are unknown. The correctness of their algorithm relies on their nonconstructive proof of the existence of tolls.

What if these set of paths are not given? We show that in series parallel graphs, the set of paths travelled by users in any Nash flow with optimal tolls is *independent* of the valuations of time of the users: In series parallel graphs, the set of paths is determined by $\tilde{f}$ only. As a consequence, we give the first algorithm that computes tolls for users from a distribution given by *any* increasing function α (not restricted to a finite number of valuation classes), in series parallel graphs. For this we assume an oracle that given $a \in [0,1]$ returns $\alpha(a)$. We compute the tolls using at most $m+1$ oracle calls.

In general graphs, it is unknown if even verifying that a given set of tolls is optimal for a given α function is in P: Carstensen [4] constructs an example with fixed latencies and tolls where the number of paths that correspond to shortest paths for varying values of α can be exponential in the size of the graph.

We conclude by giving a simple example to demonstrate that if the graph is not series parallel, then the set of paths travelled by users in the Nash flow depends critically on the function α.

2 Preliminaries

Let $G = (V, E, l, s, t)$ denote a directed graph with nonnegative, nondecreasing, continuous latency functions l_e associated with each edge $e \in E$, source node $s \in V$ and sink node $t \in V$. Let $m = |E|$. The latency of edge e is a function solely of the flow on edge e. Given a set of edges F, and a function x defined on E, we denote by x_F the total of x on F: $x_F := \sum_{e \in F} x_e$, where x_e is the function value of x at $e \in E$. For a scalar x, we denote by $[x]^+$ the maximum in $\{x, 0\}$.

A path from s to t is an ordered subset of $V \times E$ of the form $(s = v_0, e_1, v_1, e_2, \ldots, e_k, v_k = t)$ with the property that $e_i = (v_{i-1}, v_i)$. For any subset Γ of $V \times E$, we denote by $E(\Gamma)$ the set $\Gamma \cap E$. Let $\mathcal{P}_{yz}$ be the set of y-z paths in G. For s-t paths, we simply use $\mathcal{P}$.

An s-t *flow* in G is a nonnegative function $f : E \to R^+ \cup \{0\}$ that satisfies *flow conservation* at all nodes of $V \backslash \{s,t\}$: $\sum_v f_{vw} = \sum_v f_{wv}$. The *volume* of a flow is the quantity of flow that leaves s, denoted $|f| := \sum_{v \in V} f_{sv}$. A *path flow* is a flow on a path from s to t in G. A *cycle flow* is a flow around a cycle in G. A *flow decomposition* of a flow f is a set $\Gamma = \{\gamma_1, \ldots, \gamma_r\}$ of path flows and cycle flows whose sum together is f: $\sum_{i=1}^{r} \gamma_i = \Gamma$. Every flow has a flow decomposition into at most $|E|$ paths and cycles. If f is acyclic, then the flow decomposition consists of paths only. A flow around a cycle may be *canceled* by sending the flow backward around the cycle, in effect subtracting the flow. A cycle is *canceled* if flow of value equal to the minimum flow value on an edge in the cycle is sent backward around the cycle. For more basic facts on flows, see [1].

The *cost* of an edge e with latency $l(e)$ and toll $\tau(e)$ for agent a is $\alpha(a)l(e) + \tau(e)$. The cost of a path P for agent a is simply the sum of the costs of the edges in the path. Unless otherwise specified, thoughout this paper, we assume that the latency of edge e is the latency of the edge in the system optimal flow. Thus, we define the *capacity* of the edge e as the value of flow on e in the system optimal flow: $|\tilde{f}_e|$, and the capacity of a path P to be $\min_{e \in P} |\tilde{f}_e|$.

When l is convex, the system optimal flow can be computed in polynomial time via solving a convex program. While the system optimal flow may not be unique, we will assume throughout the rest of this paper, that we are talking about an arbitrary, but fixed system optimal flow $\tilde{f}$.

Given congestion-aversion function $\alpha : [0,1] \to R^+$ and toll vector $\tau : E \to R^+ \cup \{0\}$, we denote the Nash flow by f_α^τ. When α is clear from context, as it is throughout most of the paper, we will simply use f^τ. Given Nash flow f^τ, we denote by $\gamma(a)$ the path used by user a in f^τ. The Nash flow exists, and has some interesting properties summarized in the following lemma [5].

Lemma 1. *For tolls τ, there exists a Nash flow f^τ with edge latencies l that satisfies*

i. For any path $P \in \mathcal{P}$, the agents assigned to P by f^τ form a (possibly empty or degenerate) subinterval of [0,1].
ii. If $a \le b$, then $l(\gamma(a)) \ge l(\gamma(b))$.
iii. If $a \le b$, then $\tau(\gamma(a)) \le \tau(\gamma(b))$.

Since the latency functions are nonnegative, we assume without loss of generality that the Nash flow and optimal flow induce directed, acyclic graphs. As long as l is nondecreasing and continuous, the Nash flow may be computed in general by solving a convex program.

3 Linear Tolls Are Necessary and Sufficient

Let $l_{\max}$ be the maximum latency of an edge in $\tilde{f}$. Clearly $l_{\max} \le \max_e l_e(1)$. Let $L = \max_{P \in \mathcal{P}} \sum_{e \in P} l_e(1)$.

Theorem 1. *Tolls that are bounded by $1 + \alpha(1)L$ suffice to induce a minimum latency flow as a Nash flow.*

Proof. Let $T = 1 + \alpha(1)L$. Let $\sigma(\tau_e) = \min\{T, [\tau_e + \frac{f_e^\tau}{\tilde{f}} - 1]^+\}$. We show that if σ has a fixed point τ', then $f_e^{\tau'} = \tilde{f}_e$ for all $e \in E$. Then, since σ is continuous [5] and bounded, we can invoke Brouwer's fixed point theorem [3] to obtain the result.

Suppose there is a "bad" fixed point — a fixed point τ of σ with $f^\tau \neq \tilde{f}$. Then every edge e with $f_e^\tau > \tilde{f}_e$ has $\tau_e = T$ (a *taxed* edge); and every edge e with $f_e^\tau < \tilde{f}_e$ has $\tau_e = 0$ (an *untaxed* edge). We create a graph $\hat{G}$ on V with an arc (v,w) with capacity $\tilde{f}(v,w) - f^\tau(v,w)$, if $f^\tau(v,w) < \tilde{f}(v,w)$ (a forward arc), and an arc (w,v) with capacity $f^\tau(v,w) - \tilde{f}(v,w)$, if $f^\tau(v,w) > \tilde{f}(v,w)$ (a

backward arc). In words, $\hat{G}$ is the graph of the flow $\tilde{f} - f^\tau$. If $\hat{G}$ is nonempty, then a flow decomposition of $\tilde{f} - f^\tau$ yields only cycles and no paths, since the volume of both flows is the same. Thus if $\hat{G}$ is nonempty, it contains a cycle, with at least one forward and one backward arc, since both f^τ and $\tilde{f}$ are acyclic.

Intuition: Suppose the cost of a forward arc for agent a is $\alpha(a)l_e(\tilde{f})$, and the cost of backward arc for agent a is $-\alpha(a)l_e(f^\tau) - T$. Let C be a cycle in $\hat{G}$ such that agent a travels on then counterpart to all backward edges on C. Since f^τ is Nash, C cannot have negative cost. Let $|C|$ be the number of taxed (backward) edges on a cycle C. Since $\tilde{f}$ and is acyclic, $|C| \geq 1$. Thus, $\alpha(a)l_C(\tilde{f}) \geq \alpha(a)l_C(f^\tau) + T|C|$, or $T \leq \alpha(a)(l_C(\tilde{f}) - l_C(f^\tau))/|C| \leq n * l_{\max}\alpha(1)$, a contradiction if $L := n * l_{\max}$.

To expand this intuition, we show that if we have a "bad" fixed point, then there exists an agent a with incentive to change its path $P(a)$, which contradicts f^τ being a Nash flow:

Consider a cycle C in $\hat{G}$. Let A be the set of agents such that $P(a) \cap C$ contains a taxed edge. For each $a \in A$, define $P_C(a)$ to be the smallest connected component of $P(a)$ that includes t and intersects C. Define $v(a)$ to be $P_C(a) \cap C$.

Claim. For each $a \in A$, there is an alternate path for some agent $b \in A$ from s to t that uses $P_C(a)$ instead of $P_C(b)$ and at least one fewer taxed edge from C than $P(b)$ does.

We prove this claim: There is a backward arc leaving $v(a)$ on C — it corresponds to an arc in $P(a)$. If the arc entering $v(a)$ on C is a backward arc, then there is distinct agent $b \in A$ such that $v(a) \in P(b)$. Thus b can follow $P(a)$ from $v(a)$ instead of using $P(b)$. In doing so, agent b will use at least one fewer taxed arc from C — it will not use the arc in E that corresponds to the backward arc entering $v(a)$. Otherwise, the arc entering $v(a)$ on C is a forward arc. Let $V_C(a)$ be the set of vertices that are endpoints of edges in $P(a) \cap C$. In this case, there is an agent $b \in A$ such that there is a $y \in V_C(b)$ such that there is no $w \in \bigcup_{i \in A} V_C(i)$ with w on the path from y to $v(a)$ on C. In other words, all arcs from y to $v(a)$ on C are forward arcs. Then b can be rerouted from y along C to $v(a)$ and then onto $P_C(a)$ to t. This path has fewer taxed edges on C than the path from y to t along $P(b)$, since, in particular, it does not include the arc leaving y that corresponds to a backward arc in C. This establishes the claim.

Let $P_C(a^*)$ be the least toll path among all $P_C(a)$, $a \in A$. By the claim, some agent $b \in A$ can replace its current path $P_C(b)$ by using at least one fewer taxed edge in C and the subpath $P_C(a^*)$. No additional edges outside $P(b) - C - P_C(b)$ are added to this new path for b. If somehow a cycle is created in this new path, this cycle is shortcut. Thus, the change in cost that agent b experiences by choosing this path instead is at most the difference in tolls of the two paths, plus the latency of the new path minus the toll of at least one taxed edge on C. This is

$$\tau_{P_C(a^*)} - \tau_{P_C(b)} + \alpha(b)L + [-T] \leq \alpha(b)L + [-T]$$

Since f^τ is a Nash flow, this must be ≥ 0. This implies that $T <= \alpha(1)L$, a contradiction. □

Remarks. 1. The bound in Theorem 1 also holds when there are multiple sources and a single sink (or multiple sinks and a single source).

2. Theorem 1 may be strengthened by bounding the toll on each edge separately. In the map γ for edge e we can replace T with $T_e := \min\{\alpha(a) \mid e \in \gamma(a)$ in Nash flow of optimal tolls$\}$. The result is that the toll for agent a is not more than the latency of the maximum latency path times *her* valuation of time.

Linear Tolls Are Necessary

The bound in Theorem 1 is trivially tight for uniform α: consider two edges from s to t, one with latency L, the other with latency Lx^r for $r > 0$. The Nash flow will send all flow on the edge with latency x^r. In order to make the both paths attractive to users at the optimal flow, a toll of value $\alpha L\ (1 - \frac{1}{1+r})$ must be imposed on the bottom edge. For r large, this approaches αL.

4 Computing Tolls for General α

In this section, we assume that the system optimal $\tilde{f}$ is given. We seek tolls τ such that the Nash flow with agent $a \in [0,1]$ seeking to minimize $\alpha(a)l_P(f^\tau) + \tau(P)$ is $\tilde{f}$. We make no assumptions on the nondecreasing function α which reflects the aversion of each agent to congestion. We assume that we have access to α via an oracle that responds to the query $a \in [0,1]$ with $\alpha(a)$.

All latencies in this section refer to the latency of an edge given flow $\tilde{f}$. Thus $l = l(\tilde{f})$ in this section.

4.1 Series Parallel Graphs

We describe an algorithm that computes optimal tolls if G is series-parallel.

Definitions. A basic series-parallel graph is an edge with terminals a and b. Two series parallel graphs can be joined in a *series composition* by associating terminal b of the first with terminal a of the second. Two series parallel graphs can be joined in a *parallel composition* by associating terminal a of the first with terminal a of the second, and associating terminal b of the first with terminal b of the second. A maximal set of contiguous series compositions is a *series component.* A maximal set of parallel compositions is a *parallel component.* Any series-parallel graph can be completely specified by its *composition tree.* The composition tree contains a node for every series component and parallel component. A node in the composition tree *contains* an edge e if both endpoints of e are in the component associated with the node. Node X is a child of node Y if $X \subset Y$ and there is no other component Z with $X \subset Z \subset Y$. In this way, all children of parallel component nodes are series component nodes, and vice versa.

Algorithm ComputeToll

Step 1. Create a longest-path-first flow decomposition of the minimum latency flow $\tilde{f}$: Find a longest latency path P in $\tilde{f}$, and set the volume of path flow γ along P to be the capacity of P in $\tilde{f}$. Remove γ by setting $\tilde{f} = \tilde{f} - \gamma$, and iterate. Ties are broken among paths by assigning a unique numerical key to each edges, and breaking ties by choosing the path with the highest key in the first edge. Let η be the number of paths in the decomposition. Note that $\eta \leq |E|$, since each path-flow removal reduces the support of $\tilde{f}$ by at least one edge. Let this collection of paths be $\Gamma = \{\gamma_1, \ldots, \gamma_\eta\}$, indexed in order of nonincreasing lengths l (so that γ_1 is the longest latency path).

Step 2. Assign agents to the path flows in Γ: The set of agents with the highest α value are assigned to the shortest latency path. That is, agents in $(1 - |\gamma_\eta|, 1]$ are assigned to γ_η. Agents with the next highest α values to next path; and so on, until agents in $[0, |\gamma_1|]$ are assigned to γ_1. In this way, the agents are partitioned into η groups according to the path to which they are assigned. Let $[\alpha_1, \beta_1], \ldots, (\alpha_\eta, \beta_\eta]$ be the ranges of α determined by this partition. Thus, $\alpha_i \leq \beta_i \leq \alpha_{i+1}$ for all i.

Step 3. Assign tolls to edges: From (G, l, α) create a new instance (G, l, α'), where α' is a step function that depends on α, as follows. Let $\alpha'_{2i-1} = \alpha_i$, and let $\alpha'_{2i} = \beta_i$. Let the volume of users of types α'_{2i-1} and α'_{2i}, denoted by r_{2i-1} and r_{2i} respectively, be each equal to $|\gamma_i|/2$.

Find a feasible solution to the following set of inequalities in variables z and τ. The resulting value τ_e is the toll for edge e.

$$\begin{aligned} z_s^i &= 0 && \forall\, 1 \leq i \leq 2\eta \\ z_w^i - z_v^i &\leq \alpha'_i l_{vw}(\tilde{f}_{vw}) + \tau_{vw} && \forall\, i,\ \forall\, (v,w) \in E(G) \\ \textstyle\sum_{i=1}^{2\eta} r_i z_t^i &= \textstyle\sum_{i=1}^{2\eta} \sum_{e \in \gamma_{\lceil i/2 \rceil}} [\alpha_i l_e(\tilde{f}_e) + \tau_e] r_i \end{aligned} \qquad (1)$$

Analysis of Algorithm ComputeToll. Let $\mathrm{dist}_l(v, w, F)$ be the latency of the least l-latency path between v and w using edges in F. A set of paths $\{\gamma_1, \gamma_2, \ldots, \gamma_r\}$ in G with edge-length function l is said to have the *decreasing subpaths property* if for all $i < j$, $\{v, w\} \subset V(\gamma_i) \cap V(\gamma_j)$ implies that $\mathrm{dist}_l(v, w, \gamma_i) \geq \mathrm{dist}_l(v, w, \gamma_j)$.

Lemma 1 has the following simple corollary.

Corollary 1. *For any tolls τ, there exists a Nash flow f with edge latencies l such that for all $a \leq b$ the ordered set $\{\gamma(a), \gamma(b)\}$ satisfies the decreasing subpaths property.*

We now show that in series parallel graphs, a set of paths has the decreasing subpaths property if an only if it corresponds to a longest path decomposition.

Lemma 2. *Any path decomposition of a series parallel graph G is a longest path decomposition if and only if it has the decreasing subpaths property.*

Proof. Let $\Gamma = \{\gamma_1, \ldots, \gamma_\eta\}$ be a longest path decomposition. Consider any two indices $1 \leq i < j \leq \eta$ and vertices $\{v, w\} \in V(\gamma_i) \cap V(\gamma_j)$ such that $\text{dist}_l(v, w, E(\gamma_i)) < \text{dist}_l(v, w, E(\gamma_j))$. Swapping the subpath of γ_j from v to w with the parallel subpath of γ_i results in a modified path decomposition with γ_i longer than before the swap. This contradicts that γ_i is from a longest path decomposition. Thus Γ satisfies the decreasing subpaths property.

Now suppose $\Gamma = \{\gamma_1, \ldots, \gamma_\eta\}$ is a decomposition of a series parallel graph G that satisfies the decreasing subpaths property. In order to show that Γ is a longest path decomposition it is sufficient to show that γ_1 is a longest path. Then, by induction, since $\Gamma - \gamma_1$ is a decomposition of $G - \gamma_1$ that satisfies the decreasing subpaths property, $\Gamma - \gamma_1$ is a longest paths decomposition of $G - \gamma_1$.

Suppose γ_1 not a longest path. Let Y be the smallest parallel component in which γ_1 is not a longest path through Y. Let s' and t' be the end nodes of Y. By definition of Y, a longest path through Y is internally node disjoint with γ_1. Call one such longest path through Y by p. Since Γ obeys the decreasing subpaths property, edges in the subpath p cannot be on just one path in Γ, and thus must appear on at least two paths Γ. Let γ_j be the last such path, and γ_i be some other such path that satisfies the following properties: there is a smallest parallel component X such that γ_i and γ_j intersect X, $\gamma_j \cap E(X) = p$, and $\gamma_i \cap E(X)$ is not a longest path in X. Since both $\gamma_j \cap E(Y)$ and $\gamma_i \cap E(Y)$ must have length less than p by the decreasing subpaths property, such a component must exist if p is not on γ_1. But this contradicts the fact that $\{\gamma_i, \gamma_j\}$ obeys the decreasing subpaths property. Thus γ_1 must be a longest path in G. □

A simple consequence of Lemma 2 is that a longest path decomposition of series parallel graph G is also a shortest path decomposition, since a symmetric argument shows that a shortest path decomposition obeys a symmetric increasing subpaths property.

Together Lemma 2 and Corollary 1 imply that the set of paths used by users in any Nash flow forms a longest path decomposition of G. Thus, even without knowing the exact distribution α, we know 1) set of paths travelled in G by any set of selfish users; 2) if the users are ordered according to α value, we know which users travel on which path.

We now invoke a theorem of Cole, Dodis, and Roughgarden [5] that states that for an instance (G, l, α) such that α is a step function, if the path decomposition of the Nash flow with optimal tolls τ is known, then it is possible to compute τ by finding a feasible solution to a set of inequalities. We paraphrase their Theorem 4.2 and the discussion that precedes it below.

Theorem 2 (Cole, Dodis, Roughgarden). *Let (G, l, α) be an instance in which α takes on only finitely many distinct values. Let r_i be the volume of users with valuation α_i. Suppose τ induces Nash flow $\tilde{f}$, and let $\tilde{f}^i$ be the flow induced by users with valuation α_i. Then τ and $\tilde{f}$ satisfy the following system of inequalities.*

$$
\begin{aligned}
z_s^i &= 0 && \forall i \\
z_w^i - z_v^i &\le \alpha_i' l_{vw}(\tilde{f}_{vw}) + \tau_{vw} && \forall\, i,\ \forall\, (v,w) \in E(G) \\
\textstyle\sum_i r_i z_t^i &= \textstyle\sum_i \sum_{e \in E} [\alpha_i l_e(\tilde{f}_e) + \tau_e] \tilde{f}_e^i
\end{aligned}
$$

A corollary of this theorem is that if $\tilde{f}^i$ is known, and the number of distinct values of α is polynomial, then τ can be computed in polynomial time.

Theorem 3. *For any instance* (G, l, α) *where G is series-parallel, and* α *is an arbitrary increasing function on* [0,1], *algorithm* ComputeToll *finds the optimal tolls.*

Proof. By Steps 1 and 2, Corollary 1, and Lemma 2, the instance described in Step 3 of ComputeToll is of the form required by Theorem 2. Thus, the solution to this system of inequalities yields optimal tolls for the problem with valuation function α'. Together Corollary 1 and Lemma 2 imply that the set of paths used with valuation function α is the same set of paths used with valuation function α'. Lemma 1 i. and ii. implies that the bounds for the α values for the users on such paths is the same as the bounds for the α' values. Since the tolls are okay for the extremes of users on each path, they are okay for all users on the paths, and hence the tolls computed for α' are also optimal for α.

4.2 Other Graphs

If G is not series parallel, then for different functions α, flow patterns of agents with optimal tolls may be different. Thus, there is no universal flow decomposition that holds for all α.

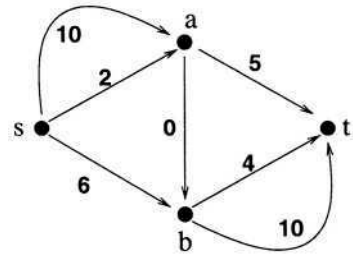


Fig. 1. In this network, the flow patterns of agents in the Nash flow with optimal tolls depend on the distribution α. The number on each arc represents the latency of the arc in the system optimal flow.

For example, consider the graph on 4 nodes $\{s, a, b, t\}$ with arc set and optimal latencies $\{(s,a,2), (s,a,10), (s,b,6), (a,b,0), (a,t,5), (b,t,4), (b,t,10)\}$ depicted in the above figure. This graph is not series-parallel, but would be series-parallel without any one of the arcs (s,b), (a,b), or (a,t).

If $\alpha(a) = 1$ for $a \in [0, 1/3]$, $\alpha(a) = 6/5$ for $a \in (1/3, 1/2]$, $\alpha(a) = 4/3$ for $a \in (1/2, 5/6]$ and $\alpha(a) = 2$ for $a \in [5/6, 1]$, then the optimal toll vector is (10,0,4,0,5,7,0) and the paths taken by users in the Nash flow are $\{(s,a,2),(a,t,5)\}, \{(s,a,10),(a,b,0),(b,t,10)\}$, and $\{(s,b,6),(b,t,4)\}$.

On the other hand, if $\alpha(a) = 1$ for $a \in [0, 2/3)$ and $\alpha(a) = 5$ for $a \in [2/3, 1]$, then the optimal toll vector is (8,0,0,0,1,6,0) and the paths taken by users in the Nash flow are $\{(s,a,2),(a,b,0),(b,t,4)\}, \{(s,a,10),(a,t,5)\}$, and $\{(s,b,6),(b,t,10)\}$.

5 Conclusions

In this paper we have provided a improved bound on the size of tolls needed to induce heterogeneous, selfish users to obey the system optimal flow in single source networks; and provided an algorithm to compute such tolls in series parallel networks. This work was motivated by an interest in understanding the tolls problem better so as to address the existence and computation of tolls in multicommodity networks. In joint work with Kamal Jain and Mohammad Mahdian, we have recently proved the existence of tolls for heterogeneous users in multicommodity networks. In the process, we have given the first constructive proof of existence of tolls for not only the system optimal flow, but for any *minimal congestion.* We give a simple algorithm for computing tolls via solving a linear program. Our work extends to general nonatomic congestion games [7].

Acknowledgement. I would like to thank Tim Roughgarden for interesting discussions on this problem.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.
2. M. Beckman, C. B. McGuire, and C. B. Winsten. *Studies in the Economics of Transportation.* Yale University Press, 1956.
3. K. C. Border. *Fixed Point Theorems with Applications to Economics and Game Theory.* Cambridge, 1985.
4. P. J. Carstensen. Parametric cost shortest path problems. Unpoblished Bellcore memo, 1984.
5. R. Cole, Y. Dodis, and T. Roughgarden. Pricing network edges for heterogeneous selfish users. In *Proc. 35th Annual ACM Symposium on the Theory of Computing,* 2003.
6. S. C. Dafermos. Toll patterns for multiclass-user transportation networks. *Transportation Sci.,* 7:211–223, 1973.
7. L. Fleischer, K. Jain, and M. Mahdian. Taxes for heterogeneous selfish users in a multicommodity network. Submitted, April 2004.
8. Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS),* volume 1563 of *Lecture Notes in Computer Science,* pages 404–413, 1999.

9. A. C. Pigou. *The Economics of Welfare*. Macmillan, 1920.
10. Tim Roughgarden and Eva Tardos. How bad is selfish routing? In *IEEE Symposium on Foundations of Computer Science,* pages 93–102, 2000.
11. M. J. Smith. The marginal cost taxation of a transportation network. *Trans. Res. Ser. B,* 13:237–242, 1979.
12. J. G. Wardrop. Some theoretical aspects of rad traffic research. In *Proc. Institute of Civil Engineers, Pt. II,* volume 1, pages 325–378, 1952.

Bounded Fixed-Parameter Tractability and $\log^2 n$ Nondeterministic Bits

Jörg Flum[1], Martin Grohe[2], and Mark Weyer[1]

[1] Abteilung für Mathematische Logik, Albert-Ludwigs-Universität, Freiburg, Germany
flum@uni-freiburg.de,Mark.Weyer@math.uni-freiburg.de
[2] Institut für Informatik, Humboldt-Universität, Berlin, Germany
grohe@informatik.hu-berlin.de

Abstract. Motivated by recent results showing that there are natural parameterized problems that are fixed-parameter tractable, but can only be solved by fixed-parameter tractable algorithms the running time of which depends non-elementarily on the parameter, we propose a notion of *bounded fixed-parameter tractability,* where the dependence of the running time on the parameter is restricted to be singly exponential.

We develop a basic theory that is centred around the class EPT of tractable problems and an EW-hierarchy of classes of intractable problems, both in the bounded sense. By and large, this theory is similar to the established *unbounded* parameterized complexity theory, but there are some remarkable differences. Most notably, certain natural model-checking problems that are known to be fixed-parameter tractable in the unbounded sense have a very high complexity in the bounded theory. The problem of computing the VC-dimension of a family of sets, which is known to be complete for the class W[1] in the unbounded theory, is complete for the class EW[3] in the bounded theory.

It turns out that our bounded parameterized complexity theory is closely related to the classical complexity theory of problems that can be solved by a nondeterministic polynomial time algorithm that only uses $\log^2 n$ nondeterministic bits, and in particular to the classes LOGSNP and LOGNP introduced by Papadimitriou and Yannakakis [15].

1 Introduction

The idea of fixed-parameter tractability is to approach hard algorithmic problems by isolating problem parameters that can be expected to be small in certain applications and then develop algorithms that are polynomial except for an arbitrary dependence on the parameter. More precisely, a problem is fixed-parameter tractable if it can be solved by an algorithm the running time of which is bounded by $f(k) \cdot p(n)$, where n denotes the size of the input, k the parameter, f is an arbitrary computable function, and p a polynomial. Since the choice of suitable parameters allows for a great flexibility, fixed-parameter algorithms have found their way into practical applications such diverse as computational biology, database systems, computational linguistics, and automated verification (cf. [3]). On the theoretical side, a theory of parameterized intractability has been developed that led to a comprehensive classification of parameterized problems into tractable and hard problems (cf. [6,3]).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 555–567, 2004.

Allowing an arbitrary computable function f in the running time bound of a fixed-parameter tractable algorithm seems questionable, though. A running time of $2^{2^k}\mathrm{poly}(n)$ cannot really be considered "tractable" even for small values of k (say, $k \leq 10$). The standard and to some extent valid response to such objections is that (a) for natural problems, such extreme parameter dependence rarely occurs and (b) to obtain a robust theory, one has to compromise. Referring to the "classical" class of tractable problems, polynomial time, one may add that (c) an algorithm with a running time of $O(n^{100})$ cannot be considered "tractable" either, even though it is a polynomial time algorithm. However, recent results due to Frick and the second author [12] show that the crucial point (a) is very questionable: There are natural fixed-parameter tractable problems that cannot be solved by an algorithm whose running time is bounded by $f(k)\mathrm{poly}(n)$ for any elementary function f. These problems are so-called model-checking problems; database query evaluation is an application that can be described by such problems [13]. The results imply that the running time of the fixed-parameter tractable algorithm obtained from Courcelle's well-known theorem [2] that monadic second-order properties of graphs of bounded tree-width can be decided in linear time also has a non-elementary dependence on the parameter. Courcelle's theorem has been viewed a centre piece of parameterized complexity theory (a long chapter in Downey and Fellows' monograph [6] is devoted to Courcelle's theorem). This raises some doubts about the theory of fixed-parameter tractability. Of course these doubts by no means diminish the value of the practical work on fixed-parameter tractable algorithms; algorithms developed in this context often have running times $c^k \cdot n$ for some constant c with $1 < c < 2$.

The important fact is that there are viable alternatives to the notion of fixed-parameter tractability: One can simply put upper bounds on the growth of the "parameter dependence" f, the two most natural being $f \in 2^{\mathrm{poly}(k)}$ and the stricter $f \in 2^{O(k)}$. The resulting *bounded fixed-parameter tractability* classes are still fairly robust, and they contain all of the problems that are "fixed-parameter tractable in practice". While we do not want to propose an industry generating papers on various bounded parameterized complexity theories, we hope that our results will convince the reader that at least the bounded theory we consider here is well worth being explored.

We study the stricter notion of bounded fixed-parameter tractability. We let EPT be the class of all parameterized problems that can be solved in time $2^{O(k)} \cdot \mathrm{poly}(n)$. We introduce a suitable notion of *ept-reduction* and define a hierarchy of classes $\mathbf{EW}[t]$, for $t \geq 1$, corresponding to the classes of the W-hierarchy of *unbounded* parameterized complexity.[1] We observe that, for all $t \geq 1$, if $\mathbf{W}[t] \neq \mathbf{FPT}$ then $\mathbf{EW}[t] \neq \mathbf{EPT}$. So we can assume that the EW-hierarchy does not collapse to EPT (that is, if we believe the assumption of the unbounded theory that the W-hierarchy does not collapse to FPT). We prove that the logical characterisations of the W-hierarchy [7,9,10] can be transfered to

[1] Some remarks on our terminology may be helpful here: *Classical complexity theory* refers to the standard, unparameterized, complexity theory. In parameterized complexity, we distinguish between the usual theory, referred to as *unbounded parameterized complexity theory* and the new *bounded parameterized complexity theory.*

Furthermore, we distinguish between *classical problems,* which are just languages $Q \subseteq \Sigma^*$ over some finite alphabet Σ, and *parameterized problems,* which are pairs (Q, κ), where $Q \subseteq \Sigma^*$ is a classical problem and $\kappa : \Sigma^* \to \mathbb{N}$ a *parameterization.*

the bounded EW-hierarchy, which shows that the classes have a certain robustness. It has to be said, though, that the EW-hierarchy is less robust than the W-hierarchy. This is particularly true for the first level EW[1] of the hierarchy.

We then consider a few complete problems for our classes. Many completeness results can easily be transfered from the unbounded to the bounded theory. As an example, we prove that the parameterized dominating set problem, which is W[2]-complete under fpt-reductions, is EW[2]-complete under ept-reductions. A surprise occurs when we consider a parameterized version of the problem of computing the VC-dimension of a family of sets. In the unbounded theory, this problem is known to be W[1]-complete under fpt-reductions. We prove that in our bounded theory, VC-dimension is EW[3]-complete under ept-reductions. Thus we are in the odd situation that in the unbounded theory, VC-dimension is "easier" than dominating set, whereas in the bounded theory, it is "harder". The completeness of the parameterized VC-dimension problem for the third level of our hierarchy seems very natural in view of Schaefer's result that a classical version of the VC-dimension problem, where the family of sets is represented succinctly, is complete for the third level of the polynomial hierarchy [16].

Less surprisingly, we prove that the (unbounded) fixed-parameter tractable model-checking problems that have been shown to have no fixed-parameter tractable algorithms with elementary parameter dependence in [12] are complete for natural intractable classes in the bounded theory. Specifically, we prove that model-checking for first-order logic on words is complete for the class $\mathbf{EAW}[*]$, the bounded analogue of the class $\mathbf{AW}[*]$.

One of the nicest features of our bounded theory is that it is intimately linked to the classical complexity theory of problems that can be solved by a nondeterministic polynomial time algorithm that only uses $\log^2 n$ nondeterministic bits. There are several natural examples of such problems, the best known may be the problem of computing the VC-dimension of a given family of sets [15] and the hypergraph traversal problem [8]. Papadimitriou and Yannakakis [15] introduced two syntactically defined complexity classes LOGSNP and LOGNP and proved that many natural problems are complete for one of these classes. The definition of these classes is reminiscent of some of the logical characterisations of the classes of the W-hierarchy and the EW-hierarchy. Motivated by this observation, we introduce a hierarchy of classical complexity classes $\mathbf{LOG}[t]$, for $t \geq 1$, which may be viewed as restrictions of the corresponding classes $\mathbf{EW}[t]$ to the parameter value $\log n$, where n denotes the size of the input. We prove that LOGSNP = LOG[2] and LOGNP = LOG[3]. Thus our classes put Papadimitriou and Yannakakis's classes into a larger context. We then show that for all $t \geq 1$ we have $\mathbf{LOG}[t] = \mathbf{PTIME}$ if, and only if, $\mathbf{EW}[t] = \mathbf{EPT}$. This establishes a nice direct connection between classical complexity theory and our bounded parameterized theory; no such connection is known for the W-hierarchy (and it probably does not exist).

Our paper is organised as follows: In Section 2 we review the basic notions of (unbounded) parameterized complexity theory and at the same time introduce the corresponding notions of the bounded theory. In Section 3, we give logical characterisations of the EW-hierarchy, and in Section 4 we prove two basic completeness results for the classes EW[2] and EW[3]. Section 5 is devoted to the connection between our bounded parameterized complexity theory and the classical theory of problems that can be solved

with $\log^2 n$ nondeterministic bits. In Section 6, we study higher levels of intractability in our bounded theory. Finally, Section 7 is devoted to the first level of the EW-hierarchy, which unfortunately does not share the robustness of the higher levels. Due to space limitations, we have to defer the proofs to the full version of the paper [11].

2 The Basic Notions

2.1 FPT and EPT

Let Σ be a finite alphabet. A *parameterized problem* (over the alphabet Σ) is a pair (Q, κ) consisting of a set $Q \subseteq \Sigma^*$ of strings over Σ and a polynomial time computable function $\kappa : \Sigma^* \to \mathbb{N}$, the *parameterization.* Any $x \in \Sigma^*$ is called an *instance* of Q and $\kappa(x)$ is the corresponding *parameter.*

Hence, a parameterized problem consists of a problem in the usual complexity theoretic sense together with a parameterization.

For example, choose a finite alphabet Σ such that propositional formulas are strings over Σ in a natural way. The parameterized problem p-SAT is the problem (Q, κ), where Q is the set of satisfiable propositional formulas and $\kappa : \Sigma^* \to \mathbb{N}$ is defined by

$$\kappa(x) := \begin{cases} \text{number of variables of } x, & \text{if } x \text{ is a propositional formula} \\ 0, & \text{otherwise.} \end{cases}$$

The following notation for p-SAT illustrates how we normally present parameterized problems:

p-SAT
Input: A propositional formula α.
Parameter: The number of variables of α.
Problem: Decide if α is satisfiable.

Definition 1. Let $\mathfrak{F}$ be a set of total functions from $\mathbb{N}$ to $\mathbb{N}$. A parameterized problem (Q, κ) over the alphabet Σ is ***$\mathfrak{F}$-fixed-parameter*** *tractable,* if there is a function $f \in \mathfrak{F}$, a polynomial $p \in \mathbb{N}[X]$, and an algorithm that, given $x \in \Sigma^*$, decides if $x \in Q$ in at most $f(\kappa(x)) \cdot p(|x|)$ steps.

We denote the class of all **$\mathfrak{F}$-fixed-parameter** tractable problems by **$\mathfrak{F}$-FPT.**

The standard notion of fixed-parameter tractability is based on the class $\mathfrak{R}$ of all computable functions. We use the standard terminology and denote the class **$\mathfrak{R}$-FPT** simply by FPT.[2] We usually refer to the "standard" parameterized complexity theory based on the class FPT as *unbounded (parameterized complexity) theory,* to distinguish it from *bounded theories* based on $\mathfrak{F}$ for "bounded" classes $\mathfrak{F}$.

In this paper, we are mainly interested in **$\mathfrak{E}$-fixed-parameter** tractability, where

$$\mathfrak{E} = 2^{O(k)}.$$

[2] Sometimes, FPT is even defined as **$\mathfrak{A}$-FPT,** where $\mathfrak{A}$ denotes the class of all functions $f : \mathbb{N} \to \mathbb{N}$. (Downey and Fellows [6] call **$\mathfrak{R}$-FPT** *strongly uniform FPT* and **$\mathfrak{A}$-FPT** *uniform-FPT.*) However, **$\mathfrak{R}$-FPT** is a more robust class.

To simplify the notation, we write EPT instead of $\mathfrak{E}$-FPT. Further natural and interesting classes are $\mathfrak{SUBE}$-FPT and $\mathfrak{EXP}$-FPT, where $\mathfrak{SUBE} = 2^{o(k)}$ and $\mathfrak{EXP} = 2^{\text{poly}(k)}$. The latter one will be investigated in detail in the third author's forthcoming PhD-thesis [17]. If $\mathfrak{F}$ is a nonempty set of polynomially bounded functions, then $\mathfrak{F}$-FPT is PTIME.or more precisely, $\mathfrak{F}$-FPT is the class of parameterized problems (Q, κ) with Q in PTIME.

Clearly, if $\mathfrak{F} \subseteq \mathfrak{F}'$ then $\mathfrak{F}$-FPT $\subseteq \mathfrak{F}'$-FPT and hence, every problem in EPT is in FPT. An example of a problem in EPT is p-SAT, where we can choose as f the function $f(k) := 2^k$. If $Q \subseteq \Sigma^*$ is a decidable problem that is not decidable in time $2^{O(n)}$ and $\kappa : \Sigma^* \to \mathbb{N}$ is defined by $\kappa(x) = |x|$, then the parameterized problem (Q, κ) is in FPT \ EPT. *Natural* problems in FPT \ EPT are known to exist under certain complexity theoretic assumptions:

$$p\text{-MC(WORDS, FO)} \in \text{FPT} \setminus \text{EPT} \quad \text{if FPT} \neq \text{AW}[*],$$
$$p\text{-MC(WORDS, MSO)} \in \text{FPT} \setminus \text{EPT} \quad \text{if P} \neq \text{NP}$$

(cf. [12]). Here, p-MC(WORDS, FO) and p-MC(WORDS, MSO) denote the parameterized model-checking problem for the class of words and first-order logic FO and the class of words and monadic second-order logic MSO, respectively (compare 3.2 for the definition of model-checking problems).

2.2 Reductions

To compare the complexities of parameterized problems that are not $\mathfrak{F}$-fixed-parameter tractable, we need a notion of reduction. We only consider many-one reductions. The crucial property expected from a notion of reduction for $\mathfrak{F}$-FPT is:

$$\text{If } (Q, \kappa) \text{ is reducible to } (Q', \kappa') \text{ and } (Q', \kappa') \in \mathfrak{F}\text{-FPT, then } (Q, \kappa) \in \mathfrak{F}\text{-FPT.} \quad (2.1)$$

We give the definitions for the cases we are interested in here, FPT and EPT.

Definition 2. Let (Q, κ) and (Q', κ') be parameterized problems over the alphabets Σ and Σ', respectively. A *reduction* from (Q, κ) to (Q', κ') is a function $R : \Sigma^* \to (\Sigma')^*$ with $\big(Qx \iff Q' R(x)\big)$ for all $x \in \Sigma^*$.

(1) R is an *fpt-reduction* if there are computable functions f, g and a polynomial p such that
   a) $R(x)$ is computable in time $f(\kappa(x)) \cdot p(|x|)$,
   b) $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

(2) R is an *ept-reduction* if there are constants $c, d \geq 0$ and a polynomial p such that
   a) $R(x)$ is computable in time $2^{c \cdot \kappa(x)} \cdot p(|x|)$,
   b) $\kappa'(R(x)) \leq d \cdot (\kappa(x) + \log |x|)$ for all $x \in \Sigma^*$.

For $x \in \{\text{e}, \text{f}\}$ we write $(Q, \kappa) \leq^{x\text{pt}} (Q', \kappa')$ if there is an $x$pt-reduction from (Q, κ) to (Q', κ') and $(Q, \kappa) \equiv^{x\text{pt}} (Q', \kappa')$ if $(Q, \kappa) \leq^{x\text{pt}} (Q', \kappa')$ and $(Q', \kappa') \leq^{x\text{pt}} (Q, \kappa)$. We let

$$[(Q, \kappa)]^{x\text{pt}} = \{(Q', \kappa') \mid (Q', \kappa') \leq^{x\text{pt}} (Q, \kappa)\}.$$

It is easy to verify (2.1) for ept-reducibility with respect to EPT and fpt-reducibility with respect to FPT.

The two notions of reduction are incomparable: To see this, let $Q \subseteq \Sigma^*$ be a problem that is not in polynomial time. Let $\kappa, \kappa' : Q \to \mathbb{N}$ be defined by $\kappa(x) = 1$ and $\kappa'(x) = \log |x|$ for all $x \in \Sigma^*$. Then clearly (Q, κ) is ept-reducible to (Q, κ'). However, it is easy to see that if (Q, κ) was fpt-reducible to (Q, κ') then Q would be in polynomial time.

Conversely, let (Q, κ) be any problem in FPT \ EPT, and let (Q', κ') be any non-trivial problem in EPT (non-trivial meaning that Q' is neither the empty set nor the set of all strings over a given alphabet). Then (Q, κ) is fpt-reducible, but not ept-reducible to (Q', κ').

2.3 The W-Hierarchy and the EW-Hierarchy

In unbounded parameterized complexity theory, most complexity classes of intractable problems were originally defined by means of weighted satisfiability problems for propositional logic. We recall the definitions and extend them to EPT.

Formulas of propositional logic are built up from *propositional variables* by taking conjunctions, disjunctions, and negations. The negation of a formula α is denoted by $\neg\alpha$. We distinguish between *small conjunctions,* denoted by $\wedge$, which are just conjunctions of two formulas, and *big conjunctions,* denoted by $\bigwedge$, which are conjunctions of arbitrary finite sequences of formulas. Analogously, we distinguish between *small disjunctions,* denoted by $\vee$, and *big disjunctions,* denoted by $\bigvee$. Every formula has a naturally defined *syntax tree,* and the size $|\alpha|$ of a formula α is the number of nodes of the syntax tree of α.

The *weight* of an assignment is the number of variables set to TRUE. A propositional formula α is ***k-satisfiable*** (where $k \in \mathbb{N}$), if there is an assignment for the set of variables of α of weight k satisfying α.

For a set Γ of propositional formulas, the *parameterized weighted satisfiability problem* $\mathbf{WSAT}(\Gamma)$ *for formulas in* Γ is the following parameterized problem:

p-WSAT(Γ)
Input: A propositional formula $\alpha \in \Gamma$ and $k \in \mathbb{N}$.
Parameter: k.
Problem: Decide if α is k-satisfiable.

For $t \geq 0$ and $d \geq 1$ define the sets $\Gamma_{t,d}$ and $\Delta_{t,d}$ by induction on t (here, by $(\lambda_1 \wedge \ldots \wedge \lambda_r)$ we mean the iterated small conjunction $((\ldots(\lambda_1 \wedge \lambda_2)\ldots) \wedge \lambda_r)$:

$$\begin{aligned}
\Gamma_{0,d} &:= \{(\lambda_1 \wedge \ldots \wedge \lambda_r) \mid \lambda_1, \ldots, \lambda_r \text{ literals and } r \leq d\},\\
\Delta_{0,d} &:= \{(\lambda_1 \vee \ldots \vee \lambda_r) \mid \lambda_1, \ldots, \lambda_r \text{ literals and } r \leq d\},\\
\Gamma_{t+1,d} &:= \{\textstyle\bigwedge \Pi \mid \Pi \subseteq \Delta_{t,d}\},\\
\Delta_{t+1,d} &:= \{\textstyle\bigvee \Pi \mid \Pi \subseteq \Gamma_{t,d}\}.
\end{aligned}$$

If in the definition of $\Gamma_{0,d}$ and $\Delta_{0,d}$ we require that all literals are positive (negative) we obtain the sets denoted by $\Gamma^+_{t,d}$ and $\Delta^+_{t,d}$ ($\Gamma^-_{t,d}$ and $\Delta^-_{t,d}$), respectively.

In unbounded parameterized complexity the classes W[1], W[2],... constitute the W-hierarchy; for $t \geq 2$,

$$\mathrm{W}[t] = [p\text{-}\mathrm{WSAT}(\Gamma_{t,1})]^{\mathrm{fpt}},$$

the case $t = 1$ will be treated separately in Section 7. We take the corresponding equality as definition for a hierarchy of classes in **&-parameterized** complexity theory:

$$\mathrm{EW}[t] := [p\text{-}\mathrm{WSAT}(\Gamma_{t,1})]^{\mathrm{ept}}.$$

Remark 3. The classes $\mathrm{W}[t]$ are robust in the sense that for all $d \geq 1$ we have $\mathrm{W}[t] = [p\text{-}\mathrm{WSAT}(\Gamma_{t,d})]^{\mathrm{fpt}}$. This robustness does not seem to be shared by the classes $\mathrm{EW}[t]$. Instead, we have to consider a matrix EW of classes given by

$$\mathrm{EW}[t,d] := [p\text{-}\mathrm{WSAT}(\Gamma_{t,d})]^{\mathrm{ept}}.$$

In this extended abstract we will only deal with the EW-hierarchy, that is, the classes $\mathrm{EW}[t,1]$ for $t \geq 2$. Our logical characterisations of the hierarchy have natural extensions to the full matrix; details can be found in the full version of the paper [11].

3 Logical Characterisations of the EW-Hierarchy

In this section we present characterisations of the classes $\mathrm{EW}[t]$, first in terms of variants of the weighted satisfiability problems defining the classes, then in terms of model-checking problems for first-order logic and finally, in terms of Fagin-definable problems. Most results (and their proofs) are extensions or refinements of the corresponding characterisations of $\mathrm{W}[t]$.

3.1 Propositional Logic

The notions of complete and hard problem for a complexity class are defined in the usual fashion; they refer to fpt-reductions or to ept-reductions depending on whether we consider a class of unbounded parameterized complexity theory or a class of **&-parameterized** complexity theory.

For even (odd) t already the weighted satisfiability problem for monotone (anti-monotone) propositional formulas is complete for $\mathrm{EW}[t]$:

Theorem 4. *(1)* $p\text{-}\mathrm{WSAT}(\Gamma_{t,1}^{+})$ *is complete for* $\mathrm{EW}[t]$ *for even* $t > 1$.
(2) $p\text{-}\mathrm{WSAT}(\Gamma_{t,1}^{-})$ *is complete for* $\mathrm{EW}[t]$ *for odd* $t > 1$.

3.2 Model-Checking Problems

In this and the following subsection, we will work with formulas of first-order logic. We only consider finite relational vocabularies and finite structures. We assume that the reader is familiar with first-order logic.

The size of a structure $\mathcal{A}$ is denoted by $||\mathcal{A}||$, and the size of a first-order formula φ by $|\varphi|$. The class of all first-order formulas is denoted by FO. For $t \geq 1$, let Σ_t be the class of all FO-formulas of the form

$$\exists x_{11} \ldots \exists x_{1k_1} \forall x_{21} \ldots \forall x_{2k_2} \ \ldots \ Q x_{t1} \ldots Q x_{tk_t} \ \psi,$$

where $Q = \forall$ if t is even and $Q = \exists$ otherwise, and where ψ is quantifier-free. The class of **Π_t-formulas** is defined analogously starting with a block of universal quantifiers. Let $t, u \geq 1$. A formula φ is **$\Sigma_{t,u}$**, if it is Σ_t and all quantifier blocks after the leading existential block have **length** $\leq u$. For example, a formula $\exists x_1 \ldots \exists x_k \forall y \exists z_1 \exists z_2 \psi$, where ψ is quantifier-free, is in $\Sigma_{3,2}$ (for every $k \geq 1$).

For a class C of structures and a class Φ of formulas, the *parameterized model-checking problem for structures in C and formulas in Φ* is defined as follows:

p-MC(C, Φ)
Input: A structure $\mathcal{A}$ in C and a sentence φ in Φ.
Parameter: $|\varphi|$.
Problem: Decide if $\mathcal{A}$ satisfies φ.

If C is the class of all structures, we denote **p-MC(C,Φ)** by **p-MC(Φ)**.

The characterisation of the classes of the EW-hierarchy in terms of model-checking problems reads as follows:

Theorem5. *p-MC($\Sigma_{t,u}$) is complete for* **EW**$[t]$ *for all $t \geq 2$ and $u \geq 1$.*

3.3 Fagin-Definability

In [7,9], two notions of definability of parameterized problems were introduced. The first is definability via model-checking problems, which we have considered in the previous section. The second is *Fagin-definability,* which we will consider now.

For every first-order formula $\varphi(X)$ with a free set variable X we let **p-FD$_{\varphi(X)}$** be the following parameterized problem:

p-FD$_{\varphi(X)}$
Input: A structure $\mathcal{A}$ and $k \in \mathbb{N}$.
Parameter: k.
Problem: Decide if there is a subset S of A of cardinality k satisfying $\varphi(X)$ in $\mathcal{A}$, that is, with $\mathcal{A} \models \varphi(S)$.

We say that $\varphi(X)$ *Fagin-defines* the problem **p-FD$_{\varphi(X)}$**.

For all formulas φ, individual variables x, and set variables X, we write $\exists x \in X \ \varphi$ as an abbreviation of $\exists x(Xx \wedge \varphi)$ and $\forall x \in X \ \varphi$ as an abbreviation of $\forall x(Xx \rightarrow \varphi)$. For $t, d \geq 1$, we let $\Pi_{t/d}$ be the class of formulas $\varphi(X)$ of the form

$$\forall \bar{y}_1 \exists \bar{y}_2 \ldots \forall \bar{y}_{t-1} \exists z_1 \in X \ldots \exists z_d \in X \ \psi \tag{3.1}$$

in case t is even, and of the form

$$\forall \bar{y}_1 \exists \bar{y}_2 \ldots \exists \bar{y}_{t-1} \forall z_1 \in X \ldots \forall z_d \in X \, \psi \tag{3.2}$$

in case t is odd; here, $\bar{y}_1, \ldots, \bar{y}_{t-1}$ denote finite sequences of variables and ψ is a quantifier-free formula not containing X. If all $\bar{y}_i$ have length 1, $\bar{y}_i = y_i$ and if $\psi = Ry_1 \ldots y_{t-1} z_1 \ldots z_d$ for a $(t-1)+d$-ary relation symbol R, then we speak of a *generic* $\Pi_{t/d}$-formula.

The following theorem contains the characterisation of the classes $\mathbf{EW}[t]$ in terms of Fagin-definability.

Theorem 6. *Let $t \geq 2$. Then $\mathbf{EW}[t]$ is the closure of the class of problems Fagin-defined by $\Pi_{t/1}$-formulas under ept-reductions.*

More precisely, for every $\Pi_{t/1}$-formula $\varphi(X)$ the problem $p\text{-}\mathrm{FD}_{\varphi(X)}$ is contained in $\mathbf{EW}[t]$, and for every generic $\Pi_{t/1}$-formula $\varphi(X)$, the problem $p\text{-}\mathrm{FD}_{\varphi(X)}$ is complete for $\mathbf{EW}[t]$.

4 Complete Problems

In this section we show that two "non-logical" problems, the parameterized dominating set problem p-DS and the parameterized Vapnik-Chervonenkis problem p-VCDIM are complete for EW[2] and EW[3], respectively. In particular, this last result is remarkable, since in unbounded parameterized complexity theory p-VCDIM is W[1]-complete [4,5].

A dominating set in a graph $\mathcal{G} = (G, E^{\mathcal{G}})$ is a subset $S \subseteq G$, such that all vertices $a \in G$ are either in S or adjacent to some vertex in S. Now, p-DS is the following problem:

p-DS
Input: A graph $\mathcal{G}$ and $k \in \mathbb{N}$.
Parameter: k.
Problem: Decide if $\mathcal{G}$ has a dominating set of size k.

Theorem 7. *p-DS is* EW[2]*-complete.*

We turn to the parameterized Vapnik-Chervonenkis problem. Let A be a finite set and $\mathcal{S} \subseteq \wp(A)$ a family of subsets of A. We say that $\mathcal{S}$ *shatters* a set $B \subseteq A$, if

$$B \cap \mathcal{S} := \{B \cap S \mid S \in \mathcal{S}\}$$

is the powerset $\wp(B)$ of B. The *Vapnik-Chervonenkis dimension of* $(A, \mathcal{S})$, denoted by $\mathrm{VC}(A, \mathcal{S})$, is the maximum size of a set $B \subseteq A$ that is shattered by $\mathcal{S}$.

The parameterized Vapnik-Chervonenkis problem is defined as follows:

p-VCDIM
Input: A finite set A, a family $\mathcal{S}$ of subsets of A, and $k \in \mathbb{N}$.
Parameter: k.
Problem: Decide if $\mathrm{VC}(A, \mathcal{S}) \geq k$.

Theorem 8. *p-VCDIM is* EW[3]*-complete.*

5 The LOG-Classes

In this section, we establish a connection between our bounded parameterized complexity theory and classical complexity. More specifically, we will be concerned with problems that can be solved by nondeterministic polynomial time algorithms using only $O(\log^2 n)$ nondeterministic bits (in the Kintala-Fischer model of limited nondeterminism [14]).

Consider the (classical) Vapnik-Chervonenkis problem

VCDIM
Input: A finite set A, a family $\mathcal{S}$ of subsets of A, and $k \in \mathbb{N}$.
Problem: Decide if $\mathrm{VC}(A, \mathcal{S}) \geq k$.

Since the power set of a set with s elements has cardinality 2^s, the VC-dimension of $(A, \mathcal{S})$ is at most $\log n$ where $n := |\mathcal{S}|$. Hence, there is a nondeterministic algorithm for VCDIM that uses $O(\log^2 n)$ nondeterministic bits.

We have a similar complexity for many parameterized problems if we restrict them to instances with parameter $\log n$. Examples are the following problems LOG-CLIQUE and LOG-DS:

LOG-CLIQUE
Input: A graph $\mathcal{G}$.
Problem: Decide if $\mathcal{G}$ has a clique of size $\log |G|$.

LOG-DS
Input: A graph $\mathcal{G}$.
Problem: Decide if $\mathcal{G}$ has a dominating set of size $\log |G|$.

(If we write $\log n$ where an integer is expected, we mean $\lceil \log n \rceil$.)

To analyse such problems, Papadimitriou and Yannakakis [15] introduced two new, "syntactically defined", complexity classes LOGSNP and LOGNP. Syntactically defined means that they are defined via logical complete problems reminiscent of our Fagin-defined problems.

Papadimitriou and Yannakakis [15] proved that VCDIM is complete for LOGNP and LOG-DS is complete for LOGSNP, both under polynomial time reductions.

For every first-order formula $\varphi(X)$ with the monadic second-order variable X we define the "logarithmic Fagin-definable" problem $\text{LOG-FD}_{\varphi(X)}$ by

$\text{LOG-FD}_{\varphi(X)}$
Input: A structure $\mathcal{A}$.
Problem: Decide if there is a subset S of A of size $\log |A|$ with $\mathcal{A} \models \varphi(S)$.

Definition 9. For $t \geq 2$, let $\text{LOG}[t]$ be the class of problems that are polynomial time reducible to $\text{LOG-FD}_{\varphi(X)}$ for some $\Pi_{t/1}$-formula $\varphi(X)$.

Proposition 10. LOG[2] = LOGSNP *and* LOG[3] = LOGNP.

For every class Φ of propositional formulas, we let

LOG-WSAT(Φ)
Input: A formula $\alpha \in \Phi$.
Problem: Decide if α is log $|\alpha|$-satisfiable.

Theorem 11. *For every* $t \geq 2$, LOG-WSAT($\Gamma_{t,1}$) *is complete for* LOG[t] *under polynomial time reductions.*

Corollary 12. LOG[t] $\subseteq$ LOG[$t+1$] *for all* $t \geq 2$.

The last result of this section is a structural result that relates parameterized and classical complexity.

Theorem 13. *Let* $t \geq 2$. *Then,* EW[t] = EPT *if and only if* LOG[t] = PTIME.

Remark 14. The classical complexity class NP[$\log^2 n$] of all problems that can be solved by nondeterministic polynomial time algorithms using only $O(\log^2 n)$ nondeterministic bits is also related to a natural bounded parameterized complexity class: the class EW[P], the bounded analogue of the class W[P] (cf. [1,6]). It is not hard to prove that EW[P] = EPT if and only if NP[$\log^2 n$] = PTIME. Details can be found in the full version of this paper [11].

6 Higher Levels of Intractability

We mentioned in Section 2 that p-MC(WORDS, FO) is in FPT but not in EPT (assuming that FPT $\neq$ AW[$*$]). In this section we analyse the $\mathfrak{C}$-parameterized complexity of this problem.

For a class of propositional formulas Γ the *alternating weighted satisfiability problem* AWSAT(Γ) is the following parameterized problem:

AWSAT(Γ)
Input: A formula $\alpha \in \Gamma$, a partition $(\mathcal{X}_m)_{1 \leq m \leq q}$ of its variables, and a sequence $(k_m)_{1 \leq m \leq q}$ of natural numbers.
Parameter: $k_1 + \ldots + k_q$.
Problem: Decide if there is a size k_1 subset $\mathcal{Y}_1$ of $\mathcal{X}_1$ such that for every size k_2 subset $\mathcal{Y}_2$ of $\mathcal{X}_2$ there exists ... such that the truth value assignment only setting the variables in $\mathcal{Y}_1 \cup \ldots \cup \mathcal{Y}_q$ to TRUE satisfies α.

In unbounded parameterized complexity theory, the class AW[$*$] consists of all problems reducible to AWSAT($\Gamma_{t,1}$) for some $t \geq 1$. Hence, we define:

$$\text{EAW}[*] := \bigcup_{t \geq 1} [\text{AWSAT}(\Gamma_{t,1})]^{\text{ept}}.$$

As in the unbounded theory, we have:

Proposition 15. AWSAT($\Gamma_{1,2}$) *and, for* $t \geq 2$, AWSAT($\Gamma_{t,1}$) *are complete for* EAW[$*$].

We now turn to model-checking problems on words. It is well known that words over an alphabet Σ can be described by ordered finite structures over a vocabulary that has a unary relation symbol for each letter of the alphabet. The class of all structures describing finite words is denoted by WORDS.

The following theorem is remarkable, since in unbounded parameterized complexity theory the problem p-MC(WORDS, FO) is in FPT whereas p-MC(FO) is AW[*]-complete.

Theorem 16. *The following problems are complete for* EAW[*]*:*

(1) p-MC(WORDS, FO).

(2) p-MC(FO).

In the full version of the paper [11], we also give a precise characterisation of the complexity of the parameterised model-checking problem for monadic second-order logic on words.

7 The First Level of the EW-Hierarchy

What is EW[1]? Recall the definition of the classes $\mathbf{EW}[t,d]$ for $t,d \geq 1$ forming the EW-matrix. For $t \geq 2$, we defined $\mathbf{EW}[t] = \mathbf{EW}[t,1]$. Since EW[1,1] = EPT, the class EPT seems to be the natural first level of the hierarchy. However, there are natural problems, such as the parameterized clique problem

p-CLIQUE
Input: A graph $\mathcal{G}$ and $k \in \mathbb{N}$.
Parameter: k.
Problem: Decide if $\mathcal{G}$ has a clique of size k.

the complexity of which seems to be between EPT and EW[2]. In the unbounded theory, such problems conveniently fall into the class

$$\mathbf{W}[1] := \bigcup_{d\geq 1} [p\text{-WSAT}(\Gamma_{t,d})]^{\mathrm{fpt}} = [p\text{-WSAT}(\Gamma_{t,2})]^{\mathrm{fpt}}.$$

We define $\mathbf{EW}[1] := [p\text{-WSAT}(\Gamma_{1,2})]^{\mathrm{ept}}$. 5

Theorem 17. p-CLIQUE *is complete for* EW[1] *under ept-reductions.*

The proof of this result simply amounts to checking that the standard reductions proving that p-CLIQUE is complete for W[1] go through in the bounded setting.

In unbounded parameterized complexity theory, we know that p-MC(Σ_1) is complete for W[1]. It is not hard to prove that p-MC(Σ_1) is contained in EW[1], but unfortunately we are unable to prove completeness.[3] Nevertheless, in the full version of the paper [11] we give logical characterisations of the class EW[1] in terms of model-checking problems and Fagin-definability (as parts of the characterisations of the classes of the EW-matrix). 5 5

[3] On a technical level, the problem is that a Σ_1-formula stating that a graph has a clique of size k has length $\Omega(k^2)$.

Theorem 18. EW[1] = EPT *if and only if* LOG-CLIQUE $\in$ PTIME.

Theorem 13 and Theorem 18 suggest to define LOG[1] as the closure of LOG-CLIQUE under polynomial time reductions.

References

1. K.A. Abrahamson, R.G. Downey, and M.R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogs. *Annals of pure and applied logic,* 73:235–276, 1995.
2. B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science,* volume B, pages 194–242. Elsevier Science Publishers, 1990.
3. R. Downey. Parameterized complexity for the skeptic. In *Proceedings of the 18th IEEE Conference on Computational Complexity,* 2003.
4. R. G. Downey, P. A. Evans, and M. R. Fellows. Parameterized learning complexity. In *Proceedings of the 6th Annual ACM Conference on Computational Learning Theory,* pages 51–57, 1993.
5. R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness III: Some structural aspects of the W-hierarchy. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory,* pages 166–191. Cambridge University Press, 1993.
6. R.G. Downey and M.R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.
7. R.G. Downey, M.R. Fellows, and K. Regan. Descriptive complexity and the *W*-hierarchy. In P. Beame and S. Buss, editors, *Proof Complexity and Feasible Arithmetic,* volume 39 of *AMS-DIMACS Volume Series,* pages 119–134. AMS, 1998.
8. T. Eiter, G. Gottlob, and K. Makino. New results on monotone dualization and generating hypergraph transversals. In *Proceedings of the 34th ACM Symposium on Theory of Computing,* pages 14–22, 2002.
9. J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model checking. *SIAM Journal on Computing,* 31(1):113–145, 2001.
10. J. Flum and M. Grohe. Model-checking problems as a basis for parameterized intractability. To appear in *Proceedings of the 19th IEEE Symposium on Logic in Computer Science,* 2004. Full version available as Technical Report 23/2003, Fakultät für Mathematik und Physik, Albert-Ludwigs-Universität Freiburg, 2003.
11. J. Flum, M. Grohe, and M. Weyer. Bounded fixed-parameter tractability and $\log^2 n$ nondeterministic bits. Technical Report 04/2004, Fakultät für Mathematik und Physik, Albert-Ludwigs-Universität Freiburg, 2004.
12. M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proceedings of the 17th IEEE Symposium on Logic in Computer Science,* pages 215–224, 2002.
13. M. Grohe. Parameterized complexity for the database theorist. *SIGMOD Record,* 31(4):86–96, 2002.
14. C. Kintala and P. Fischer. Refining nondeterminism in relativised polynomial time bounded computations. *SIAM Journal on Computing,* 9:46–53, 1980.
15. C. H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of V-C dimension. *Journal of Computer and System Sciences,* 53:161–170, 1996.
16. M. Schaefer. Deciding the Vapnik-Červonenkis dimension is Σ_3^P-complete. *Journal of Computer and System Sciences,* 58:177–182, 1999.
17. M. Weyer. PhD thesis. In preparation.

Exact (Exponential) Algorithms for Treewidth and Minimum Fill-In

Fedor V. Fomin[1], Dieter Kratsch[2], and Ioan Todinca[3]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway,
fomin@ii.uib.no
[2] LITA, Université de Metz, 57045 Metz Cedex 01, France,
kratsch@sciences.univ-metz.fr
[3] LIFO, Université d'Orléans, 45067 Orléans Cedex 2, France,
Ioan.Todinca@lifo.univ-orleans.fr

Abstract. We show that for a graph G on n vertices its treewidth and minimum fill-in can be computed roughly in 1.9601^n time. Our result is based on a combinatorial proof that the number of minimal separators in a graph is $\mathcal{O}(n \cdot 1.7087^n)$ and that the number of potential maximal cliques is $\mathcal{O}(n^4 \cdot 1.9601^n)$.

1 Introduction

The last few years have seen an emerging interest in fast exponential algorithms for NP-hard problems. There are several explanations to this interest. Today almost everyone does expect that there is no polynomial time algorithm solving an NP-hard problem. Trying different ways of avoiding intractability several approaches were proposed such as: approximation algorithms, randomized algorithms, heuristic methods etc. Each of these approaches has weak points like necessity of exact solutions, difficulty of approximation, limited power of the method itself and many others. All these obstacles encourage us to try a direct (and perhaps desperate) way of coping with NP-hardness: Design of exponential algorithms that are significantly faster than exhaustive search. With the increased speed of modern computers, fast algorithms, even though they have exponential running times in the worst case, may actually lead to practical algorithms for certain NP-hard problems, at least for moderate instance sizes. Nice examples of fast exponential algorithms are Eppstein's graph coloring algorithm with time complexity $\mathcal{O}^*(2.4150^n)$ [14] and an $\mathcal{O}^*(1.4802^n)$ time algorithm for 3-SAT [12]. (In this paper we use a modified big-Oh notation that suppresses all other (polynomially bounded) terms. For functions f and g we write $f(n) = \mathcal{O}^*(g(n))$ if $f(n) = \mathcal{O}(g(n) \cdot poly(|n|))$, where $poly(|n|)$ is a polynomial. This modification may be justified by the exponential growth of $f(n)$.) Many natural questions in the field of exact algorithms can not be answered by the classical complexity theory, for example why some NP-hard problems can be solved significantly faster than others? For a good overview of the field see the recent survey written by Gerhard Woeginger [24].

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 568–580, 2004.

Treewidth is one of the most basic parameters in algorithms and it plays an important role in structural graph theory. It serves as one of the main tools in Robertson & Seymour's Graph Minors project [23]. It is well known that many intractable problems can be solved in polynomial (and very often in linear time) when the input is restricted to graphs of bounded treewidth. In recent years [11] it was shown that the results on graphs of bounded treewidth (branchwidth) are not only of theoretical interest but can successfully be applied to find optimal solutions or lower bounds for diverse optimization problems. (See also [5] for a comprehensive survey.) Treewidth also plays a crucial role in Downey & Fellows parameterized complexity theory [13]. An efficient solution to treewidth is the base for many applications in artificial intelligence, databases and logical-circuit design. See [1] for further references.

The minimum fill-in problem has important applications in sparse matrix computations and computational biology.

Previous results. Treewidth and minimum fill-in are known to be NP-hard even when the input is restricted to complements of bipartite graphs (so called cobipartite graphs) [2,25]. Despite of the importance of treewidth almost nothing is known on how to cope with its intractability. It is known that it can be approximated with a factor $\log OPT$ [1,9] and it is an old open question if the treewidth can be approximated with a constant factor. Treewidth is known to be fixed parameter tractable, moreover, for a fixed k, the treewidth of size k can be computed in linear time (with a huge hidden constant) [4]. There is a number of algorithms that for a given graph G and integer k, either reports that the treewidth of G is at least k, or produces a tree decomposition of width at most $c_1 \cdot k$ in time $O(c_2^k n^{O(1)})$, where c_1, c_2 are some constants (see e.g. [1]). Fixed parameter algorithms are known for fill-in problem as well [10,19].

There is no previous work on exact algorithms for treewidth or fill-in and almost nothing was known about it. Both problems can be solved in $\mathcal{O}^*(2^n)$ either by using the algorithm of Arnborg et al. [2] or by reformulating them as problems of finding a special vertex ordering and using the technique proposed by Held & Karp [18] for the travelling salesman problem.

Our results. In this paper we break the $\mathcal{O}^*(2^n)$ barrier by obtaining the first exact algorithm for treewidth of running time $\mathcal{O}^*(c^n)$ for $c < 2$. Our algorithm can be adapted not only for treewidth but for a number of other minimal triangulation problems like minimum fill-in. Our main result is the $\mathcal{O}^*(1.9601^n)$ algorithm computing the treewidth and minimum fill-in of a graph on n vertices. The algorithm can be regarded as dynamic programming across partial solutions and is based on results of Bouchitté & Todinca [7,8]. The running time analysis of the algorithm is difficult and is based on a careful counting of potential maximal cliques, i.e. vertex subsets in a graph that can be maximal cliques in some minimal triangulation of this graph. More precisely, we start by modifying the algorithm due to Bouchitté & Todinca [7] to compute the treewdith and minimum fill-in of a graph G with the given set Π_G of all potential maximal cliques of G in time $\mathcal{O}^*(|\Pi_G|)$. Then we obtain a number of structural results which provide us with the proof that $|\Pi_G| = \mathcal{O}^*(1.9601^n)$ and with an algorithm com-

puting all potential maximal cliques in time $\mathcal{O}^*(1.9601^n)$. Our analysis is based on combinatorial bounds for the number of minimal separators as well as for the number of potential maximal cliques in a graph on n vertices. Determining such bounds is an attractive combinatorial problem on its own.

For the class of AT-free graphs, for which both problems remain NP-complete, we were able to prove that the number of minimal separators and potential maximal cliques, and thus the running time of our algorithm, is $\Theta^*(2^{n/2})$ (see the full version of the paper [16]).

2 Basic Definitions

We denote by $G = (V,E)$ a finite, undirected and simple graph with $|V| = n$ vertices and $|E| = m$ edges. For any non-empty subset $W \subseteq V$, the subgraph of G induced by W is denoted by $G[W]$. For $S \subseteq V$ we often use $G \setminus S$ to denote $G[V \setminus S]$. The *neighborhood* of a vertex v is $N(v) = \{u \in V : \{u,v\} \in E\}$ and for a vertex set $S \subseteq V$ we put $N(S) = \bigcup_{v \in S} N(v) \setminus S$. A *clique* C of a graph G is a subset of V such that all the vertices of C are pairwise adjacent. By $\omega(G)$ we denote the maximum clique-size of a graph G.

The notion of treewidth is due to Robertson & Seymour [22]. A *tree decomposition* of a graph $G = (V,E)$, denoted by $TD(G)$, is a pair (X,T) in which $T = (V_T, E_T)$ is a tree and $X = \{X_i | i \in V_T\}$ is a family of subsets of V such that: (i) $\bigcup_{i \in V_T} X_i = V$; (ii) for each edge $e = \{u,v\} \in E$ there exists an $i \in V_T$ such that both u and v belong to X_i; and (iii) for all $v \in V$, the set of nodes $\{i \in V_T | v \in X_i\}$ forms a connected subtree of T.

The maximum of $|X_i| - 1$, $i \in V_T$, is called the *width* of the tree decomposition. The *treewidth* of a graph G ($\mathrm{tw}(G)$) is the minimum width over all tree decompositions of G.

A graph H is *chordal* (or *triangulated*) if every cycle of length at least four has a chord, i.e. an edge between two non-consecutive vertices of the cycle. A *triangulation* of a graph $G = (V, E)$ is a chordal graph $H = (V, E')$ such that $E \subseteq E'$. H is a *minimal triangulation* if for any intermediate set E'' with $E \subseteq E'' \subset E'$, the graph $F = (V, E'')$ is not chordal.

The following result is very useful for our algorithms.

Theorem 1 (Folklore). *For any graph G, $\mathrm{tw}(G) \leq k$ if and only if there is a triangulation H of G such that $\omega(H) \leq k+1$.*

Thus the treewidth of G can be defined as the minimum over all triangulations H of G, of $\omega(H) - 1$.

The *minimum fill-in* of a graph $G = (V,E)$, denoted by $\mathrm{mfi}(G)$, is the smallest value of $|E_H - E|$, where the minimum is taken over all triangulations $H = (V, E_H)$ of G.

In other words, computing the treewidth of G means finding a triangulation with the smallest maximum clique-size, while computing the minimum fill-in means finding a triangulation with the smallest number of edges. In both cases we can restrict our work to minimal triangulations.

Minimal separators and potential maximal cliques. Minimal separators and potential maximal cliques are the most important tools used in our proofs. Let a and b be two non adjacent vertices of a graph $G = (V, E)$. A set of vertices $S \subseteq V$ is an a,b-*separator* if a and b are in different connected components of the graph $G[V \setminus S]$. S is a *minimal* a,b-*separator* if no proper subset of 5 is an a,b-separator. We say that S is a *minimal separator* of G if there are two vertices a and b such that S is a minimal a,b-separator. Notice that a minimal separator can be strictly included in another one. We denote by Δ_G the set of all minimal separators of G. A set of vertices Ω of a graph G is called a *potential maximal clique* if there is a minimal triangulation H of G such that Ω is a maximal clique of H. We denote by Π_G the set of all potential maximal cliques of G.

The following result will be used to list all minimal separators.

Theorem 2 ([3]). *There is an algorithm listing all minimal separators of an input graph G in $\mathcal{O}(n^3|\Delta_G|)$ time.*

For a set $K \subseteq V$, a connected component C of $G \setminus K$ is a *full component associated to* K if $N(C) = K$.

The following structural characterization of potential maximal cliques is extremely useful for our purposes.

Theorem 3 ([7]). *Let $K \subseteq V$ be a set of vertices and let $\mathcal{C}(K) = \{C_1, \ldots, C_p\}$ be the set of the connected components of $G \setminus K$. Let $\mathcal{S}(K) = \{S_1, S_2, \ldots, S_p\}$ where $S_i(K)$ is the set of vertices of K adjacent to at least one vertex of $C_i(K)$. Then K is a potential maximal clique if and only if :*

1. *$G \setminus K$ has no full components associated to K, and*
2. *the graph on the vertex set K obtained from $G[K]$ by turning each $S_i \in \mathcal{S}(K)$ into a clique, is a complete graph.*

Moreover, if K is a potential maximal clique, then $\mathcal{S}(K)$ is the set of the minimal separators of G contained in K.

By Theorem 3, for every potential maximal clique Ω of G, the sets $S_i(\Omega)$ are exactly the minimal separators of G contained in K. Let us point out that for each minimal separator S_i, $\Omega \setminus S_i$ is contained in a same component of $G \setminus S_i$.

The following result is an easy consequence of Theorem 3.

Theorem 4 ([7]). *There is an algorithm that, given a graph $G = (V,E)$ and a set of vertices $K \subseteq V$, verifies if K is a potential maximal clique of G. The time complexity of the algorithm is $\mathcal{O}(nm)$.*

3 Computing Treewidth and Minimum Fill-In

We describe the algorithm of [7] that, given a graph, all its minimal separators and all its potential maximal cliques, computes the treewidth and the minimum fill-in of the graph. New observation here is that this algorithm can be implemented to run in $\mathcal{O}^*(|\Pi_G|)$ time.

For a minimal separator S and a component $C \in \mathcal{C}(S)$ of $G \setminus S$, we say that *(S, C)* is a *block* associated to S. We sometimes use the notation *(S, C)* to denote the set of vertices $S \cup C$ of the block. It is easy to notice that if $X \subseteq V$ corresponds the set of vertices of a block, then this block *(S, C)* is unique: indeed $S = N(V \setminus X)$ and $C = X \setminus S$.

A block *(S, C)* is called *full* if C is a full component associated to S. The graph $R(S,C) = G_S[S \cup C]$ obtained from $G[S \cup C]$ by completing S into a clique is called the *realization* of the block B.

Theorem 5 ([20]). *Let G be a non-complete graph. Then*

$$\mathrm{tw}(G) = \min_{S \in \Delta_G} \max_{C \in \mathcal{C}(S)} \mathrm{tw}(R(S,C))$$

$$\mathrm{mfi}(G) = \min_{S \in \Delta_G} (\mathrm{fill}(S) + \sum_{C \in \mathcal{C}(S)} \mathrm{mfi}(R(S,C)))$$

where fill(S) *is the number of non-edges of* $G[S]$.

In the equations of Theorem 5 we may take the minimum only over the inclusion-minimal separators of G. Then all the blocks in the equations are full.

We now express the treewidth and the minimum fill-in of realizations of full blocks from realizations of smaller full blocks. Let Ω be a potential maximal clique of G. We say that a block (S', C') is *associated to* Ω if C' is a component of $G \setminus \Omega$ and $S' = N(C')$.

Theorem 6 ([7]). *Let (S,C) be a full block of G. Then*

$$\mathrm{tw}(R(S,C)) = \min_{S \subset \Omega \subseteq (S,C)} \max(|\Omega| - 1, \mathrm{tw}(R(S_i, C_i)))$$

$$\mathrm{mfi}(R(S,C)) = \min_{S \subset \Omega \subseteq (S,C)} \left(\mathrm{fill}(\Omega) - \mathrm{fill}(S) + \sum \mathrm{mfi}(R(S_i, C_i))\right)$$

where the minimum is taken over all potential maximal cliques Ω such that $S \subset \Omega \subseteq (S,C)$ and (S_i, C_i) are the blocks associated to Ω in G such that $S_i \cup C_i \subset S \cup C$.

Theorem 7. *There is an algorithm that, given a graph G together with the list of its minimal separators Δ_G and the list of its potential maximal cliques Π_G, computes the treewidth and the minimum fill-in of G in $\mathcal{O}^*(|\Pi_G|)$ time. Moreover, the algorithm constructs optimal triangulations for the treewidth and the minimum fill-in.*

Proof's sketch. We provide here only some ideas, for a complete proof see [16]. W.l.o.g. we can assume that G is connected graph (otherwise we can run the algorithm for each connected component of G) First the algorithm computes all the full blocks and sorts them by their size. If there is no block, then G is a complete graph and in this case the solution is trivial. For each inclusion

minimal full block *(S, C)* (i.e. block containing no other full blocks), we assign $\mathrm{tw}(R(S,C)) = |S \cup C| - 1$ and $\mathrm{mfi}(R(S,C)) = \mathrm{fill}(S \cup C)$. Then, for each full block (S, C) in the increasing order, the treewidth and the minimum fill-in of the realization *R(S, C)* is computed by making use of Theorem 6. Finally, tw(G) and mfi(G) are obtained by Theorem 5.

The time complexity is given by the computation of the treewidth and minimum fill-in parameters for the realizations of all the full blocks, which works as follows:

```
for each full block (S, C)
    for each potential maximal clique Ω such that S ⊂ Ω ⊆ S ∪ C
        apply a polynomial time treatment to each block (S_i, C_i)
        associated to Ω s.t. S_i ∪ C_i ⊂ S ∪ C.
```

When S, C and Ω are fixed, the number of blocks associated to Ω is at most n. Indeed, each of these blocks corresponds to a component of $G \setminus \Omega$. These blocks can be computed in polynomial time. Let us show that the number of triples (S, C, Ω) with $S \subset \Omega \subseteq S \cup C$ is bounded by $n|\Pi_G|$. By Theorem 3, for a fixed potential maximal clique Ω there are at most n minimal separators $S \subset \Omega$. (The number of components in $G \setminus \Omega$, and hence the number of sets $S_i(\Omega)$ is at most n.) Also by Theorem 3, for each minimal separator $S \subset \Omega$, there is a unique component C of $G \setminus S$ such that $\Omega \subseteq S \cup C$.

It remains to turn this argument into an implementation such that the two nested loops have at most $n|\Pi_G|$ iterations. During a preprocessing step, for each potential maximal clique Ω we compute all minimal separators contained in Ω (by Theorem 3, they are the neighborhoods of the components of $G \setminus \Omega$). For each $S \subset \Omega$, we compute the unique component C of $G \setminus S$ containing $\Omega \setminus S$. For each block *(S, C)* we keep a pointer to Ω. Using appropriate data structures these preprocessing costs $\mathcal{O}(nm)$ for each potential maximal clique. In this way, each block (S, C) will keep the list of all the potential maximal cliques Ω s.t. $S \subset \Omega \subseteq S \cup C$. The number of iterations of the two loops is then the number of "good" triples (S, C, Ω), i.e. at most $n|\Pi_G|$. □

After Theorem 7, the only missing ingredient of our algorithms is listing of all the minimal separators and the potential maximal cliques of a graph in time $\mathcal{O}^*(c^n)$ for some $c < 2$. In the next two sections we discuss this issue.

4 Upper Bounding the Number of Minimal Separators

In this section we show that any graph with n vertices has $\mathcal{O}(n \cdot 1.7087^n)$ minimal separators. For the main algorithm of this paper the upper bound $\mathcal{O}^*(1.9601^n)$ would be sufficient. However, bounding the number of minimal separators in a graph is a nice combinatorial problem and we prefer to give here the best upper bound we were able to find.

Let S be a separator in a graph $G = (V,E)$. For $x \in V \setminus S$, we denote by $C_x(S)$ the component *of* $G \setminus S$ containing x. The following lemma is an exercise in [17].

Lemma 1 (Folklore). *A set S of vertices of G is a minimal a,b-separator if and only if a and b are in different full components associated to S. In particular, S is a minimal separator if and only if there are at least two distinct full components associated to S.*

Theorem 8. *For any graph G, $|\Delta_G| = \mathcal{O}(n \cdot 1.7087^n)$.*

Let us note, that by Theorem 2, Theorem 8 yields immediately that all minimal separators of a graph can be listed in time $\mathcal{O}(n^4 \cdot 1.7087^n)$.

Proof. For a constant α, $0 < \alpha < 1$, we distinguish two types of minimal separators: *small* separators, of size at most αn, and *big* separators, of size more than αn. We denote the cardinalities of these sets by `#small sep` and `#big sep`. Notice that $|\Delta_G| = \#\texttt{small sep} + \#\texttt{big sep}$.

Counting big separators. Let S be a minimal separator. By Lemma 1, there are at least two full components associated to S. Hence at least one of these full components has at most $n(1-\alpha)/2$ vertices. For every $S \in \Delta_G$ we choose one of these full components, and call it the *small* component of S, denoted by $\mathrm{s}(S)$.

By the definition of a full component, $S = N(\mathrm{s}(S))$. In particular, for distinct minimal separators S and T, we have that $\mathrm{s}(S) \neq \mathrm{s}(T)$. Therefore the number `#big sep` of big minimal separators is at most the number of small components and we conclude that `#big sep` does not exceed the number of subsets of V of cardinality at most $n(1-\alpha)/2$, i.e.

$$\#\texttt{big sep} \leq \sum_{i=1}^{\lceil n(1-\alpha)/2 \rceil} \binom{n}{i} \tag{1}$$

Counting small separators. To count small separators we use a different technique. Let S be a minimal separator, let x be a vertex of a full component $C_x(S)$ associated to S with minimum number of vertices and let $X \subset V$ be a vertex subset. We say that (x, X) is a *bad pair* associated to S if $C_x(S) \subseteq X \subseteq V \setminus S$.

Note that the connected component of $G[X]$ containing x is $C_x(S)$, by the fact that $C_x(S) \subseteq X$ and $X \cap S = \emptyset$. Let S and T be two distinct minimal separators. Consider two bad pairs (x, X) and (y, Y), associated to S and T respectively. We prove that $(x, X) \neq (y, Y)$. Suppose that $x = y$ and $X = Y$. By the observation above, we have $C_x(S) = C_y(T)$. Since $C_x(S)$ is a full component associated to S in G, we have that $S = N(C_x(S))$ and $T = N(C_y(T))$. Thus $S = T$ which is a contradiction.

By Lemma 1, there are at least two full components associated to every small separator S. For a full component $C_x(S)$ associated to S with the minimum number of vertices, $|V \setminus (S \cup C_x(S))| \geq n \cdot (1-\alpha)/2$. For any $Z \subseteq V \setminus (S \cup C_x(S))$, the pair $(x, Z \cup C_x(S))$ is a bad pair associated to S. Therefore there are at least $2^{n \cdot (1-\alpha)/2}$ distinct bad pairs associated to S. Hence the total number of bad

pairs is at least $\#\texttt{small sep} \cdot 2^{n\cdot(1-\alpha)/2}$. On the other hand, the number of bad pairs is at most $n \cdot 2^n$. We conclude that

$$\#\texttt{small sep} \leq n2^{n\cdot(1+\alpha)/2} \tag{2}$$

For $\alpha = 0.5456$, by making use of Stirling's formula (1) and (2) yield that $\#\texttt{small sep} + \#\texttt{big sep} = \mathcal{O}(n \cdot 1.7087^n)$. □

5 Final Step: Counting Potential Maximal Cliques

We prove in this section the main technical result of this paper, namely that a graph with n vertices has at most $\mathcal{O}^*(\binom{n}{2n/5})$ potential maximal cliques, that is $\mathcal{O}^*(1.9601^n)$, enumerable with the same time complexity.

Roughly speaking, the idea is to show that each potential maximal clique of a graph can be identified by a set of vertices of size at most $2n/5$. The algorithm for generating all the potential maximal cliques of a graph lists all the sets of vertices of size at most $2n/5$ and then, by applying a polynomial time procedure for each set, generates all the potential maximal cliques of the input graph.

Lemma 2. *Let Ω be a potential maximal clique of G, S be a minimal separator contained in Ω and C be the component of $G\backslash S$ intersecting Ω. Then one of the following holds:*

1. $\Omega = N(C \setminus \Omega)$;
2. *there is* $a \in \Omega \setminus S$ *such that* $\Omega = N[a]$;
3. *there is* $a \in S$ *such that* $\Omega = S \cup (N(a) \cap C)$.

Proof. Since C is a component of $G \setminus S$ and S is contained in Ω, we have that $N(C \setminus \Omega) \subseteq \Omega$. If every vertex of Ω is adjacent to a vertex of $C \setminus \Omega$, then $\Omega = N(C \setminus \Omega)$.

Suppose that there is a vertex $a \in \Omega$ having no neighbors in $C \setminus \Omega$. We consider first the case $a \in \Omega \setminus S$. We claim that in this case $\Omega = N[a]$. Because $a \in \Omega \setminus S \subseteq C$ we conclude that $N[a] \subseteq \Omega$. Thus to prove the claim we need to show that $\Omega \subseteq N[a]$. For sake of contradiction, suppose that there is $b \in \Omega$ which is not adjacent to a. By Theorem 3, every two non adjacent vertices of a potential maximal clique are contained in some minimal separator $S_i(\Omega)$. Thus both a and b should have neighbors in a component $C_i(\Omega)$ of $G \setminus \Omega$. Since $a \in \Omega \setminus S \subseteq C$, we have that $C_i(\Omega) \subseteq C \setminus \Omega$. But this contradicts our assumption that a has no neighbors in $C \setminus \Omega$.

The case $a \in S$ is similar. Suppose that $\Omega \setminus S \neq N(a) \cap C$, i.e. there is $b \in \Omega \setminus S$ non adjacent to a. Then again, a and b are contained in some minimal separator and thus should have neighbors in a component $C_i(\Omega) \subseteq C$ of $G \setminus \Omega$ which is a contradiction. □

Let Ω be a potential maximal clique of G. The triple (S, a, b) is called a *separator representation* of Ω if S is a minimal separator of G, $a \in S$, $b \in V \setminus S$ and $\Omega = S \cup (N(a) \cap C_b(S))$, where $C_b(S)$ is the component of $G \setminus S$ containing

b. Let us note that for a given triple (S, a, b), one can check in polynomial time whether (S, a, b) is the separator representation of a (unique) potential maximal clique Ω.

The number of all possible separator representations of a graph is bounded by $n^2|\Delta_G|$. Unfortunately, in the case when a potential maximal clique Ω has no separator representation, we cannot say that Ω is small (i.e. of size at most βn for some $\beta < 0.5$) or is the neighborhood of a small set. In the next subsection we introduce another type of representation, the *neighborhood representation,* that allows us to show that all the potential maximal cliques can be represented by small sets of vertices.

Counting nice potential maximal cliques. Let Ω be a potential maximal clique of a graph G and let $S \subset \Omega$ be a minimal separator of G. We say that S is an *active separator for* Ω if Ω is not a clique in the graph $G_{\mathcal{S}(\Omega)\setminus\{S\}}$, obtained from G by completing all the minimal separators contained in Ω, except S. If S is active, a pair of vertices $x, y \in S$ non adjacent in $G_{\mathcal{S}(\Omega)\setminus\{S\}}$ is called an *active pair.*

Theorem 9 ([8]). *Let Ω be a potential maximal clique of G and $S \subset \Omega$ a minimal separator, active for Ω. Let (S, C) be the block associated to S containing Ω and let $x, y \in \Omega$ be an active pair. Then $\Omega \setminus S$ is a minimal x, y-separator in $G[C \cup \{x, y\}]$.*

We say that a potential maximal clique Ω is *nice* if at least one of the minimal separators contained in Ω is active for Ω. In this subsection we shall prove that a graph with n vertices has $\mathcal{O}^*(\binom{n}{2n/5})$ nice potential maximal cliques.

Lemma 3. *Let Ω be a nice potential maximal clique, S be a minimal separator, active for Ω, $x, y \in S$ be an active pair, and C be the component of $G \setminus S$ containing $\Omega \setminus S$. There is a partition (D_x, D_y) of $C \setminus \Omega$ such that $N(D_x \cup \{x\}) \cap C = N(D_y \cup \{y\}) \cap C = \Omega \setminus S$.*

Proof. By Theorem 9, $\Omega \setminus S$ is a minimal x, y-separator in $G[C \cup \{x, y\}]$. Let C_x be the full component associated to $\Omega \setminus S$ in $G[C \cup \{x, y\}]$, containing x, $D_x = C_x \setminus \{x\}$, and $D_y = C \setminus D_x$. Notice that the full component associated to $\Omega \setminus S$ in $G[C \cup \{x, y\}]$ and containing y, is in $D_y \cup \{y\}$. Therefore D_x and D_y satisfy the above condition. □

Let us note that if one of the two sets of the partition in Lemma 3, say D_x, is empty, then for any $z \in C$, the triple (S, x, z) is a separator representation of Ω. Let C' be a component of $G \setminus \Omega$ such that $N(C') = S$ and let $c \in C'$. If D_x is not empty, we have that the triple $(X = C' \cup D_x, x, c)$ is sufficient for computing Ω. Indeed, C' is the component of $G[X]$ containing c, $D_x = X \setminus C'$, C is the component of $G - N(C')$ containing D_x, and finally $\Omega = S \cup (\Omega \setminus S) = N(C') \cup (N(D_x \cup \{x\}) \cap C)$.

The triple $(C' \cup D_x, x, c)$ can be used to represent Ω.

More formally: For a potential maximal clique Ω of G, we say that a triple (X, x, c), where $X \subseteq V$, $c \in X$, and $x \notin X$ is a *neighborhood representation* of Ω if the following hold:

1. $D_x = X \setminus C'$ is not empty, where C' is the component of $G[X]$ containing c;
2. $\Omega = N(C') \cup (N(D_x \cup \{x\}) \cap C)$, where C is the component of $G - N(C')$ containing D_x.

Let us remark that for a given triple (X, x, c), we can check in polynomial time whether (X, x, c) is a neighborhood representation of a (unique) potential maximal clique Ω.

We state now the main tool for counting the nice potential maximal cliques.

Lemma 4. *Let Ω be a nice potential maximal clique of G. Then one of the following holds:*

1. $|\Omega| \leq 2n/5$;
2. *There is a vertex a such that $\Omega = N[a]$;*
3. *Ω has a separator representation;*
4. *There is a set of vertices X such that $\Omega = N(X)$ and $|X| \leq 2n/5$;*
5. *Ω has a neighborhood representation (X, x, c) such that $|X| \leq 2n/5$.*

Proof. Let S be a minimal separator active for Ω, $x, y \in S$ be an active pair, and C be the component of $G \setminus S$ containing $\Omega \setminus S$. By Lemma 3, there is a partition (D_x, D_y) of $C \setminus \Omega$ such that $N(D_x \cup \{x\}) \cap C = N(D_y \cup \{y\}) \cap C = \Omega \setminus S$. If one of the sets D_x, D_y, say D_x, is empty, then for any $z \in C$, the triple (S, x, z) is a separator representation of Ω. Suppose that none of the first three lemma's conditions hold. Then D_x and D_y are nonempty. Let C' be a component of $G \setminus \Omega$ such that $N(C') = S$ and $c \in C'$. Since D_x and D_y are not empty, we have that $(C' \cup D_x, x, c)$ and $(C' \cup D_y, y, c)$ are neighborhood representations of Ω.

By Lemma 2, $\Omega = N(C \setminus \Omega)$. Since (D_x, D_y) form a partition of $C \setminus \Omega$, it remains to prove that at least one of the sets $C \setminus \Omega = D_x \cup D_y, C' \cup D_x$ and $C' \cup D_y$ has at most $2n/5$ vertices. Clearly D_x, D_y and C' are pairwise disjoint. Since $|D_x| + |D_y| + |C'| \leq |V(G) \setminus \Omega| \leq 3n/5$, the conclusion follows. □

It is not hard to check that, for each case of Lemma 4 there are $\mathcal{O}^*(\binom{n}{2n/5})$ potential maximal cliques of that type. More precisely:

Lemma 5. *A graph on n vertices has at most $n^2 \sum_{i=1}^{2n/5} \binom{n}{i}$ nice potential maximal cliques, enumerable in $\mathcal{O}^*(\binom{n}{2n/5})$ time.*

Counting all the potential maximal cliques. Not all the potential maximal cliques of a graph are necessarily nice (see [8] for an example). For counting and enumerating all the potential maximal cliques of a graph, we need the following theorem, used in [8] for showing that the number of potential maximal cliques of G is $\mathcal{O}^*(|\Delta_G|^2)$.

Theorem 10 ([8]). *Let Ω be a potential maximal clique of G, let a be a vertex of G and $G' = G \setminus \{a\}$. Then one of the following cases holds:*

1. *either Ω or $\Omega \setminus \{a\}$ is a potential maximal clique of G'.*

2. $\Omega = S \cup \{a\}$, *where* S *is a minimal separator of* G.
3. Ω *is nice.*

Theorem 11. *A graph* G *on* n *vertices has at most* $n^3 \sum_{i=1}^{2n/5} \binom{n}{i} = \mathcal{O}(n^4 \cdot 1.9601^n)$ *potential maximal cliques, enumerable in* $\mathcal{O}^*(1.9601^n)$ *time.*

Proof. Let $x_1, x_2, \ldots, x_n$ be the vertices of G and $G_i = G[\{x_1, \ldots, x_i\}]$, $\forall i, 1 \leq i \leq n$. Theorem 10 and Lemma 5 imply that $|\Pi_{G_i}| \leq |\Pi_{G_{i-1}}| + n|\Delta_{G_i}| + n^2 \sum_{i=1}^{2n/5} \binom{n}{i}$, for all $i, 2 \leq i < n$. By Theorem 8, $|\Pi_G| \leq n^3 \sum_{i=1}^{2n/5} \binom{n}{i}$. From the same arguments it follows that Π_G can be computed and enumerated in $\mathcal{O}^*(\binom{n}{2n/5})$ time which is approximately $\mathcal{O}^*(1.9601^n)$. □

Theorems 7 and 11 imply the main result of this paper.

Theorem 12. *For a graph* G *on* n *vertices, the treewidth and the minimum fill-in of* G *can be computed in* $\mathcal{O}^*(1.9601^n)$ *time.*

6 Open Problems

An interesting question is whether the upper bounds in Theorems 8 and 11 can be improved. Let us note that the lower bound we have for minimal separators and potential cliques is of order $3^{n/3} \sim 1.4422^n$.

Our algorithms for treewidth and fill-in can also be applied for solving other problems that can be expressed in terms of minimal triangulations like finding a tree decomposition of minimum cost [6] or computing treewidth of weighted graph. However, there are two 'width' parameters related to treewidth, namely bandwidth and pathwidth and one parameter called profile, related to minimum fill-in, that do not fit into this framework. Bandwidth can be computed in time $\mathcal{O}^*(10^n)$ [15] and reducing Feige's bounds is a challenging problem. Pathwidth (and profile) can be expressed as vertex ordering problems and thus solved in $\mathcal{O}^*(2^n)$ time by applying a dynamic programming approach similar to Held & Karp's approach [18] for TSP. Let us note that reaching time complexity $\mathcal{O}^*(c^n)$, for any constant $c < 2$ even for the Hamiltonian cycle problem is a long standing problem. So it is highly unlikely that some modification of Held & Karp's approach provide us with a better exact algorithm for pathwidth or profile. It is tempting to ask if one can reach time complexity $\mathcal{O}^*(c^n)$, for any constant $c < 2$ for these problems.

Acknowledgements. We thank the referees for pointing out that the algorithm of Arnborg et al. [2] can be used to compute treewidth and minimum fill-in in $\mathcal{O}^*(2^n)$ time. Fedor Fomin acknowledges support of Norges forskningsråd, project 160778/V30.

References

1. E. AMIR, *Efficient approximation for triangulation of minimum treewidth,* in Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (UAI-2001), San Francisco, CA, 2001, Morgan Kaufmann Publishers, pp. 7–15.
2. S. ARNBORG, D. G. CORNEIL, AND A. PROSKUROWSKI, *Complexity of finding embeddings in a k-tree,* SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 277–284.
3. A. BERRY, J.P. BORDAT, AND O. COGIS, *Generating all the minimal separators of a graph,* In *Workshop on Graph-theoretic Concepts in Computer Science (WG'99),* vol. 1665 of *Lecture Notes in Computer Science.* Springer-Verlag, 1999.
4. H. L. BODLAENDER, *A linear-time algorithm for finding tree-decompositions of small treewidth,* SIAM J. Comput., 25 (1996), pp. 1305–1317.
5. ——, *A partial k-arboretum of graphs with bounded treewidth,* Theoret. Comput. Sci., 209 (1998), pp. 1–45.
6. H. L. BODLAENDER AND F. V. FOMIN, *Tree decompositions with small cost,* Proceedings of the 8th Scandinavian Workshop on Algorithm Theory (SWAT 2002), volume 2368 of *Lecture Notes in Computer Science,* pp. 378–387, Springer-Verlag, 2002.
7. V. BOUCHITTÉ AND I. TODINCA, *Treewidth and minimum fill-in: grouping the minimal separators,* SIAM J. on Computing, 31(1):212 – 232, 2001.
8. V. BOUCHITTÉ AND I. TODINCA, *Listing all potential maximal cliques of a graph,* Theoretical Computer Science, 276(1-2):17–32, 2002.
9. V. BOUCHITTÉ, D. KRATSCH, H. MÜLLER, AND I. TODINCA, *On treewidth approximation,* Discr. Appl. Math., to appear.
10. L. CAI, *Fixed-parameter tractability of graph modification problems for hereditary properties,* Inform. Process. Lett., 58 (1996), pp. 171–176.
11. W. COOK AND P. SEYMOUR, *Tour merging via branch-decomposition,* INFORMS J. on Computing, 15 (3)(2003), pp. 233–248.
12. E. DANTSIN, A. GOERDT, E. A. HIRSCH, R. KANNAN, J. KLEINBERG, C. PAPADIMITRIOU, P. RAGHAVAN, AND U. SCHÖNING, *A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search,* Theoret. Comput. Sci., 289 (2002), pp. 69–83.
13. R. G. DOWNEY AND M. R. FELLOWS, *Parameterized Complexity,* Springer-Verlag, New York, 1999.
14. D. EPPSTEIN, *Small maximal independent sets and faster exact graph coloring,* in Algorithms and data structures (Providence, RI, 2001), vol. 2125 of Lecture Notes in Comput. Sci., Springer, Berlin, 2001, pp. 462–470.
15. U. FEIGE, *Coping with the NP-hardness of the graph bandwidth problem,* in Algorithm theory—SWAT 2000 (Bergen), vol. 1851 of Lecture Notes in Comput. Sci., Springer, Berlin, 2000, pp. 10–19.
16. F. FOMIN, D. KRATSCH AND I. TODINCA, *Exact (exponential) algorithms for treewidth and minimum fill-in,* Research Report RR-2004-09, LIFO–University of Orléans, 2004.
17. M. C. GOLUMBIC, *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.
18. M. HELD AND R. KARP, *A dynamic programming approach to sequencing problems,* J. Soc. Indust. Appl. Math., 10 (1962), pp. 196–210.
19. H. KAPLAN, R. SHAMIR, AND R. E. TARJAN, *Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs,* SIAM J. Comput., 28 (1999), pp. 1906–1922.

20. T. KLOKS, D. KRATSCH, AND J. SPINRAD, *On treewidth and minimum fill-in of asteroidal triple-free graphs,* Theoretical Computer Science, 175:309–335, 1997.
21. A. PARRA AND P. SCHEFFLER, *Characterizations and algorithmic applications of chordal graph embeddings,* Discrete Appl. Math., 79(1-3):171–188, 1997.
22. N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. II. Algorithmic aspects of tree-width,* J. Algorithms, 7 (1986), pp. 309–322.
23. ——, *Graph minors. X. Obstructions to tree-decomposition,* J. Combin. Theory Ser. B, 52 (1991), pp. 153–190.
24. G. J. WOEGINGER, *Exact algorithms for NP-hard problems: a survey,* in Combinatorial Optimization: "Eureka, you shrink", M. J. et al., ed., Springer, vol. 2570, Berlin, 2003, pp. 185–207.
25. M. YANNAKAKIS, *Computing the minimum fill-in is NP-complete,* SIAM J. Algebraic Discrete Methods, 2 (1981), pp. 77–79.

Fast Parameterized Algorithms for Graphs on Surfaces: Linear Kernel and Exponential Speed-Up*

Fedor V. Fomin[1] and Dimitrios M. Thilikos[2]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway,
fomin@ii.uib.no
[2] Departament de Llenguatges i Sistemes Informàtics,
Universitat Politècnica de Catalunya, Campus Nord – Mòdul C5,
c/Jordi Girona Salgado 1-3, E-08034, Barcelona, Spain,
sedthilk@lsi.upc.es

Abstract. Preprocessing by data reduction is a simple but powerful technique used for practically solving different network problems. A number of empirical studies shows that a set of reduction rules for solving Dominating Set problems introduced by Alber, Fellows & Niedermeier leads efficiently to optimal solutions for many realistic networks. Despite of the encouraging experiments, the only class of graphs with *proven* performance guarantee of reductions rules was the class of planar graphs. However it was conjectured in that similar reduction rules can be proved to be efficient for more general graph classes like graphs of bounded genus. In this paper we (i) prove that the same rules, applied to any graph G of genus g, reduce the k-dominating set problem to a kernel of size $O(k+g)$, i.e. *linear kernel.* This resolves a basic open question on the potential of kernel reduction for graph domination, (ii) Using such a kernel we improve the best so far algorithm for k-dominating set on graphs of genus $\leq g$ from $2^{O(g\sqrt{k}+g^2)}n^{O(1)}$ to $2^{O(\sqrt{gk}+g)}+n^{O(1)}$. (iii) Applying tools from the topological graph theory, we improve drastically the best so far combinatorial bound to the branchwidth of a graph in terms of its minimum dominating set and its genus. Our new bound provides further exponential speed-up of our algorithm for the k-dominating set and we prove that the same speed-up applies for a wide category of parameterized graph problems such as k-vertex cover, k-edge dominating set, k-vertex feedback set, k-clique transversal number and several variants of the k-dominating set problem. A consequence of our results is that the non-parameterized versions of all these problems can be solved in *subexponential time* when their inputs have sublinear genus.

Keywords: Dominating set, branch-width, Σ-embedded graphs, parameterized algorithms, subexponential algorithms

* Part of this research has been initiated during the Dagstuhl Seminar no 01311 on Parameterized Complexity. The first author is supported by Norges forskningsråd project 160778/V30. The second author is supported by the EU within the 6th Framework Program under contract 001907 (DELIS) and by the Spanish CICYT project TIC-2002-04498-C05-03 (TRACER).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 581–592, 2004.

1 Introduction

The theory of fixed-parameter algorithms and parameterized complexity has been thoroughly developed over the past few years [9]. Dominating Set is one of the basic problems in parameterized complexity belonging to the complexity class W[2] and it is not surprising that it was investigated intensively. In the last three years, there was a breakthrough in understanding the parameterized complexity of Dominating Set on planar graph and different generalizations. The first fixed-parameter algorithm for k-dominating set in planar graphs [9] has running time $O(11^k n)$; subsequently, the first subexponential parameterized algorithm with running time $O(4^{6\sqrt{34k}} n)$ have been obtained by Alber et al. [1]. The development in the area of subexponential parameterized algorithms has proceeded in several directions:

Direction (i): *Reduction to linear kernel.* Let $\mathcal{L}$ be a parameterized problem, i.e. $\mathcal{L}$ consists of pairs (I, k) where k is the *parameter* of the problem. *Reduction to linear problem kernel* is the replacement of problem inputs (I, k) by a reduced problem with inputs (I', k') (linear kernel) with constants c_1, c_2 such that $k' \leq c_1 k$, $|I'| \leq c_2 k'$ and $(I, k) \in \mathcal{L} \Leftrightarrow (I', k') \in \mathcal{L}$. The existence of linear kernel for Dominating Set is highly unlikely because existing of such a kernel would imply the collapsing of the W-hierarchy. For planar graphs Alber, Fellows & Niedermeier [2] proved that Dominating Set has a linear kernel. This kernel is obtained by repetitively applying on the input graph G a set of *reduction rules.* We call this reduction *AFN-reduction.* It was also conjectured in [2] that the AFN-reduction provide linear kernels not only for class of planar graphs but for more general classes, like graphs of bounded genus. This was one of the biggest remaining challenges in the field.

Direction (ii): *The generality of graph classes to which these algorithms apply.* Ellis, Fan & Fellows [10] claimed that Dominating Set is fixed parameter tractable for graphs of bounded genus. Demaine et al. [5] recently proved this result by obtaining a subexponential parameterized algorithm that requires $2^{O(g\sqrt{k}+g^2)} n^{O(1)}$ steps on graphs of genus g. Subexponential parameterized algorithms are also known for graphs excluding a fixed graph as a minor [5], map graphs [4] and graphs of bounded local treewidth [6].

Direction (iii): *Optimization of the constants in the exponents of the running time.* The running time of Alber et al. algorithm [1] was improved to an algorithm of $O(2^{27\sqrt{k}} n)$-time by Kanj & Perković in [13], and finally to the $O(2^{15.13\sqrt{k}} k + n^3 + k^4)$-time algorithm of [11].

Direction (iv): *Extensions to other parameters.* In [1,7,14] it was observed that dominating set number is related to several graph parameters in a way that implies the existence of subexponential parameterized algorithms for all of them. This observation has been generalized in [5] to the general family of the *bidimensional* parameters. Examples of such parameters are: vertex cover, r-domination, edge-dominating set, weighted vertex dominating set, feedback set, maximal marching, clique transversal number, perfect code, and total dominating set.

Our contribution. Our results span all the research directions that we just mentioned. We enumerate them in the same order:
(i) We answer affirmatively the conjecture of [2]. More precisely, we prove that the application of the AFN-reduction on any graph G reduce it to a graph G' of size $O(k+g)$ where k and g are the dominating set number and the Euler genus of G respectively (see Section 3).
(ii) The kernel existence implies combinatorial bounds that are able to improve the best so far $2^{O(g\sqrt{k}+g^2)}n^{O(1)}$-time algorithm given in [5] to one of $2^{O(\sqrt{kg}+g)}+n^{O(1)}$-time (see Section 4).
(iii) All our algorithms have small hidden constants in the "O"-notation of their exponential part. We stress that this is not a straightforward consequence of the kernel existence and for this we need to prove better combinatorial bounds using elements of the Graph Minor Theory (see Section 5).
(iv) Using the above combinatorial bounds we can design $2^{O(\sqrt{kg}+g)}n^{O(1)}$-time algorithms for the majority of the parameters examined in direction (iii) (see Section 6).

The main graph theoretic tool of this paper is the representativity of a graph embedded in a surface Σ that is the minimum number of vertices met by an edge-avoiding non-contractible cycle of Σ. Very roughly, we implement the following "trick" several times: For graphs of representativity more than 6 we prove that they are enough "locally planar" and that certain arguments about planar graphs can be extended to graphs that are embedded that way on a surface. If representativity is at most 6, we can "cut" the surface, "split" the graph, decrease its genus, and apply certain inductive arguments.

We note that the contribution of the genus in the time complexity of our algorithms has some more general consequences. The first, is that the k-dominating set problem can be solved by a subexponential parameterized algorithm when restricted to graphs of genus $o(\log n)$. The second is that the algorithm remains subexponential on k even when $g=o(k)$. Therefore, for graphs with genus $o(n)$ the dominating set problem admits a subexponential *exact* algorithm. The same holds for a number of other problems discussed in Section 6.

2 Preliminaries

We denote by G a finite, undirected and simple graph with $|V(G)|=n$ vertices and $|E(G)|=m$ edges. For any non-empty subset $W\subseteq V(G)$, the subgraph of G induced by W is denoted by $G[W]$. The *neighbourhood* of a vertex v is $N(v)=\{u\in V(G):\ \{u,v\}\in E(G)\}$ and for a vertex set $S\subseteq V(G)$ we put $N[S]=\bigcup_{v\in S}N[v]$ and $N(S)=N[S]\setminus S$.

A set $D\subseteq V(G)$ is a *dominating set* in a graph G if every vertex in $V(G)\setminus D$ is adjacent to a vertex in D. Graph G is *D-dominated* if D is a dominating set in G. We denote by $\gamma(G)$ the minimum size of dominating set in G.

Given an edge $e=\{x,y\}$ of a graph G, the graph G/e is obtained from G by contracting the edge e; that is, to get G/e we identify the vertices x and y and remove all loops and duplicate edges. A graph H obtained by a sequence of

edge-contractions is said to be a *contraction* of G. H is a *minor* of G if H is the subgraph of a contraction of G.

Graphs on surfaces. A *surface* Σ is a compact 2-manifold without boundary. We will always consider connected surfaces. We denote by $\mathbb{S}_0$ the sphere $(x, y, z \mid x^2 + y^2 + z^2 = 1)$. A *line* in Σ is subset homeomorphic to [0, 1]. An O-arc is a subset of Σ homeomorphic to a circle. Let G be a graph 2-cell embedded in Σ. To simplify notations we do not distinguish between a vertex of G and the point of Σ used in the drawing to represent the vertex or between an edge and the line representing it. We also consider G as the union of the points corresponding to its vertices and edges. That way, a subgraph H of G can be seen as a graph H where $H \subseteq G$. We call by *region* of G any connected component of $\Sigma - E(G) - V(G)$. (Every region is an open set.) We use the notation $V(G)$ and $E(G)$, for the set of the vertices and edges of G. For $\Delta \subseteq \Sigma$, $\overline{\Delta}$ is the *closure* of Δ. The boundary of Δ is $\mathbf{bor}(\Delta) = \overline{\Delta} \cap \overline{\Sigma - \Delta}$.

A subset of Σ meeting the drawing only in vertices of G is called *G-normal*. If an O-arc is G-normal then we call it *noose*. The length of a noose is the number of its vertices. For a D-dominated Σ-embedded graph G we define a D-noose on G as a noose meeting exactly two vertices x, y of D, two neighbors of x and two neighbors of y. A D-noose N is *consecutive* is any two vertices of G that are met consecutively in N are adjacent.

Representativity [16] is the measure how dense a graph is embedded on a surface. The *representativity* (or *face-width*) $\mathbf{rep}(G)$ of a graph G embedded in surface $\Sigma \neq \mathbb{S}_0$ is the smallest length of a non-contractible noose in Σ. In other words, $\mathbf{rep}(G)$ is the smallest number k such that Σ contains a non-contractible (non null-homotopic in Σ) closed curve that intersects G in k points.

It is more convenient to work with Euler genus. The *Euler genus* $\mathbf{eg}(\Sigma)$ of a surface Σ is equal to the non-orientable genus $\tilde{g}(\Sigma)$ (or the crosscap number) if Σ is a non-orientable surface. If Σ is an orientable surface, $\mathbf{eg}(\Sigma)$ is $2g(\Sigma)$, where $g(\Sigma)$ is the orientable genus of Σ. Given a graph G its Euler genus $\mathbf{eg}(G)$ is the minimum $\mathbf{eg}(\Sigma)$ where Σ is a surface where G can be embedded.

Let N be a noose in a Σ-embedded graph G. We need to define *cutting along* the noose N. The formal definition can be found in [15], here we prefer to give a more intuitive one. We suppose that G is embedded in Σ such that for any $v \in N \cap V(G)$, there exists an open disk Δ containing v and such that for every edge e adjacent to v, $e \cap \Delta$ is connected. We also assume that $\Delta - N$ has two connected components Δ_1 and Δ_2. Thus we can define partition of $N(v) = N_1(v) \cup N_2(v)$, where $N_1(v) = \{u \in N(v) \colon \{u, v\} \cap \Delta_1 \neq \emptyset\}$ and $N_2(v) = \{u \in N(v) \colon \{u, v\} \cap \Delta_2 \neq \emptyset\}$. Now for each $v \in N \cap V(G)$ we do the following: (a) remove v and its incident edges (b) introduce two new vertices v^1, v^2 and (c) connect v^i with the vertices in $N_i, i = 1, 2$. The resulting graph is obtained from Σ-embedded graph G by cutting along N.

The following lemma is very useful in proofs by induction on the genus. The first part of the lemma follows from Proposition 4.2.1 (corresponding to surface separating cycle) and the second part follows from Lemma 4.2.4 (corresponding to non-separating cycle) in [15].

Lemma 1. *Let G be a Σ-embedded graph and let G' be a graph obtained from G by cutting along a non-contractible noose N on G. Then one of the following holds*

- *G' is the disjoint union of graphs G_1 and G_2 that can be embedded in surfaces Σ_1 and Σ_2 such that $\mathbf{eg}(\Sigma) = \mathbf{eg}(\Sigma_1) + \mathbf{eg}(\Sigma_2)$ and $\mathbf{eg}(\Sigma_i) > 0$, $i = 1, 2$.*
- *G' can be embedded in a surface with Euler genus strictly smaller than $\mathbf{eg}(\Sigma)$.*

Branch-width. A *branch decomposition* of a graph (or a hyper-graph) G is a pair (T, τ), where T is a tree with vertices of degree 1 or 3 and τ is a bijection from the set of leaves of T to $E(G)$. For a subset of edges $X \subseteq E(G)$ let $\delta_G(X)$ be the set of all vertices incident to edges in X and $E(G) \setminus X$. For each edge e of T, let $T_1(e)$ and $T_2(e)$ be the sets of leaves in two components of $T \setminus e$. The *order* of an edge e in T is $|\bigcup_{v \in T_1(e)} \delta_G(\tau(v))|$. In other words, the order of e is the number of vertices $v \in V(G)$ such that there are leaves t_1, t_2 in T in different components of $T(V(T), E(T) \setminus e)$ with $\tau(t_1)$ and $\tau(t_2)$ both containing v as an endpoint. The *width* of (T, τ) is the maximum order over all edges of T, and the *branch-width* of G, $\mathbf{bw}(G)$, is the minimum width over all branch decompositions of G.

The following relation was obtained in [11].

Theorem 1 ([11]). *For any planar D-dominated graph G, $\mathbf{bw}(G) \leq 3\sqrt{4.5}\sqrt{|D|}$.*

The following lemma that is based on Theorem 1 and the results of Djidjev & Venkatesan on planarizing sets in [8].

Lemma 2. *For any Σ-embedded graph G on n vertices, $\mathbf{bw}(G) \leq (\sqrt{4.5} + 2\sqrt{2 \cdot \mathbf{eg}(\Sigma)})\sqrt{n}$.*

3 Kernelization

Alber et al. [2] introduce reduction rules for the dominating set problem. Let us call these rules *AFN-reduction*. AFN-reduction can be applied to any graph G and the domination number of the reduced graph is equal to the domination number of G. As it was proved in [2], when G is planar, the reduced graph has at most $335\gamma(G)$ vertices, i.e. is a *linear* kernel. Also it was conjectured in [2] that AFN-reduction produces a kernel for graphs embedded in a surface of bounded genus. In this section we give an affirmative answer to this conjecture by proving that for any Σ-embedded graph G the size of the reduced graph is $O(\gamma(G) + \mathbf{eg}(\Sigma))$.

In fact, the rules of AFN-reduction are not important for our proofs. The only facts we need are the properties of the reduced graph proved in [2] and due to space restriction we move the description of the rules to Appendix. We also call a graph *reduced* if none of these rules can be applied to it.

The rules are based on a partition of the open neighbourhood of vertices or pair of vertices into three categories of sets.

For every vertex $v \in V(G)$ we partition $N(v)$ into:
- $N_{\text{exit}} = \{u \in N(v) \mid N(u) - N[v] \neq \emptyset\}$
- $N_{\text{guard}} = \{u \in N(v) - N_{\text{exit}} \mid N(u) \cap N_{\text{exit}}(v) \neq \emptyset\}$
- $N_{\text{prison}} = N(v) - (N_{\text{exit}}(v) \cup N_{\text{guard}}(v))$

For every pair v, w we partition $N(v,w) = N(v) \cup N(w)$ into:
- $N_{\text{exit}}(v,w) = \{u \in N(v,w) \mid N(u) - N[v,w] \neq \emptyset\}$
- $N_{\text{guard}}(v,w) = \{u \in N(v,w) - N_{\text{exit}}(v,w) \mid N(u) \cap N_{\text{exit}}(v,w) \neq \emptyset\}$
- $N_{\text{prison}}(v,w) = N(v,w) - (N_{\text{exit}}(v,w) \cup N_{\text{guard}}(v,w))$.

Lemma 3. *Let G be an n-vertex Σ-embedded graph. Then AFN-reduction can be performed in $O(n^2 \cdot \mathbf{eg}(\Sigma))$ steps.*

The main result of [2] it it the following:

Theorem 2. *For any reduced planar graph G, $|V(G)| \leq 335 \cdot \gamma(G)$.*

The next proof is a generalization of Theorem 2 for graphs embedded in arbitrary surfaces of representativity at least 6. Such graphs are "locally planar" in whatever the AFN-reduction is concerned. In particular, the machinery of the reduction and the proof of its correctness in [2] are applied on planar discs with boundary of length ≤ 6. This gives an opportunity to reproduce the arguments from [2] for Σ-embedded graphs of representativity at least 6.

Theorem 3. *Let G be a reduced Σ-embedded graph where* $\mathbf{rep}(G) > 6$. *Then* $|V(G)| \leq 335 \cdot \gamma(G) + 333 \cdot \mathbf{eg}(\Sigma)$.

Let G be a Σ-embedded graph. For a noose N in Σ we define the graph G_N as follows. First we take the graph G' obtained from G after cutting along N. Then for every $v \in N \cap V(G)$ if v^i, $i = 1, 2$, is not adjacent to a vertex u which is pendant in G, we add to G' a pendant vertex u^i adjacent to v^i. Thus in G_N each new vertex obtained from splitting of vertices $N \cap V(G)$ is adjacent to exactly one pendant vertices. Clearly, G_N has the same genus as G'. Since every dominating set D in G can be turned into dominating set of G_N by adding all new vertices to D, we have that $\gamma(G_N) \leq \gamma(G) + 2|N \cap V(G)|$.

According to [2], a graph G is reducible iff it satisfies the following properties:
(i) For every $v \in V(G)$, the set $N_{\text{prison}}(v)$ is empty with only one exception: $N_{\text{prison}}(v)$ can contain one "gadget" pendant vertex.
(ii) For all $v, w \in V(G)$ there exist a single vertex $v \in N_{\text{guard}}(v,w) \cup N_{\text{prison}}(v,w)$ where $N_{\text{prison}}(v,w) \subseteq N[v]$ (i.e. v dominates all vertices in $N_{\text{prison}}(v,w)$).

By construction, every vertex v^i, $i = 1, 2$, $v \in N \cap V(G)$, is not a prison vertex (it is adjacent to pendant vertex) and every vertex vertex has no more than one pendant neighbor. So we conclude that if G is reducible then G_N is also reducible.

Theorem 4. *For any reduced Σ-embedded graph G, $|V(G)| \leq 335(\gamma(G) + 24 \cdot \mathbf{eg}(\Sigma))$.*

Proof. If $\Sigma = \mathcal{S}_0$, the result follows from Theorem 2. Suppose then that $\mathbf{eg}(G) > 0$. We prove a stronger inequality: $|V(G)| \leq 335(\gamma(G) + 24\mathbf{eg}(\Sigma) - 12)$

by induction on $\mathbf{eg}(\Sigma)$. For $\mathbf{eg}(\Sigma) = 1$ and $\mathbf{rep}(G) > 6$ the result follows from Theorem 3. For $\mathbf{eg}(\Sigma) = 1$ and $\mathbf{rep}(G) \leq 6$, Lemma 1 implies that the graph G' obtained from G by cutting along N is planar, and hence the graph G_N is also planar. By Theorem 2 $|V(G_N)| \leq 335 \cdot \gamma(G_N)$ and thus (the length of N is at most 6), $|V(G)| \leq |V(G_N)| \leq 335 \cdot \gamma(G_N) \leq 335 \cdot (\gamma(G) + 12)$.

Assume now that $|V(G)| \leq 335(\gamma(G) + 24 \cdot \mathbf{eg}(\Sigma) - 12)$ for any Σ-embedded graph where $1 \leq \mathbf{eg}(\Sigma) < g$ and let G be a Σ-embedded graph where $\mathbf{eg}(\Sigma) = g \geq 2$. Again by Theorem 3, it is enough to examine the case where $\mathbf{rep}(G) \leq 6$. Let N be a non-contractible noose of minimum length in Σ. Then the length of N is at most 6.

By Lemma 1, either G_N is the disjoint union of graphs G_1 and G_2 that can be embedded in surfaces Σ_1 and Σ_2 such that $\mathbf{eg}(\Sigma) = \mathbf{eg}(\Sigma_1) + \mathbf{eg}(\Sigma_2)$ and $\mathbf{eg}(\Sigma_i) > 0$, $i = 1, 2$ (this is the case when N is surface separating curve), or G_N can be embedded in a surface with Euler genus strictly smaller than $\mathbf{eg}(\Sigma)$ (this holds when N is not surface separating).

Let us consider first the case when G_N is the disjoint union of graphs G_1 and G_2 that can be embedded in surfaces Σ_1 and Σ_2. As we discussed above, G_N is a reduced graph and thus G_1 and G_2 are also reduced graphs. The conditions $\mathbf{eg}(\Sigma) = \mathbf{eg}(\Sigma_1) + \mathbf{eg}(\Sigma_2)$ and $\mathbf{eg}(\Sigma_i) > 0$, $i = 1, 2$, imply that $1 \leq \mathbf{eg}(\Sigma_i) \leq \mathbf{eg}(\Sigma) - 1 < g$. Therefore we can apply the induction hypothesis on G_i and get that $|V(G_i)| \leq 335(\gamma(G_i) + 24 \cdot \mathbf{eg}(\Sigma_i) - 12), i = 1, 2$. Thus $|V(G)| \leq |V(G_N)| = |V(G_1)| + |V(G_2)| \leq 335(\gamma(G_1) + 24 \cdot \mathbf{eg}(\Sigma_1) - 12) + 335(\gamma(G_2) + 24 \cdot \mathbf{eg}(\Sigma_2) - 12) = 335(\gamma(G_1) + \gamma(G_2) + 24 \cdot \mathbf{eg}(\Sigma_1) + 24 \cdot \mathbf{eg}(\Sigma_2) - 24) = 335(\gamma(G') + 24 \cdot (\mathbf{eg}(\Sigma'_1) + \mathbf{eg}(\Sigma'_2)) - 24) \leq 335(\gamma(G) + 12 + 24 \cdot \mathbf{eg}(\Sigma) - 24) = 335(\gamma(G) + 24 \cdot \mathbf{eg}(\Sigma) - 12)$. For the second case, when G_N can be embedded in a surface Σ' with Euler genus strictly smaller than $\mathbf{eg}(\Sigma)$, we have that $1 \leq \mathbf{eg}(\Sigma') \leq \mathbf{eg}(\Sigma) - 1 < g$ and therefore we can apply the induction hypothesis on G_N. Thus $|V(G)| \leq |V(G_N)| \leq 335(\gamma(G_N) + 24 \cdot \mathbf{eg}(\Sigma') - 12) \leq 335(\gamma(G) + 12 + 24 \cdot (\mathbf{eg}(\Sigma) - 1) - 12) \leq 335(\gamma(G) + 24 \cdot \mathbf{eg}(\Sigma) - 24) < 335(\gamma(G) + 24 \cdot \mathbf{eg}(\Sigma) - 12)$ □

Lemma 3 and Theorem 4 imply the main result of this section.

Theorem 5. *Let G be a graph that can be embedded in Σ. AFN-reduction constructs in $O(n^3 \cdot \mathbf{eg}(\Sigma))$ steps a graph G' of size $\leq 335(\gamma(G) + 24 \cdot \mathbf{eg}(\Sigma))$ such that $\gamma(G) = \gamma(G')$.*

4 Direct Consequences of the Kernel Construction

As far as we have kernel reduction we can improve the algorithms given in [5, 11,12] for the dominating set problem. The key observation is that after the AFN-reduction, the size of the remaining kernel depends only on the genus and the minimum dominating set of the initial graph and, because of Lemma 2, the same will hold for its branchwidth as well.

Theorem 6. *For a given graph G and constants k, g, there is an $2^{O(\sqrt{kg}+g)} \mathsf{poly}(k, g) + O(n^3)$ algorithm that either computes a dominating set in G of size $\leq k$, or concludes that at least one of the following holds: (a) $\gamma(G) > k$, (b) G can not be embedded in a surface of Euler genus g.*

Theorem 6 improves asymptotically the algorithm for dominating set in [5] that requires $2^{O(g\sqrt{k}+g^2)}n^{O(1)}$ steps. However, we should admit that the hidden constants in the big-O notation are quite big. Even using the smallest factor approximation algorithm of [3], for $k = 1$ and $\mathbf{eg}(\Sigma) = 1$ the algorithm requires more than 2^{200} steps, which makes this result interesting only from theoretical point of view. In the next section we explain how the combinatorial bound to the branchwidth of G' in step 3 can be improved. Such an improvement immediately accelerates steps 2 and 3 that dominate the exponential part of the running time of the algorithm.

5 Better Combinatorial Bounds – Faster Algorithms

We call a D-dominated graph G *uniquely dominated* if there is no path of length < 3 connecting two vertices of D. Notice that this implies that each vertex $x \in V(G) \setminus D$ has exactly one neighbor in D (i.e. is uniquely dominated). The proof of the following normalization lemma is omitted because of lack of space.

Lemma 4. *For every D-dominated Σ-embedded graph G without multiple edges, there exists a Σ-embedded graph H such that (a) G is a minor of H, (b) H is uniquely D-dominated, (c) If $x, y \in D$ have distance 3 in H then there exist at least two internally disjoint (x,y)-paths in H, and (d) Any D-noose of Σ is consecutive.*

Let G be a connected D-dominated Σ-embedded graph satisfying properties (b) – (d) of Lemma 4. We call such graphs *nicely* D-dominated Σ-embedded graphs.

Let G be a nicely D-dominated Σ-embedded graph. We say that a cycle of length 6 is a D-cycle if it contains exactly two vertices from D. If $\mathbf{rep}(G) > 6$, every D-cycle C is contractible and thus one of the components of $\Sigma \setminus C$ is homeomorphic to $\{(x,y)\colon x^2 + y^2 \leq 1\}$. We denote such a disk by $\mathrm{disk}(C)$. Clearly, $G \cap \mathbf{disk}(C)$ is a planar graph.

A D-cycle C of a nicely D-dominated Σ-embedded graph G is *maximal* if there is no D-cycle of G where $\mathbf{disk}(C) \subset \mathbf{disk}(C')$. We denote as $\mathcal{C}(G)$ the set of all the maximal cycles of G.

For a nicely D-dominated Σ-embedded graph G and the set $\mathcal{C}(G)$ of all maximal D-cycles of G, we define hypergraph $\mathcal{H}(G) = (V(G), E(G) \cup \{V(C) \mid C \in \mathcal{C}(G)\})$, i.e. $\mathcal{H}(G)$ is obtained from G by adding hyperedges corresponding to maximal D-cycles of G. Clearly, $\mathbf{bw}(G) \leq \mathbf{bw}(\mathcal{H}(G))$.

If representativity of G is more than 6, for every D-maximal cycle C (which is of length 6), the hypergraph $\mathcal{H}(C) = \mathcal{H}(G) \cap \mathbf{disk}(C)$ is a hypergraph that can be obtained from a planar graph $H(C)$ by adding one hyperedge of cardinality 6. Since the planar graph H is D'-dominated for some $D' \subseteq D$, we have that by Theorem 1, $\mathbf{bw}(\mathcal{H}(C)) \leq 3\sqrt{4.5}\sqrt{|D|} + 6$.

We also define a hypergraph $\mathcal{S}(G)$ as the hypergraph obtained by removing from $\mathcal{H}(G)$ all edges of graphs $G \cap \mathbf{disk}(C)$, $C \in \mathcal{C}(G)$. Using properties (c) and

(d) one can prove that the hyperedges of $\mathcal{S}(G)$ are exactly the maximal D-cycles of G (all edges of G will be removed).

We need the following technical Lemma from [11]

Lemma 5. *If $\mathcal{G}_1$ and $\mathcal{G}_2$ are hypergraphs where $V(\mathcal{G}_1) \cap V(\mathcal{G}_1) = f$ and $\{f\} = E(\mathcal{G}_1) \cap E(\mathcal{G}_2)$, then* $\mathbf{bw}(\mathcal{G}_1 \cup \mathcal{G}_2) \leq \max\{\mathbf{bw}(\mathcal{G}_1), \mathbf{bw}(\mathcal{G}_1), |f|\}$.

For every $C \in \mathcal{C}(G)$ hypergraphs $\mathcal{H}(C)$ and $\mathcal{S}(G)$ have only hyperedge C in common and Theorem 1 and Lemma 5 imply the following result.

Lemma 6. *Let G be a nicely D-dominated Σ-embedded graph of representativity > 6. Then* $\mathbf{bw}(G) \leq \mathbf{bw}(\mathcal{H}(G)) \leq \max\{3\sqrt{4.5}\sqrt{|D|} + 6, \mathbf{bw}(\mathcal{S}(G))\}$.

Thus to obtain the upper bound for branch-width of nicely dominated graphs we need to estimate the branch-width of $\mathcal{S}(G)$.

Lemma 7. *Let G be a nicely D-dominated Σ-embedded graph of representativity > 6. Then* $\mathbf{bw}(\mathcal{S}(G)) \leq 3(\sqrt{4.5} + 2\sqrt{2 \cdot \mathbf{eg}(\Sigma)})\sqrt{|D|}$.

Proof (Sketch). Let us show first that for any two distinct maximal cycles $C_1, C_2 \in \mathcal{C}(G)$ **(i):** For *each* $u \in V(C_1) \cap V(C_2)$, $u \in N_G[v]$ *for some* $v \in D$.

In other words, for any two distinct maximal cycles $C_1, C_2 \in \mathcal{C}(G)$ the set $C_1 \cap C_2$ is either empty, or a vertex of D, or a set of vertices adjacent to one vertex of D. In fact, if $(V(C_1) \cap V(C_2)) \cap D = \emptyset$ then every vertex $u \in V(C_1) \cap V(C_2)$ is not uniquely dominated. If $|V((C_1) \cap V(C_2)) \cap D| = 2$ then cycles are not maximal. If $(V(C_1) \cap V(C_2)) \cap D = v$, we again have that every vertex $u \in (V(C_1) \cap V(C_2)) \setminus N_G[v]$ is not uniquely dominated. In all three cases we obtain a contradiction either to the definition of maximal cycle, or to the property (b) of nicely D-dominated graphs.

To estimate the value of $\mathbf{bw}(\mathcal{S}(G))$ we need the following notion. Let D' be the set of vertices of D that are also vertices of some maximal cycles, i.e. $D' = D \cap \bigcup_{C \in \mathcal{C}(G)} V(C)$. For a nicely D-dominated Σ-embedded graph G and the set of its maximal D-cycles $\mathcal{C}$ we define *concise* graph, $\mathbf{con}(G)$, as the graph with vertex set D' and where two vertices $x, y \in D'$ are adjacent in $\mathbf{con}(G)$ if and only if the distance x and y in G is 3. There is a natural bijection π correspondence between hyperedges of $\mathcal{S}(G)$ and $\mathbf{con}(G)$. Every cycle $C \in \mathcal{C}(G)$ (which is edge in $\mathcal{S}(G)$) π maps to an edge of $\mathbf{con}(G)$ with endpoints $D \cap V(C)$. By property (c) of nicely dominated graphs, π is surjection. Because cycles in $\mathcal{C}$ are maximal, π is injection.

By making use of **(i)** one can prove that $\mathbf{con}(G)$ is also Σ-embedded graph. Then by Lemma 2, $\mathbf{bw}(\mathbf{con}(G)) \leq (\sqrt{4.5} + 2\sqrt{2 \cdot \mathbf{eg}(\Sigma)})\sqrt{|D|}$ **(ii)** which implies the lemma if $\mathbf{bw}(\mathcal{S}(\mathcal{C})) \leq 3 \cdot \mathbf{bw}(\mathbf{con}(G))$ **(iii).**

Let us prove **(iii)** first for the case when the maximum vertex degree in $\mathbf{con}(G)$ is at most 3. Let A, B be a partition of $\mathcal{C}(G)$. We claim that $|\delta_{\mathcal{S}(\mathcal{C})}(A)| \leq 3|\delta_{\mathbf{con}(G)}(\pi(A))|$ **(iv).** Let $v \in D$. By **(i),** every $u \in N_G(v)$ is contained in at most two hyperedges of $\mathcal{S}(G)$ and both these edges contain v. Also for every vertex $u \in N_G(v)$, $u \in \delta_{\mathcal{S}(G)}(A)$ if and only if $u \in V(C_1) \cap V(C_2)$ for some $C_1 \in A$ and $C_2 \in B$. The degree of v in $\mathbf{con}(G)$ is ≤ 3. Thus v is contained in

at most three maximal cycles and therefore at most two neighbors of v in G can be in $\delta_{\mathcal{S}(G)}(A)$. Hence For each $v \in \delta_{\mathbf{con}(G)}(\pi(A))$, $|N_G[v] \cap \delta_{\mathcal{S}(G)}(A)| \leq 3$ **(v)**. Now **(iv)** follows from **(i)** and **(v)**. Finally, **(iv)** implies **(iii)** when the maximum vertex degree of $\mathbf{con}(G)$ is at most 3.

To prove **(iii)** in general case we need the following deep result following from Theorem (4.3) of [17] and (6.6) of [18]: for *any* Σ-embedded *graph* G *of branch-width* ≥ 2, *the branch-width of* G *is equal to the branch-width of its dual.*

A Σ-embedded graph G is *multiply triangulated* if all its regions are of length 2 or 3. A graph is (2, 3)-regular if all its vertices have degree 2 or 3. Notice that the dual of a multiply triangulated graph is (2, 3)-regular and vice versa. The proof of the following claim is similar to the proof for planar graphs (Lemma 3.3 in [11]) and we omit it here. *Every 2-connected* Σ-embedded *graph* G *has a weak triangulation* H *such that* $\mathbf{bw}(H) = \mathbf{bw}(G)$.

We claim now that every *2-connected* Σ-embedded *graph* G *is the contraction of a (2,3)-regular* Σ-embedded *graph* H *such that* $\mathbf{bw}(H) = \mathbf{bw}(G)$. In fact, let G^d be the dual graph of G. By Robertson & Seymour theorem, $\mathbf{bw}(G^d) = \mathbf{bw}(G)$. There is a weak triangulation H^d of G^d such that $\mathbf{bw}(H^d) = \mathbf{bw}(G^d)$. The dual of H^d, we denote it by H, contains G as a contraction (each edge removal in a Σ-embedded graph corresponds to an edge contraction in its dual and vice versa). Applying Robertson & Seymour the second time, we obtain that $\mathbf{bw}(H) = \mathbf{bw}(H^d)$. Hence, $\mathbf{bw}(H) = \mathbf{bw}(G)$. Since H^d is multiply triangulated, we have that H is (2, 3)-regular.

Suppose that now that $\mathbf{con}(G)$ is 2-connected. For $\mathbf{con}(G)$ we construct (2,3)-regular Σ-embedded graph H such that $\mathbf{con}(G)$ is the contraction of H and $\mathbf{bw}(H) = \mathbf{bw}(\mathbf{con}(G))$. Then one can construct a hypergraph $\mathbf{ext}(H)$ such that $\mathbf{bw}(\mathcal{S}(G)) \leq \mathbf{bw}(\mathbf{ext}(H))$ and H is the concise graph of $\mathbf{ext}(H)$. Such a construction is similar to the case of planar graphs (see [11]) and we omit it here. Since **(iii)** is already proved for concise graphs of degree ≤ 3, we have that $\mathbf{bw}(\mathcal{S}(G)) \leq \mathbf{bw}(\mathbf{ext}(H)) \leq 3 \cdot \mathbf{bw}(H) = 3 \cdot \mathbf{bw}(\mathbf{con}(G))$ and **(iii)** follows.

So we proved that **(iii)** holds when $\mathbf{con}(G)$ is 2-connected. To finish the proof we use induction on the number of 2-connected components of $\mathbf{con}(G)$. □

Theorem 7. *For any* Σ-*embedded graph* G,
$\mathbf{bw}(G) \leq 3(\sqrt{4.5} + 2\sqrt{2 \cdot \mathbf{eg}(\Sigma)})\sqrt{\gamma(G) + 6 \cdot \mathbf{eg}(G)}$.

Proof. We use induction on the Euler genus of Σ. For $\Sigma = \mathbb{S}_0$ the result follows from Theorem 1. Suppose that the theorem is correct for all graphs that can be embedded in surfaces of Euler genus $< g$ for some $g > 0$. Let G be a D-dominated Σ-embedded graph where $\mathbf{eg}(\Sigma) = g$. If representativity of G is more than 6, By Lemma 4, there is a nicely D-dominated graph H such that G is a minor of H. Thus $\mathbf{bw}(G) \leq \mathbf{bw}(H)$ and by Lemmata 6 and 7, $\mathbf{bw}(G) \leq \mathbf{bw}(H) \leq 3(\sqrt{4.5} + 2\sqrt{2 \cdot \mathbf{eg}(\Sigma)})\sqrt{|D|}$.

If representativity of G is ≤ 6, let G' be the graph obtained from G by cutting along a non-contractible noose N of length ≤ 6. Let $G_1, \ldots, G_q$ be the

connected components of G'. Clearly, each of the components G_i has a dominating set of size at most $|D|+6$. By Lemma 5, $\mathbf{bw}(G) \leq \max_{1\leq i\leq q} \mathbf{bw}(G_i)+6$ and by Lemma 1, every component G_i of G' can be embedded in a surface Σ_i of Euler genus $\leq g-1$. Thus $\mathbf{bw}(G) \leq \max_{1\leq i\leq q} \mathbf{bw}(G_i)+6 \leq 3(\sqrt{4.5}+2\sqrt{2\cdot(g-1)})\sqrt{|D|+6+6\cdot(g-1)}+6 \leq 3(\sqrt{4.5}+2\sqrt{2\cdot g})\sqrt{|D|+6\cdot g}$. □

A simplification of the formula in Theorem 7 gives that any graph with dominating set $\leq k$ and Euler genus $\leq g$ has branchwidth at most $(7+9\sqrt{g})\sqrt{k+6g}$. Applying Theorem 7 to the reduced graph G' in the second step of the algorithm of Theorem 6 we have that $\mathbf{bw}(G') \leq (7+9\sqrt{g})\sqrt{k+6g}$. Therefore, it is enough to apply Amir's algorithm for $\omega = \frac{3}{2}(7+9\sqrt{g})\sqrt{k+6g}$ and get a tree decomposition of width $\leq (3+\frac{2}{3})\frac{3}{2}(7+9\sqrt{g})\sqrt{k+6g} = 5.5\cdot(7+9\sqrt{g})\sqrt{k+6g}$. This improves significantly the constants of the exponential part in the time of the algorithm in Theorem 6. As we will see in the next section, Theorem 7 has consequences to the design of subexponential parameterized algorithms for more parameters.

6 Generalizations

The combinatorial and algorithmic results of the previous two sections can be generalized to a general family of parameters. Due to lack of space we just mention the results and leave the proofs for the full version. We describe a general class of parameterized problems C including minimum vertex cover, the minimum edge dominating set, the minimum clique transversal set, the minimum vertex feedback set, the minimum maximal matching, variations of domination like minimum independent dominating set, the total minimum dominating set, the minimum perfect dominating set, the minimum perfect code, the minimum weighted dominating set, and the minimum total perfect dominating set, and prove that for any graph G every problem in P can be solved in $2^{O(\sqrt{k\cdot\mathbf{eg}(G)}+\mathbf{eg}(G))}n^{O(1)}$ steps. This implies that for $\mathbf{eg}(G) = o(\log n)$ all these problems can be solved in subexponential parameterized time (i.e. in $2^{o(k)}n^{O(1)}$-time) and for $\mathbf{eg}(\Sigma) = o(n)$ all these problems can be computed in subexponential time (i.e. in $2^{o(n)}$-time).

References

1. J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier, *Fixed parameter algorithms for dominating set and related problems on planar graphs,* Algorithmica, 33 (2002), pp. 461–493.
2. J. Alber, M. R. Fellows, and R. Niedermeier, *Efficient data reduction for dominating set: A linear problem kernel for the planar case,* in SWAT 2002, Springer, vol. 2368, Berlin, 2002, pp. 150–159. To appear in the Journal of the ACM.
3. E. Amir, *Efficient approximation for triangulation of minimum treewidth,* in Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (UAI-2001), San Francisco, CA, 2001, Morgan Kaufmann Publishers, pp. 7–15.

4. E. D. DEMAINE, F. V. FOMIN, M. HAJIAGHAYI, AND D. M. THILIKOS, *Fixed-parameter algorithms for the (k,r)-center in planar graphs and map graphs,* in The 30th International Colloquium on Automata, Languages and Programming (ICALP 2003), vol. 2719, 2003, pp. 829–844.
5. ——, *Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs,* in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2004, pp. 823–832. to appear.
6. E. D. DEMAINE AND M. HAJIAGHAYI, *Equivalence of local treewidth and linear local treewidth and its algorithmic applications,* in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2004. to appear.
7. E. D. DEMAINE, M. HAJIAGHAYI, AND D. M. THILIKOS, *Exponential speedup of fixed parameter algorithms on $K_{3,3}$-minor-free or K_5-minor-free graphs,* in The 13th Anual International Symposium on Algorithms and Computation—ISAAC 2002 (Vancouver, Canada), Springer, Lecture Notes in Computer Science, Berlin, vol.2518, 2002, pp. 262–273.
8. H. N. DJIDJEV AND S. M. VENKATESAN, *Planarization of graphs embedded on surfaces,* in WG, vol. 1017 of Lecture Notes in Comput. Sci., Springer, Berlin, 1995, pp. 62–72.
9. R. G. DOWNEY AND M. R. FELLOWS, *Parameterized Complexity,* Springer-Verlag, New York, 1999.
10. J. ELLIS, H. FAN, AND M. FELLOWS, *The dominating set problem is fixed parameter tractable for graphs of bounded genus,* in The 8th Scandinavian Workshop on Algorithm Theory—SWAT 2002 (Turku, Finland), Springer, Lecture Notes in Computer Science, Berlin, vol. 2368, 2002, pp. 180–189.
11. F. V. FOMIN AND D. M. THILIKOS, *Dominating sets in planar graphs: Branch-width and exponential speed-up,* in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003, pp. 168–177.
12. ——, *New upper bounds on the decomposability of planar graphs and fixed parameter algorithms,* Technical Report 240, Department of Informatics, University of Bergen, Norway, 2003. Extended abstract *A Simple and Fast Approach for Solving Problems on Planar Graphs* to appear in the proceedings of STACS 2004.
13. I. KANJ AND L. PERKOVIĆ, *Improved parameterized algorithms for planar dominating set,* in Mathematical Foundations of Computer Science—MFCS 2002, Springer, Lecture Notes in Computer Science, Berlin, vol.2420, 2002, pp. 399–410.
14. T. KLOKS, C. M. LEE, AND J. LIU, *New algorithms for k-face cover, k-feedback vertex set, and k-disjoint set on plane and planar graphs,* in The 28th International Workshop on Graph-Theoretic Concepts in Computer Science(WG 2002), Springer, Lecture Notes in Computer Science, Berlin, vol. 2573, 2002, pp. 282–296.
15. B. MOHAR AND C. THOMASSEN, *Graphs on surfaces,* Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 2001.
16. N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. VII. Disjoint paths on a surface,* J. Combin. Theory Ser. B, 45 (1988), pp. 212–254.
17. N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. X. Obstructions to tree-decomposition,* Journal of Combinatorial Theory Series B, 52 (1991), pp. 153–190.
18. N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. XI. Circuits on a surface,* J. Combin. Theory Ser. B, 60 (1994), pp. 72–106.

Selfish Unsplittable Flows*

Dimitris Fotakis[1,2], Spyros Kontogiannis[1,3], and Paul Spirakis[1]

[1] Research Academic Computer Technology Institute, Riga Feraiou 61, 26221 Patras, Greece.
{fotakis,kontog,spirakis}@cti.gr
[2] Dept. of Mathematical, Physical and Computational Sciences,
Aristotle University of Thessaloniki, 54006 Thessaloniki, Greece.
[3] Dept. of Computer Science, University of Ioannina, 45110 Ioannina, Greece.

Abstract. What is the price of anarchy when unsplittable demands are routed selfishly in general networks with load-dependent edge delays? Motivated by this question we generalize the model of [14] to the case of *weighted congestion games*. We show that varying demands of users crucially affect the nature of these games, which are no longer isomorphic to exact potential games, even for very simple instances. Indeed we construct examples where even a single-commodity (weighted) network congestion game may have no pure Nash equilibrium.
On the other hand, we study a special family of networks (which we call the ℓ-layered *networks*) and we prove that any weighted congestion game on such a network with resource delays equal to the congestions, possesses a pure Nash Equilibrium. We also show how to construct one in pseudo-polynomial time. Finally, we give a surprising answer to the question above for such games: The price of anarchy of any weighted ℓ-layered network congestion game with m edges and edge delays equal to the loads, is $\Theta\left(\frac{\log m}{\log\log m}\right)$.

1 Introduction

Consider a model where selfish users having varying demands compete for some shared resources. The quality of service provided by a resource decreases with its *congestion,* ie, the amount of demands of the users willing to be served by it. Each user may reveal its actual (unique) choice (called a *pure strategy*) among the resources available to it, or it may reveal a probability distribution for choosing one of its candidate resources (a *mixed strategy*). The users determine their actual behavior based on other users' behavior, but they do not cooperate. We are interested in situations where the users have reached some kind of equilibrium. The most popular notion of equilibrium in noncooperative game theory is the *Nash equilibrium:* a "stable point" among the users, from which no user is willing to deviate unilaterally. In [14] the notion of the *coordination ratio* or *price of anarchy* was introduced, as a means for measuring the performance degradation due to lack of users' coordination when sharing common goods.

A realistic scenario for the above model is when unsplittable demands are routed selfishly in general networks with load-dependent edge delays. When the underlying network

* This work was partially supported by the EU within the Future and Emerging Technologies Programme under contract IST-2001-33135 (CRESCCO) and within the 6th Framework Programme under contract 001907 (DELIS).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 593–605, 2004.

consists of two nodes and parallel links between them, there has been an extensive study on the existence and computability of equilibria, as well as on the price of anarchy. Motivated by the work of [14], we generalize their concept to the *weighted congestion games* in a non-trivial way. When users have identical demands, such a game is indeed isomorphic to an *exact potential game* ([19]) and thus always possesses a pure Nash equilibrium, ie, an equilibrium where each user adopts a pure strategy. We show that varying demands of users crucially affect the nature of these games, which are no longer isomorphic to exact potential games. Indeed we construct examples where even a single-commodity (weighted) network congestion game may have no pure Nash equilibrium at all.

On the other hand, we explore weighted congestion games on a special family of networks, the *ℓ-layered networks*. We prove the existence of pure Nash equilibria for such games. We also propose a pseudo-polynomial time algorithm for constructing one. Finally, we study the price of anarchy for these networks and we come to a rather surprising conclusion: Within constant factors, the worst case instance (wrt the price of anarchy) among weighted **ℓ-layered** network congestion games with m edges and edge delays equal to the loads, is the parallel links game introduced in [14].

1.1 The Model

Consider having a set of resources E in a system. For each $e \in E$, let $d_e(\cdot)$ be the delay per user that requests its service, as a function of the total usage of this resource by all the users. Each such function is considered to be non-decreasing in the total usage of the corresponding resource. Each resource may be represented by a pair of points: an entry point to the resource and an exit point from it. So, we represent each resource by an arc from its entry point to its exit point and we associate with this arc the cost (eg, the delay as a function of the load of this resource) that each user has to pay if she is served by this resource. The entry/exit points of the resources need not be unique; they may coincide in order to express the possibility of offering joint service to users, that consists of a sequence of resources. We denote by V the set of all entry/exit points of the resources in the system. Any nonempty collection of resources corresponding to a directed path in $G \equiv (V, E)$ comprises an *action* in the system.

Let $N \equiv [n]$ be a set of users, each willing to adopt some action in the system. $\forall i \in N$, let w_i denote user i's *demand* (eg, the flow rate from a source node to a destination node), while $\Pi_i \subseteq 2^E \setminus \emptyset$ is the collection of actions, any of which would satisfy user i (eg, alternative routes from a source to a destination node, if G represents a communication network). The collection Π_i is called the *action set* of user i and each of its elements contains at least one resource. Any tuple $\varpi \in \Pi \equiv \times_{i=1}^{n} \Pi_i$ is a *pure strategies profile,* or a *configuration* of the users. Any real vector $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ s.t. $\forall i \in [n]$, $p_i : \Pi_i \to [0, 1]$ is a probability distribution over the set of allowable actions for user i, is called a *mixed strategies profile* for the n users.

A congestion model typically deals with users of identical demands, and thus, resource delay functions depend on the *number* of users adopting each action ([21,19,7]). In this work we consider the more general case, where a *weighted congestion model* is the tuple $((w_i)_{i \in N}, (\Pi_i)_{i \in N}, (d_e)_{e \in E})$. That is, we allow the users to have different demands for service from the whole system, and thus affect the resource delay functions in a different way, depending on their own weights. The *weighted congestion game* associated

with this model, is the game in strategic form with the set of users N and user demands $(w_i)_{i\in N}$, the action sets $(\Pi_i)_{i\in N}$ and cost functions $(\lambda^i_{\varpi_i})_{i\in N,\varpi_i\in\Pi_i}$ defined as follows: For any configuration $\varpi \in \Pi$ and $\forall e \in E$, let $\Lambda_e(\varpi) = \{i \in N : e \in \varpi_i\}$ be the set of users exploiting resource e according to ϖ. The *cost* $\lambda^i(\varpi)$ *of user* i *for adopting strategy* $\varpi_i \in \Pi_i$ in a given configuration ϖ is $\lambda^i(\varpi) = \lambda_{\varpi_i}(\varpi) = \sum_{e\in\varpi_i} d_e(\theta_e(\varpi))$ where, $\forall e \in E, \theta_e(\varpi) \equiv \sum_{i\in\Lambda_e(\varpi)} w_i$ is the load on resource e wrt the configuration ϖ. On the other hand, for a mixed strategies profile $\mathbf{p}$, the *expected cost of user* i *for adopting strategy* $\varpi_i \in \Pi_i$ is $\lambda^i_{\varpi_i}(\mathbf{p}) = \sum_{\varpi^{-i}\in\Pi^{-i}} P(\mathbf{p}^{-i}, \varpi^{-i}) \cdot \sum_{e\in\varpi_i} d_e\left(\theta_e(\varpi^{-i} \oplus \varpi_i)\right)$ where, ϖ^{-i} is a configuration of all the users except for i, $\mathbf{p}^{-i}$ is the mixed strategies profile of all users except for i, $\varpi^{-i} \oplus \varpi_i$ is the new configuration with i choosing strategy ϖ_i, and $P(\mathbf{p}^{-i}, \varpi^{-i}) \equiv \prod_{j\in N\setminus\{i\}} p_j(\varpi_j)$ is the occurrence probability of ϖ^{-i}.

A congestion game in which all users are indistinguishable (ie, they have the same user cost functions) and have the same action set, is called *symmetric*. When each user's action set Π_i consists of sets of resources that comprise (simple) paths between a unique origin-destination pair of nodes (s_i, t_i) in (V, E), we referto a *network congestion game*. If additionally all origin-destination pairs of the users coincide with a unique pair (s, t) we have a *single commodity network congestion game* and then all users share exactly the same action set. Observe that a single-commodity network congestion game is not necessarily symmetric because the users may have different demands and thus their cost functions will also differ.

Selfish Behavior. Fix an arbitrary (mixed in general) strategies profile $\mathbf{p}$ for a congestion game $((w_i)_{i\in N}, (\Pi_i)_{i\in N}, (d_e)_{e\in E})$. We say that $\mathbf{p}$ is a *Nash Equilibrium* (*NE*) if and only if $\forall i \in N$, $\forall \varpi_i, \pi_i \in \Pi_i$, $p_i(\varpi_i) > 0 \Rightarrow \lambda^i_{\varpi_i}(\mathbf{p}) \leq \lambda^i_{\pi_i}(\mathbf{p})$. A configuration $\varpi \in \Pi$ is a *Pure Nash Equilibrium* (*PNE*) if and only if $\forall i \in N, \forall \pi_i \in \Pi_i$, $\lambda_{\varpi_i}(\varpi) \leq \lambda_{\pi_i}(\varpi^{-i} \oplus \pi_i)$ where, $\varpi^{-i} \oplus \pi_i$ is the same configuration with ϖ except for user i that now chooses action π_i. The *social cost* $SC(\mathbf{p})$ in this congestion game is $SC(\mathbf{p}) = \sum_{\varpi\in\Pi} P(\mathbf{p}, \varpi) \cdot \max_{i\in N}\{\lambda_{\varpi_i}(\varpi)\}$, where $P(\mathbf{p}, \varpi) \equiv \prod_{i=1}^{n} p_i(\varpi_i)$ is the probability of configuration ϖ occurring, wrt the mixed strategies profile $\mathbf{p}$. The *social optimum* of this game is defined as $OPT = \min_{\varpi\in\Pi}\{\max_{i\in N}[\lambda_{\varpi_i}(\varpi)]\}$. The *price of anarchy* for this game is then defined as $\mathcal{R} = \max_{\mathbf{p} \text{ is a NE}} \left\{\frac{SC(\mathbf{p})}{OPT}\right\}$.

Configuration Paths and Dynamics Graph. For a congestion game $\Gamma = ((w_i)_{i\in N}, (\Pi_i)_{i\in N}, (d_e)_{e\in E})$, a *path* in $\Pi = \times_{i\in N}\Pi_i$ is a sequence of configurations $\gamma = (\varpi(0), \varpi(1), \ldots, \varpi(k))$ s.t. $\forall j \in [k]$, $\varpi(j) = (\varpi(j-1))^{-i} \oplus \pi_i$, for some $i \in N$ and $\pi_i \in \Pi_i$. γ is a *closed path* if $\varpi(0) = \varpi(k)$. It is a *simple path* if no configuration is contained in it more than once. γ is an *improvement path* wrt Γ, if $\forall j \in [k]$, $\lambda^{i_j}(\varpi(j)) < \lambda^{i_j}(\varpi(j-1))$ where i_j is the unique user differing in its strategy between $\varpi(j)$ and $\varpi(j-1)$. The *Dynamics Graph* of Γ is a directed graph whose vertices are configurations and there is an arc from a configuration ϖ to a configuration $\varpi^{-i} \oplus \pi_i$ for some $\pi_i \in \Pi_i$ if and only if $\lambda^i(\varpi) > \lambda^i(\varpi^{-i} \oplus \pi_i)$.

Layered Networks. We now define a special family of networks whose behavior wrt the price of anarchy (we shall prove that) is asymptotically equivalent to that of the

parallel links model of [14], which is actually a 1-layered network: Let $\ell \geq 1$ be an integer. A directed network $G = (V,E)$ with a distinguished source - destination pair (s,t), $s,t \in V$, is *ℓ-layered* if every directed $s-t$ path has length exactly ℓ and each node lies on a directed $s-t$ path. In a layered network there are no directed cycles and all directed paths are simple. In the following, we always use m to denote the number $|E|$ of edges in an **ℓ-layered** network $G = (V, E)$.

Atomic Assignments. We consider *atomic* assignments of users to actions, ie, each user $i \in N$ requires all its demand w_i from exactly one allowable action $\varpi_i \in \Pi_i$. Nevertheless, we allow users to adopt mixed strategies. Our focus in this paper is two-fold: We are interested in families of resource delay functions for which the weighted single-commodity network congestion game has a PNE, and we are also interested in the price of anarchy for a special case of this problem where G has the form of an **ℓ-layered** network (to be defined later) and the delay functions are identical to the loads of the resources.

1.2 Related Work

Existence and Tractability of PNE. It is already known that the class of unweighted (atomic) congestion games (ie, users have the same demands and thus, the same affection on the resource delay functions) is guaranteed to have at least one PNE: actually, Rosenthal ([21]) proved that any potential game has at least one PNE and it is easy to write any unweighted congestion game as an exact potential game using Rosenthal's potential function[1] (eg, [7, Thm1]). In [7] it is proved that a PNE for any unweighted single-commodity network congestion game[2] (no matter what resource delay functions are considered, so long as they are non-decreasing with loads) can be constructed in polynomial time, by computing the optimum of Rosenthal's potential function, through a nice reduction to min-cost flow. On the other hand, it is shown that even for a symmetric congestion game or an unweighted multicommodity network congestion game, it is PLS-complete to find a PNE (though it certainly exists).

The special case of single-commodity, parallel-edges network congestion game where the resources are considered to behave as parallel machines, has been extensively studied in recent literature. In [9] it was shown that for the case of users with varying demands and uniformly related parallel machines, there is always a PNE which can be constructed in polynomial time. It was also shown that it is NP-hard to construct the best or the worst PNE. In [10] it was proved that the fully mixed NE (FMNE), introduced and thoroughly studied in [17], is worse than any PNE, and any NE is at most $(6+\epsilon)$ times worse than the FMNE, for varying users and identical parallel machines. In [16] it was shown that the FMNE is the worst possible for the case of two related machines and tasks of the same size. In [15] it was proved that the FMNE is the worst possible when the global objective is the sum of squares of loads.

[8] studies the problem of constructing a PNE from any initial configuration, of social cost at most equal to that of the initial configuration. This immediately implies the existence of a PTAS for computing a PNE of minimum social cost: first compute

[1] For more details on Potential Games, see [19].

[2] Since [7] only considers unit-demand users, this is also a symmetric network congestion game.

a configuration of social cost at most $(1+\epsilon)$ times the social optimum ([11]), and consequently transform it into a PNE of at most the same social cost. In [6] it is also shown that even for the unrelated parallel machines case a PNE always exists, and a potential-based argument proves a convergence time (in case of integer demands) from arbitrary initial configuration to a PNE in time $O\left(mW_{\text{tot}} + 4^{W_{\text{tot}}/m+w_{\max}}\right)$ where $W_{\text{tot}} = \sum_{i \in N} w_i$ and $w_{\max} = \max_{i \in N}\{w_i\}$.

[18] studies the problem of weighted parallel-edges network congestion games with user-specific costs: each allowable action of a user consists of a single resource and each user has its own private cost function for each resource. It is shown that: (1) weighted (parallel-edges network) congestion games involving only two users, or only two possible actions for all the users, or equal delay functions (and thus, equal weights), always possess a PNE; (2) even a single-commodity, 3-user, 3-actions, weighted (parallel-edges network) congestion game may not possess a PNE (using 3-wise linear delay functions).

Price of Anarchy in Congestion Games. In the seminal paper [14] the notion of coordination ratio, or price of anarchy, was introduced as a means for measuring the performance degradation due to lack of users' coordination when sharing common resources. In this work it was proved that the price of anarchy is 3/2 for two related parallel machines, while for m machines and users of varying demands, $\mathcal{R} = \Omega(\log m/\log\log m)$ and $\mathcal{R} = O\left(\sqrt{m \log m}\right)$. For m identical parallel machines, [17] proved that $\mathcal{R} = \Theta(\log m/\log\log m)$ for the FMNE, while for the case of m identical parallel machines and users of varying demands it was shown in [13] that $\mathcal{R} = \Theta(\log m/\log\log m)$. In [4] it was finally shown that $\mathcal{R} = \Theta(\log m/\log\log\log m)$ for the general case of related machines and users of varying demands. [3] presents a thorough study of the case of general, monotone delay functions on parallel machines, with emphasis on delay functions from queuing theory. Unlike the case of linear delays, they show that the price of anarchy for non-linear delays is in general far worse and often even unbounded.

In [22] the price of anarchy in a multicommodity network congestion game among infinitely many users, each of negligible demand, is studied. The social cost in this case is expressed by the total delay paid by the whole flow in the system. For linear resource delays, the price of anarchy is at most 4/3. For general, continuous, non-decreasing resource delay functions, the total delay of any Nash flow is at most equal to the total delay of an optimal flow for double flow demands. [23] proves that for this setting, it is actually the class of allowable latency functions and not the specific topology of a network that determines the price of anarchy.

1.3 Our Contribution

In this paper, we generalize the model of [14] (KP-model) to the weighted congestion games. We also define a special class of networks, the **ℓ-layered** networks, which demonstrate a rather surprising behavior: their worst instance wrt the price of anarchy is (within constant factors) the parallel links network introduced in [14]. More specifically, we prove that: (I) Weighted congestion games are *not* isomorphic to potential games. We show the existence of weighted single-commodity network congestion games with resource delays being either linear or 2-wise linear functions of the loads, for which there

PNE cannot exist (lemma 1). (II) There exist weighted single-commodity network congestion games which admit no exact potential function, even when the resource delays are identical to their loads (lemma 2). (III) For a weighted ℓ-layered network congestion game with resource delays equal to their loads, at least one PNE exists and can be constructed in pseudo-polynomial time (theorem 1). (IV) The price of anarchy of any weighted ℓ-layered network congestion game with m resources (edges) and resource delays equal to their loads, is at most $8\,e\,(\frac{\log m}{\log\log m}+1)$, where e is the basis of the natural logarithm (theorem 2). To our knowledge this is the first time that the KP-model is studied in non-trivial networks (other than the parallel links).

2 Pure Nash Equilibria

In this section we deal with the existence and tractability of PNE in the weighted single-commodity network congestion games. First we show that it is not always the case that a PNE exists for such a congestion game, even when we allow only linear and 2-wise linear (ie, the maximum of two linear functions) resource delays. In contrast, it is well known ([21,7]) that any unweighted (not necessarily single-commodity, or even network) congestion game has a PNE, for any kind of non-decreasing delays.

Lemma 1. *There exist instances of weighted single-commodity network congestion games with resource delays being either linear or 2-wise linear functions of the loads, for which there is no PNE.*

Proof. We demonstrate this by the example shown in figure 1. In this example there are exactly two users of demands $w_1 = 1$ and $w_2 = 2$, from node s to node t. The possible paths that the two users may follow are labeled in the figure. The resource delay functions are indicated by the 3 possible values they may take given the two users. Observe now that this example has no PNE: there is a simple closed path $\gamma = ((P3,P2),(P3,P4),(P1,P4),(P1,P2),(P3,P2))$ of length 4 that is an improvement path (actually, each defecting user moves to its new best choice) and additionally, any other configuration not belonging in γ is either one, or two best-choice moves away from some of these nodes. Therefore there is no sink in the Dynamics Graph of the game and thus there exists no PNE. Observe that the delay functions are not user-specific in our example, as was the case in [18]. □

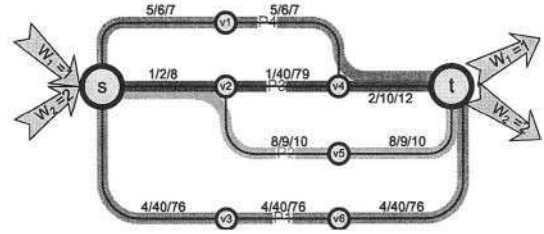


Fig. 1. A weighted single-commodity network congestion game that has no PNE, for two players with demands $w_1 = 1$ and $w_2 = 2$. The notation $a/b/c$ means that a load of 1 has delay a, a load of 2 has delay b and a load of 3 has delay c.

Consequently we show that there may exist no exact potential function[3] for a weighted single-commodity network congestion game, even when the resource delays are identical to their loads. The next argument shows that theorem 3.1 of [19] does not hold anymore even in this simplest case of weighted congestion games.

Lemma 2. *There exist weighted single-commodity network congestion games which are not exact potential games, even for resource delays identical to their loads.*

Proof. Let $\Gamma = ((w_i)_{i\in N}, (\Pi_i)_{i\in N}, (d_e)_{e\in E})$ denote a weighted single commodity network congestion game with $d_e(x) = x$, $\forall e \in E$. Let's define the quantity $I(\gamma,\lambda) = \sum_{k=1}^{r} \left[\lambda^{i_k}(\varpi(k)) - \lambda^{i_k}(\varpi(k-1))\right]$, where i_k is the unique user in which the configurations $\varpi(k)$ and $\varpi(k-1)$ differ. Our proof is based on the fact that Γ is an (exact) potential game if and only if every simple closed path γ of length 4 has $I(\gamma,\lambda) = 0$ ([19, Thm2.8]). Indeed, for an arbitrary initial configuration ϖ and any $\pi_1 \in \Pi_1\backslash\{\varpi_1\}$, $\pi_2 \in \Pi_2\backslash\{\varpi_2\}$, we consider the closed, simple 4-path $\gamma = (\varpi, \varpi^{-1}\oplus \pi_1, \varpi^{-(1,2)} \oplus (\pi_1,\pi_2), \varpi^{-2}\oplus\pi_2), \varpi$. We then prove (see full paper) that $I = (w_1 - w_2)\cdot[|(\pi_1 \setminus \varpi_1)\cap(\pi_2\setminus\varpi_2)| + |(\varpi_1\setminus\pi_1)\cap(\varpi_2\setminus\pi_2)| - |(\varpi_1\setminus\pi_1)\cap(\pi_2\setminus\varpi_2)| - |(\pi_1\setminus\varpi_1)\cap(\varpi_2\setminus\pi_2)|]$, which is typically not equal to zero for a single-commodity network. It should be noted that the second parameter, which is network dependent, can be non-zero even for some cycle of a very simple network. For example, in the network of figure 1 (which is a simple 2-layered network) the simple closed path $\gamma = (\varpi(0) = (P1,P3), \varpi(1) = (P2,P3), \varpi(2) = (P2,P1), \varpi(3) = (P1,P1), \varpi(4) = (P1,P3))$ has this quantity equal to –4 and thus no weighted single commodity network congestion game on this network can admit an exact potential. □

Our next step is to focus our interest on the **ℓ-layered** networks with resource delays identical to their loads. We shall prove that any weighted **ℓ-layered** network congestion game with these delays admits at least one PNE, which can be computed in pseudo-polynomial time. Although we already know that even the case of weighted **ℓ-layered** network congestion games with delays equal to the loads cannot have any exact potential[4], we will next show that $\Phi(\varpi) \equiv \sum_{e\in E}[\theta_e(\varpi)]^2$ is a **b**-potential for such a game and some positive ***n*-vector b,** assuring the existence of a PNE.

Theorem 1. *For any weighted **ℓ-layered** network congestion game with resource delays equal to their loads, at least one PNE exists and can be computed in pseudo-polynomial time.*

Proof. Fix an arbitrary **ℓ-layered** network *(V, E)* and denote by $\mathcal{P}$ all the $s-t$ paths in it from the unique source s to the unique destination t. Let $\varpi \in \mathcal{P}^n$ be an arbitrary configuration of the users for the corresponding congestion game on *(V, E)*. Also, let i be a user of demand w_i and fix some path $\pi_i \in \mathcal{P}$. Denote $\varpi' \equiv \varpi^{-i} \oplus \pi_i$. Observe that $\Phi(\varpi) - \Phi(\varpi') = \sum_{e\in\varpi_i\setminus\pi_i}\left(\theta_e^2(\varpi) - \theta_e^2(\varpi')\right) + \sum_{e\in\pi_i\setminus\varpi_i}\left(\theta_e^2(\varpi) - \theta_e^2(\varpi')\right) = \sum_{e\in\varpi_i\setminus\pi_i}\left([\theta_e(\varpi^{-i}) + w_i]^2 - \theta_e^2(\varpi^{-i})\right) + \sum_{e\in\pi_i\setminus\varpi_i}\left(\theta_e^2(\varpi^{-i}) - [\theta_e(\varpi^{-i}) + w_i]^2\right)$

[3] Fix a vector $\mathbf{b} \in \mathbb{R}^n_{>0}$. $F : \times_{i\in N}\Pi_i \to \mathbb{R}$ is a **b**-*potential* for a weighted congestion game $\Gamma = ((w_i)_{i\in N}, (\Pi_i)_{i\in N}, (d_e)_{e\in E})$, if $\forall \varpi \in \times_{i\in N}\Pi_i$, $\forall i \in N, \forall \pi_i \in \Pi_i$, $\lambda^i(\varpi) - \lambda^i(\varpi^{-i}\oplus\pi_i) = b_i\cdot[F(\varpi) - F(\varpi^{-i}\oplus\pi_i)]$). It is an *exact potential* for Γ, if $\mathbf{b} = \mathbf{1}$.

[4] The example at the end of the proof of lemma 2 involves the 2-layered network of figure 1.

$= 2w_i \cdot \left(\sum_{e \in \varpi_i \setminus \pi_i} \theta_e(\varpi^{-i}) - \sum_{e \in \pi_i \setminus \varpi_i} \theta_e(\varpi^{-i}) \right) = 2w_i \cdot \left[\lambda^i(\varpi) - \lambda^i(\varpi') \right]$, since, $\forall e \in \varpi_i \cap \pi_i \; \theta_e(\varpi) = \theta_e(\varpi')$, in ℓ-**layered** networks $|\varpi_i \setminus \pi_i| = |\pi_i \setminus \varpi_i|$, $\lambda^i(\varpi) = \sum_{e \in \varpi_i} \theta_e(\varpi) = \sum_{e \in \varpi_i \setminus \pi_i} \theta_e(\varpi^{-i}) + w_i |\varpi_i \setminus \pi_i| + \sum_{e \in \varpi_i \cap \pi_i} \theta_e(\varpi)$ and $\lambda^i(\varpi') = \sum_{e \in \pi_i} \theta_e(\varpi') = \sum_{e \in \pi_i \setminus \varpi_i} \theta_e(\varpi^{-i}) + w_i |\pi_i \setminus \varpi_i| + \sum_{e \in \varpi_i \cap \pi_i} \theta_e(\varpi)$. Thus, Φ is a **b**-potential for our game, where $\mathbf{p} = (1/(2w_i))_{i \in N} > \mathbf{0}$, assuring the existence of at least one PNE.

Wlog assume that the users have integer weights. Then each user performing any improving defection, must reduce its cost by at least 1 and thus the potential function decreases by at least $2w_{\min} \geq 2$ along each arc of the Dynamics Graph of the game. Consequently, the **naïve** algorithm that, starting from an arbitrary initial configuration $\varpi \in \mathcal{P}$, follows any improvement path that leads to a sink (ie, a PNE) of the Dynamics Graph, cannot move more than $\frac{1}{2}|E|W_{\text{tot}}^2$ times, since $\forall \varpi \in \mathcal{P}, \Phi(\varpi) \leq |E|W_{\text{tot}}^2$. □

3 The Price of Anarchy in ℓ-Layered Networks

In this section we focus our interest on weighted ℓ-**layered** network congestion games where the resource delays are identical to their loads. The main reason why we focus on this specific category of resource delays is that selfish unsplittable flows have usually unbounded price of anarchy. In [22, p. 256] an example is given where the price of anarchy is unbounded. This example is easily converted in an ℓ-**layered** network. The resource delay functions used are either constant or M/M/1-like delay functions. But we can be equally bad even with linear resource delay functions: Observe the following example of figure 2. Two users, each of unit demand, want to move selfishly from s to t. The edge delays are shown above them. We assume that $a \gg b \gg 1 \geq c$). It is easy to see that the configuration (sCBt,sADt) is a PNE of social cost $2 + b$ while the optimum configuration is (sABt,sCDt) whose social optimum is $2 + c$. Thus, $\mathcal{R} = \frac{b+2}{c+2}$. So in this section we study weighted ℓ-**layered** networks whose resource delays equal their loads. Our main tool is to interpret mixed (in general) strategies profiles into some sort of (splittable) flows in this network.

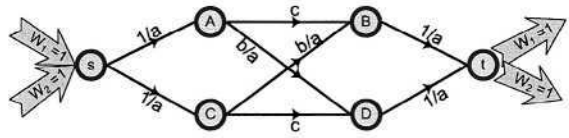


Fig. 2. Example of an ℓ-**layered** network with linear resource delays and unbounded anarchy.

Flows and Mixed Strategies Profiles. Fix an arbitrary ℓ-**layered** network $G = (V, E)$ and n distinct users willing to satisfy their own traffic demands from the unique source $s \in V$ to the unique destination $t \in V$. Again, $\mathbf{w} = (w_i)_{i \in [n]}$ denotes the varying demands of the users. Fix an arbitrary mixed strategies profile $\mathbf{p} = (p_1, p_2, \ldots, p_n)$. A *feasible flow* for the n users is a function $\rho : \mathcal{P} \mapsto \mathbb{R}_{\geq 0}$, s.t. $\sum_{\pi \in \mathcal{P}} \rho(\pi) = W_{\text{tot}} \equiv \sum_{i \in [n]} w_i$, ie, all users' demands are actually met. We distinguish between *unsplittable* and *splittable* (feasible) flows. A flow is unsplittable if each user's traffic demand is

satisfied by a unique path of $\mathcal{P}$. A flow is splittable if the traffic demand of each user is divided into infinitesimally small parts which are then routed over several paths of $\mathcal{P}$.

We map any profile $\mathbf{p}$ to a flow $\rho_{\mathbf{p}}$ as follows: $\forall \pi \in \mathcal{P}$, $\rho_{\mathbf{p}}(\pi) \equiv \sum_{i \in [n]} w_i \cdot p_i(\pi)$. That is, we handle the *expected load traveling along* π *according to* $\mathbf{p}$ as a splittable flow created by all the users, where $\forall i \in [n]$, i routes a fraction $p_i(\pi)$ of its total demand w_i along π. Observe that for the special case where $\mathbf{p}$ is a pure strategies profile, the corresponding flow is then unsplittable. Recall now that $\forall e \in E$, $\theta_e(\mathbf{p}) \equiv \sum_{i=1}^{n} w_i \sum_{\pi \ni e} p_i(\pi) = \sum_{\pi \ni e} \rho_{\mathbf{p}}(\pi) \equiv \theta_e(\rho_{\mathbf{p}})$ is the expected load (and in our case, also the expected delay) of e wrt $\mathbf{p}$. As for the expected delay along a path $\pi \in \mathcal{P}$ according to $\mathbf{p}$, this is $\theta_\pi(\mathbf{p}) \equiv \sum_{e \in \pi} \theta_e(\mathbf{p}) = \sum_{e \in \pi} \sum_{\pi' \ni e} \rho_{\mathbf{p}}(\pi') = \sum_{\pi' \in \mathcal{P}} |\pi \cap \pi'| \rho_{\mathbf{p}}(\pi') \equiv \theta_\pi(\rho_{\mathbf{p}})$. Let $\theta^{\min}(\rho) = \min_{\pi \in \mathcal{P}}\{\theta_\pi(\rho)\}$ be the minimum expected delay among all $s-t$ paths. From now on for simplicity we drop the subscript of $\mathbf{p}$ from its corresponding flow $\rho_{\mathbf{p}}$, when this is clear by the context. We evaluate flow ρ using the objective of *maximum latency among used paths:* $L(\rho) \equiv \max_{\pi:\rho(\pi)>0}\{\theta_\pi(\rho)\} = \max_{\pi:\exists i,\, p_i(\pi)>0}\{\theta_\pi(\mathbf{p})\} \equiv L(\mathbf{p})$. This is nothing more than the *maximum expected delay paid by the users,* wrt $\mathbf{p}$. Sometimes we also evaluate flow ρ using the objective of *total latency:* $C(\rho) \equiv \sum_{\pi \in \mathcal{P}} \rho(\pi)\theta_\pi(\rho) = \sum_{e \in E} \theta_e^2(\rho) = \sum_{e \in E} \theta_e^2(\mathbf{p}) \equiv C(\mathbf{p})$. We get the second equality by summing over the edges of π and reversing the order of the summation. From now on we denote by ρ^* and ρ_f^* the optimal unsplittable and splittable flows respectively.

Flows at Nash Equilibrium. Let $\mathbf{p}$ be a mixed strategies profile and let ρ be the corresponding flow. The cost of user i on path π is $\lambda_\pi^i(\mathbf{p}) = \ell w_i + \theta_\pi^{-i}(\mathbf{p})$ (G is an ℓ-**layered** network with resource delays equal to the loads), where $\theta_\pi^{-i}(\mathbf{p})$ is the expected delay along path π if the demand of user i was removed from the system: $\theta_\pi^{-i}(\mathbf{p}) = \sum_{\pi' \in \mathcal{P}} |\pi \cap \pi'| \sum_{j \neq i} w_j p_j(\pi') = \theta_\pi(\mathbf{p}) - w_i \sum_{\pi' \in \mathcal{P}} |\pi \cap \pi'| p_i(\pi')$ and thus, $\lambda_\pi^i(\mathbf{p}) = \theta_\pi(\mathbf{p}) + \left[\ell - \sum_{\pi' \in \mathcal{P}} |\pi \cap \pi'| p_i(\pi')\right] w_i$. Observe now that, if $\mathbf{p}$ is a NE, then $L(\mathbf{p}) = L(\rho) \leq \theta^{\min}(\rho) + \ell w_{\max}$. Otherwise, the users routing their traffic on a path of expected latency greater than $\theta^{\min}(\rho) + \ell w_{\max}$ could improve their latency by defecting to a path of expected latency $\theta^{\min}(\rho)$. When we say that a flow ρ corresponding to a mixed strategies profile $\mathbf{p}$ is a NE, we imply that it is actually $\mathbf{p}$ which is a NE.

Maximum Latency versus Total Latency. We show that a splittable flow is optimal wrt the objective of maximum latency if and only if it is optimal wrt the objective of total latency. As a corollary, we obtain that the optimal splittable flow defines a NE where all users adopt the same mixed strategy for their demands. Consider the $s-t$ flow polytope (FP): $\left\{\sum_{\pi \in \mathcal{P}} \rho(\pi) = W_{\text{tot}};\ \rho(\pi) \geq 0,\ \forall \pi \in \mathcal{P}\right\}$. One can ask for the flow that minimizes either $L(\rho) = \max_{\pi:\rho(\pi)>0}\{\theta_\pi(\rho)\}$, or $C(\rho) = \sum_{e \in E} \theta_e^2(\rho)$. For general resource delay functions the two objectives are different. However, in the special case that the delay of an edge is equal to the load routed through it, we prove that the two objectives are equivalent.

Lemma 3. *There is a unique splittable flow* ρ *which minimizes both* $L(\rho)$ *and* $C(\rho)$.

Proof. For every flow ρ, the average latency of ρ cannot exceed the maximum latency induced by ρ: $C(\rho) = \sum_{\pi \in \mathcal{P}} \rho(\pi)\theta_\pi(\rho) = \sum_{\pi:\rho(\pi)>0} \rho(\pi)\theta_\pi(\rho) \leq L(\rho)W_{\text{tot}}$. A

(splittable) flow ρ minimizes $C(\rho) = \sum_{e \in E} \theta_e^2(\rho)$ if and only if for every $\pi_1, \pi_2 \in \mathcal{P}$: $\rho(\pi_1) > 0$, $\theta_{\pi_1}(\rho) \leq \theta_{\pi_2}(\rho)$ (eg, [2], [20, Section 7.2], [22, Corollary 4.2]). Hence, if ρ is optimal wrt the total latency, then $\forall\, \pi_1, \pi_2 \in \mathcal{P} : \rho(\pi_1) \cdot \rho(\pi_2) > 0$, $\theta_{\pi_1}(\rho) = \theta_{\pi_2}(\rho) = L(\rho)$, implying that $C(\rho) = \sum_{\pi \in \mathcal{P}:\rho(\pi)>0} \rho(\pi)\theta_\pi(\rho) = L(\rho)W_{\text{tot}}$.

Let ρ be the flow that minimizes the total latency and let ρ' be the flow that minimizes the maximum latency. We prove the lemma by establishing that the two flows are identical. Observe that $L(\rho') \geq \frac{C(\rho')}{W_{\text{tot}}} \geq \frac{C(\rho)}{W_{\text{tot}}} = L(\rho)$. The first inequality follows from the general bound on $C(\rho')$, while the rest comes from the assumption that ρ minimizes the total latency. On the other hand, $L(\rho') \leq L(\rho)$ due to the assumption that the flow ρ' minimizes the maximum latency. Hence, $L(\rho') = L(\rho)$ and $C(\rho') = C(\rho)$. Since the function $C(\rho)$ is strictly convex and the $s-t$ flow polytope (FP) is also convex, there is a unique flow which minimizes the total latency. □

Lemma 3 implies that the optimal splittable flow can be computed in polynomial time, since it is the solution of a convex program. The following corollary states that the optimal splittable flow defines a NE where all users follow exactly the same strategy.

Corollary 1. *Let ρ_f^* be the optimal splittable flow and* $\mathbf{p}$ *the mixed strategies profile where $\forall i \in N$ and $\forall \pi \in \mathcal{P}$, $p_i(\pi) = \rho_f^*(\pi)/W_{\text{tot}}$. Then,* $\mathbf{p}$ *is a NE.*

Proof. See full paper. □

An Upper Bound on the Social Cost. We derive an upper bound on the social cost of any strategy profile whose maximum expected delay (ie, the maximum latency of its flow) is within a constant factor from the maximum latency of an optimal flow.

Lemma 4. *Let ρ^* be the optimal unsplittable flow, and let* $\mathbf{p}$ *be a mixed strategies profile and ρ its corresponding flow. If $L(\mathbf{p}) = L(\rho) \leq \alpha\, L(\rho^*)$, for some $\alpha \geq 1$, then, if $m = |E|$ is the number of edges in the network,* $\mathrm{SC}(\mathbf{p}) \leq (\alpha+1)\, \mathrm{O}(\frac{\log m}{\log \log m})\, L(\rho^*)$.

Proof. $\forall e \in E$ and $\forall i \in [n]$, let the r.v. describing the actual load routed through e by i be $X_{e,i} = w_i \cdot \mathbf{I}_{[i\text{'s demand is routed through a path } \pi \ni e]}$. Then, $\mathbf{E}[X_{e,i}] = \sum_{\pi \ni e} w_i p_i(\pi)$. Since each user selects its path independently, for each fixed edge e, the r.v.s of $\{X_{e,i}\}_{i \in [n]}$ are independent of each other. $\forall e \in E$, let $X_e = \sum_{i=1}^n X_{e,i}$ describe the actual load routed through e, and thus, the actual delay paid by any user traversing e. By linearity of expectation, $\mathbf{E}[X_e] = \theta_e(\rho)$. By applying the Hoeffding bound[5] with $w = w_{\max}$ and $t = \mathrm{e}\kappa \max\{\theta_e(\rho), w_{\max}\}$, we obtain that $\forall \kappa \geq 1$, $\mathbf{P}[X_e \geq \mathrm{e}\kappa \max\{\theta_e(\rho), w_{\max}\}] \leq \kappa^{-\mathrm{e}\kappa}$. By the union bound we conclude that $\mathbf{P}[\exists e \in E : X_e \geq \mathrm{e}\kappa \max\{\theta_e(\rho), w_{\max}\}] \leq m\kappa^{-\mathrm{e}\kappa}$. Now, $\forall \pi \in \mathcal{P} : \rho(\pi) > 0$, we define the r.v. $X_\pi = \sum_{e \in \pi} X_e$ describing the actual delay along π. The social cost of $\mathbf{p}$, which is equal to the expected maximum delay experienced by some user, cannot exceed the expected maximum delay among paths π with $\rho(\pi) > 0$. Formally, $\mathrm{SC}(\mathbf{p}) \leq \mathbf{E}[\max_{\pi:\rho(\pi)>0}\{X_\pi\}]$. If $\forall e \in E$, $X_e \leq \mathrm{e}\kappa \max\{\theta_e(\rho), w_{\max}\}$, then $\forall \pi \in$

[5] We use the following version of the Hoeffding bound ([12]): Let $X_1, X_2, \ldots, X_n$ be independent r.v.s with values in $[0, w]$. Let $X = \sum_{i=1}^n X_i$. Then, $\forall t > 0$, $\mathbf{P}[X \geq t] \leq \left(\frac{\mathrm{e}\,\mathbf{E}[X]}{t}\right)^{t/w}$.

$\mathcal{P} : \rho(\pi) > 0$, $X_\pi \leq \mathrm{e}\kappa \sum_{e\in\pi} \max\{\theta_e(\rho), w_{\max}\} \leq \mathrm{e}\kappa \sum_{e\in\pi}(\theta_e(\rho) + w_{\max}) = \mathrm{e}\kappa(\theta_\pi(\rho) + \ell w_{\max}) \leq \mathrm{e}\kappa(L(\rho) + \ell w_{\max}) \leq \mathrm{e}(\alpha+1)\kappa L(\rho^*)$. The third equality follows from $\theta_\pi(\rho) = \sum_{e\in\pi}\theta_e(\rho)$, the fourth inequality from $\theta_\pi(\rho) \leq L(\rho)$ since $\rho(\pi) > 0$, and the last inequality from the hypothesis that $L(\rho) \leq \alpha L(\rho^*)$ and the fact that $\ell w_{\max} \leq L(\rho^*)$ because ρ^* is an unsplittable flow. Therefore, we conclude that $\mathbb{P}[\max_{\pi:\rho(\pi)>0}\{X_\pi\} \geq \mathrm{e}(\alpha+1)\kappa L(\rho^*)] \leq m\kappa^{-\mathrm{e}\kappa}$. In other words, the probability that the actual maximum delay caused by $\mathbf{p}$ exceeds the optimal maximum delay by a factor greater than $2\mathrm{e}(\alpha+1)\kappa$ is at most $m\kappa^{-\mathrm{e}\kappa}$. Therefore, for every $\kappa_0 \geq 2$, $\mathrm{SC}(\mathbf{p}) \leq \mathrm{e}(\alpha+1)L(\rho^*)\left(\kappa_0 + \sum_{k=\kappa_0}^{\infty} km k^{-\mathrm{e}k}\right) \leq \mathrm{e}(\alpha+1)L(\rho^*)\left(\kappa_0 + 2m\kappa_0^{-\mathrm{e}\kappa_0+1}\right)$. If $\kappa_0 = \frac{2\log m}{\log\log m}$, then $\kappa_0^{-\mathrm{e}\kappa_0+1} \leq m^{-1}$, $\forall m \geq 4$. Thus, $\mathrm{SC}(\mathbf{p}) \leq 2\mathrm{e}(\alpha+1)\left(\frac{\log m}{\log\log m} + 1\right)L(\rho^*)$. □

Bounding the Coordination Ratio. We finally show that the maximum expected delay of every NE is a good approximation of the optimal maximum latency. Then, we can apply Lemma 4 to bound the price of anarchy for our selfish routing game.

Lemma 5. *For any flow ρ corresponding to a NE $\mathbf{p}$, $L(\rho) \leq 3L(\rho^*)$.*

Proof. We actually show that $L(\rho) \leq L(\rho_f^*) + 2\ell w_{\max}$, where ρ_f^* is the optimal splittable flow. This implies the lemma because $L(\rho^*) \geq \max\{L(\rho_f^*), \ell w_{\max}\}$. The proof is based on Dorn's Theorem [5] establishing strong duality in quadratic programming. Let Q be the square matrix describing the number of edges shared by pairs of paths. Ie, $\forall \pi, \pi' \in \mathcal{P}, Q[\pi, \pi'] = |\pi \cap \pi'|$. Clearly Q is symmetric. We prove that it is also positive semi-definite (see full paper). $\forall \rho \in \mathbb{R}_{\geq 0}^{|\mathcal{P}|}$, the total latency of flow ρ is $C(\rho) = \rho^T Q \rho$. In addition, $(Q\rho)_\pi = \theta_\pi(\rho)$. Thus, the problem of computing a flow of value W_{tot} and minimum total latency is equivalent to computing the optimal solution of the following quadratic program (CP): $\min\{\rho^T Q\rho : \mathbf{1}^T\rho \geq W_{\mathrm{tot}}, \rho \geq \mathbf{0}\}$. Notice that no flow of value greater than W_{tot} can be optimal for CP. The Dorn' s dual of (CP) is (DP): $\max\{zW_{\mathrm{tot}} - \rho^T Q\rho : 2Q\rho \geq \mathbf{1}z, z \geq 0\}$ (see, [5], [1, Chapter 6]). We observe that any flow ρ which is feasible for (CP) can be regarded as a feasible solution for (DP) if we set $z = 2\theta^{\min}(\rho)$. The objective value of the solution $(\rho, 2\theta^{\min}(\rho))$ in (DP) is $2\theta^{\min}(\rho)W_{\mathrm{tot}} - C(\rho)$. Hence, an intuitive way of thinking about the dual program is that it asks for the flow ρ that maximizes the difference $2\theta^{\min}(\rho)W_{\mathrm{tot}} - C(\rho)$.

By Dorn's Theorem [5], since Q is symmetric and positive semi-definite and both (CP) and (DP) are feasible, they both have optimal solutions of the same objective value. In our case, the optimal splittable flow ρ_f^*, which is the optimal solution for (CP), corresponds to the solution $(\rho_f^*, 2\theta^{\min}(\rho_f^*))$, which is feasible for (DP). Moreover, for ρ_f^*, $L(\rho_f^*) = \theta^{\min}(\rho_f^*)$ and $C(\rho_f^*) = W_{\mathrm{tot}}L(\rho_f^*) = W_{\mathrm{tot}}\theta^{\min}(\rho_f^*)$ (see also the proof of Lemma 3). Thus, the objective value of the solution $(\rho_f^*, 2\theta^{\min}(\rho_f^*))$ in (DP) is exactly $C(\rho_f^*)$, and thus by Dorn's Theorem [5], $(\rho_f^*, 2\theta^{\min}(\rho_f^*))$ is optimal for (DP). For every feasible flow ρ for (CP), $(\rho, 2\theta^{\min}(\rho))$ is feasible for (DP). Since the optimal solution for (DP) has objective value $C(\rho_f^*)$, it must be $2\theta^{\min}(\rho)W_{\mathrm{tot}} - C(\rho) \leq C(\rho_f^*)$. If the flow ρ is a NE, then $L(\rho) \leq \theta^{\min}(\rho) + \ell w_{\max}$. Hence, it suffices to prove that $\theta^{\min}(\rho) \leq L(\rho_f^*) + \ell w_{\max}$. Since the average latency of ρ cannot exceed its maximum latency (see also the proof of Lemma 3), it is the case that $C(\rho) \leq L(\rho)W_{\mathrm{tot}} \leq$

$\theta^{\min}(\rho)\, W_{\mathrm{tot}} + \ell\, w_{\max}\, W_{\mathrm{tot}}$. Combining this with the last inequality, we obtain that $\theta^{\min}(\rho)\, W_{\mathrm{tot}} \leq C(\rho_f^*) + \ell\, w_{\max}\, W_{\mathrm{tot}}$. Using $C(\rho_f^*) = L(\rho_f^*)\, W_{\mathrm{tot}}$, we conclude that $\theta^{\min}(\rho) \leq L(\rho_f^*) + \ell\, w_{\max}$. □

The following theorem is an immediate consequence of Lemma 5 and Lemma 4.

Theorem 2. *The price of anarchy of any* ℓ*-layered network congestion game with resource delays equal to their loads, is at most* $8\,\mathrm{e}\,(\frac{\log m}{\log\log m} + 1)$.

References

1. Bazaraa M.S. and Sherali H.D. and Shetty C.M. *Nonlinear Programming: Theory and Algorithms (2nd edition).* John Wiley and Sons, Inc., 1993.
2. Beckmann M. and McGuire C.B. and Winsten C.B. *Studies in the Economics of Transportation.* Yale University Press, 1956.
3. Czumaj A. and Krysta P. and Vöcking B. Selfish traffic allocation for server farms. In *Proc. of the 34th ACM Symp. on Theory of Computing (STOC '02),* pages 287–296, 2002.
4. Czumaj A. and Vöcking B. Tight bounds for worst-case equilibria. In *Proc. of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA '02),* pages 413–420, 2002.
5. DornW.S. Duality in quadratic programming. *Quarterly of Applied Mathematics,* 18(2): 155–162, 1960.
6. Even-Dar E. and Kesselman A. and Mansour Y. Convergence time to nash equilibria. In *Proc. of the 30th International Colloquium on Automata, Languages and Programming (ICALP '03),* pages 502–513. Springer-Verlag, 2003.
7. Fabrikant A. and Papadimitriou C. and Talwar K. The complexity of pure nash equilibria. In *Proc. of the 36th ACM Symp. on Theory of Computing (STOC '04),* 2004.
8. Feldmann R. and Gairing M. and Lücking T.and Monien B. and Rode M. Nashification and the coordination ratio for a selfish routing game. In *Proc. of the 30th International Colloquium on Automata, Languages and Programming (ICALP '03),* pages 514–526. Springer-Verlag, 2003.
9. Fotakis D. and Kontogiannis S. and Koutsoupias E. and Mavronicolas M. and Spirakis P. The structure and complexity of nash equilibria for a selfish routing game. In *Proc. of the 29th International Colloquium on Automata, Languages and Programming (ICALP '02),* pages 123–134. Springer-Verlag, 2002.
10. Gairing M. and Lücking T. and Mavronicolas M. and Monien B. and Spirakis P. Extreme nash equilibria. In *8th Italian Conference on Theoretical Computer Science (ICTCS'03).* Springer-Verlag, 2003.
11. Hochbaum D. and Shmoys D. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.,* 17(3):539–551, 1988.
12. Hoeffding W. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association,* 58(301):13–30, 1963.
13. Koutsoupias E. and Mavronicolas M. and Spirakis P. Approximate equilibria and ball fusion. *ACM Transactions on Computer Systems,* 36:683–693, 2003.
14. Koutsoupias E. and Papadimitriou C. Worst-case equilibria. In *Proc. of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS '99),* volume LNCS 1563, pages 404–413. Springer-Verlag, 1999.
15. Lücking T. and Mavronicolas M. and Monien B. and Rode M. A New Model for Selfish Routing. In *Proc. of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS '04),* page to appear. Springer-Verlag, 2004.

16. Lücking T. and Mavronicolas M. and Monien B. and Rode M. and Spirakis P. and Vrto I. Which is the worst-case nash equilibrium? In *26th International Symposium on Mathematical Foundations of Computer Science (MFCS'03),* pages 551–561. Springer-Verlag, 2003.
17. Mavronicolas M. and Spirakis P. The price of selfish routing. In *Proc. of the 33rd ACM Symp. on Theory of Computing (STOC '01),* pages 510–519, 2001.
18. Milchtaich I. Congestion games with player-specific payoff functions. *Games and Economic Behavior,* 13:111–124, 1996.
19. Monderer D. and Shapley L. Potential games. *Games and Economic Behavior,* 14:124–143, 1996.
20. Papadimitriou C. and Steiglitz K. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., 1982.
21. Rosenthal R.W. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory,* 2:65–67, 1973.
22. Roughdarden T. and Tardos É. How bad is selfish routing? *J. Assoc. Comput. Mach.,* 49(2):236–259, 2002.
23. Roughgarden T. The price of anarchy is independent of the network topology. In *Proc. of the 34th ACM Symp. on Theory of Computing (STOC '02),* pages 428–437, 2002.

A General Technique for Managing Strings in Comparison-Driven Data Structures

Gianni Franceschini and Roberto Grossi

Dipartimento di Informatica, Università di Pisa
Largo Pontecorvo 1, 56127 Pisa, Italy

Abstract. This paper presents a general technique for optimally transforming any dynamic data structure $\mathcal{D}$ that operates on atomic and indivisible keys by constant-time comparisons, into a data structure $\mathcal{D}'$ that handles unbounded-length keys whose comparison cost is not a constant.

1 Introduction

Many applications manage keys that are arbitrarily long strings, such as multidimensional points, multiple-precision numbers, multi-key data, URL addresses, IP addresses, XML path strings, and that are modeled either as k-dimensional keys for a given positive integer $k > 1$, or as variable-length keys. In response to the increasing variety of these applications, the keys need to be maintained in sophisticated data structures. The comparison of any two keys is more realistically modeled as taking time proportional to their length, producing an undesirable slowdown factor in the complexity of the operations supported by the data structures.

More efficient *ad hoc* data structures have been designed to tackle this drawback. A first version of lexicographic or ternary search trees [6] dates back to [7] and is alternative to tries. Each node contains the i-th symbol of a k-dimensional key along with three branching pointers (left, middle, and right) for the three possible comparison outcomes [<, =, >] against that element. The dynamic balancing of ternary search trees was investigated with lexicographic D-trees [18], multidimensional B-trees [13], lexicographic globally biased trees [5], lexicographic splay trees [23], k-dimensional balanced binary search trees [11], and balanced binary search trees or kBB-trees [25]. Most of these data structures make use of sophisticated and tricky techniques to support search, insert, and delete of a key of length k in $O(k + \log n)$ time [5,11]. Some others support also split and concatenate operations in $O(k + \log n)$ time [13,18,23,25]. Moreover, other data structures allow for weighted keys (e.g., access frequencies) and the $\log n$ term in their time complexity is replaced by the logarithm of the ratio between the total weights and the weight of the key at hand [5,18,23,25].

This multitude of *ad hoc* data structures stems from the lack of a general data structural transformation from indivisible (i.e., constant-time comparable) keys to strings. Many searching data structures, such as AVL-trees, red-black trees[24], (a, b)-trees [15], weight-balanced BB[α]-trees [20], self-adjusting

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 606–617, 2004.

trees [23], and random search trees [22], etc., are currently available, with interesting combinatorial properties that make them attractive both from the theoretical and from the practical point of view. They are defined on a set of indivisible keys supporting an order relation. Searching and updating is driven by constant-time comparisons against the keys stored in them. One may wonder whether should data structuring designers reinvent the wheel in some cases or can they reuse the properties of these solutions.

A first step in reusing this body of knowledge and obtaining new data structures for managing strings has been presented theoretically in [12] and validated with experiments in [9]. It is based on the topology of the data structures by augmenting the nodes along the access paths to keys, each node with a pair of integers. By topology awareness, we mean that the designer must know the combinatorial properties and the invariants that are used to search and update the data structures, since he has to deal with all possible access paths to the same node. This depends on how the graph structure behind the data structure is maintained. While a general scheme is described for searching under this requirement, updating is discussed on an individual basis for the above reason. A random access path, for example, cannot be managed unless the possible access paths are limited in number. Also, adding an internal link may create many access paths to a given node. Related techniques, although not as general as that in [12], have been explored in [16,21] for specific data structures being extended to manage strings.

In this paper, we go on one step ahead. We completely drop any topological knowledge of the underlying data structures and still obtain the asymptotic bounds of previous results. The goal is to show that a more general transformation is indeed possible. In particular, we present a general technique which is capable of reusing many kinds of (heterogeneous) data structures so that they can operate on strings. We just require that each such data structure, say $\mathcal{D}$, is driven by constant-time comparisons among the keys (i.e., no hashing or bit manipulation of the keys) and that the insertion of a key into $\mathcal{D}$ identifies the predecessor or the successor of that key in $\mathcal{D}$. We are then able to transform $\mathcal{D}$ into a new data structure, $\mathcal{D}'$, storing n strings as keys while preserving all the nice features of $\mathcal{D}$. Asymptotically speaking, this transformation is costless. First, the space complexity of $\mathcal{D}'$ is $\mathcal{S}(n) + O(n)$, where $\mathcal{S}(n)$ denotes the space complexity of $\mathcal{D}$ (just store the pointers to strings, not the strings themselves). Second, each operation involving $O(1)$ strings taken from $\mathcal{D}'$ requires $O(\mathcal{T}(n))$ time, where $\mathcal{T}(n)$ denotes the time complexity of the corresponding operation originally supported in $\mathcal{D}$. Third, each operation involving a string y *not* stored in $\mathcal{D}'$ takes $O(\mathcal{T}(n) + |y|)$ time, where $|y|$ denotes the length of y.

The field of interest for our technique is especially for sub-logarithmic costs, when $\mathcal{T}(n) = o(\log n)$: either in the worst case (e.g., $\mathcal{D}$ is a finger search tree), in amortized sense (e.g., $\mathcal{D}$ is a self-adjusting tree) or with high probability (e.g., $\mathcal{D}$ is a treap), when considering access frequencies in the analysis.

Our technique exploits the many properties of one-dimensional searching, and combines in a variety of novel ways techniques from data structures and string

algorithms. Formally, we manage input strings $x_1, x_2, \ldots, x_n$ of total length $M = \sum_{i=1}^{n} |x_i|$. Each string x_i is a sequence of $|x_i|$ symbols drawn from a potentially unbounded alphabet Σ, and the last symbol of x_i is a special endmarker less than any symbol in Σ. In order to compare two strings x and y, it is useful to employ the length of their longest common prefix, defined as $lcp(x, y) = \max\{\ell \geq 0 \mid x[1..\ell] = y[1..\ell]\}$ (here, $\ell = 0$ denotes empty prefixes). Given that length, we can compare x and y in constant time by simply comparing their first mismatching symbol, which is at position $1 + lcp(x, y)$ in x and y.

With this fact in mind, we can use the underlying data structure $\mathcal{D}$ as a black box. We use simple properties of strings and introduce a powerful oracle for string comparisons that extends the functionalities of the Dietz-Sleator list [10], which is able to maintain order information in a dynamic list (shortly, DS list). We call the resulting structure a DS_{lcp} list, which stores the sorted input strings in $O(n)$ space, and allows us to find the length of the longest common prefix of any two strings stored in the DS_{lcp} list, in constant time. [1] We can maintain dynamically a DS_{lcp} list in constant time per operation (see Section 2.1 for the operations thus supported) by using a simple but key idea in a restricted dynamic version of the range minima query problem [4]. Note that otherwise would not be possible to achieve constant time per operation in the fully dynamic version of this problem as we can perform sorting with it.

Using our general technique, we obtain previous theoretical bounds in an even simpler way. We also obtain new results on searching and sorting strings. For example, we can perform suffix sorting, a crucial step in text indexing and in block sorting compression based on the Burrows-Wheeler transform, in $O\big(n + \sum_{i=1}^{n} \mathcal{T}(i)\big)$ time, also for unbounded alphabet Σ. For this alphabet, this appears to be a new result; the known literature reports the time complexity of $\Theta(n \log n)$ in the worst case as it tantamounts to sorting the alphabet symbols (a linear time bound is possible in some special cases). Using our result, we can perform suffix sorting in $O\big(n \log(F/n)\big)$ time, where $0 \leq F \leq n(n-1)/2$ is the number of inversions. This new result is a simple consequence of our result, when applied to the techniques for one-dimensional keys given, for example, in [19]. Another example of use is that of storing implicitly the root-to-nodes paths in a tree as strings, so that we can support dynamic lowest common ancestor *(lca)* queries in constant time, where the update operations involve adding/removing leaves. In previous work, this result has been obtained with a special data structure based upon a more sophisticated solution treating also insertions that split arcs [8]. We obtain a simple method for a restricted version of the problem.

As a final remark for our technique, we do not claim that it is as amenable to implementation in a practical setting such as the technique in [9,12]. Nevertheless, we believe that our general technique may be helpful in the theoretical setting for providing an immediate benchmark to the data structuring designer.

[1] When $\mathcal{T}(n) = \Omega(\log n)$, there are alternative techniques, e.g., using compacted tries and dynamic lowest common ancestor queries [8], as $O\big(\mathcal{T}(n) + |y|\big)$ absorbs the cost of inserting a string y into the trie, which is either $O(|y| \log |\Sigma|)$ or $O(|y| + \log n)$ in the worst case, as is the case for unbounded Σ in the comparison model.

```
1: m ← DS_lcp(best.friend, x)
2: IF m ≥ best.lcp THEN
3:    m ← best.lcp
4:    WHILE x[m + 1] = y[m + 1] DO m ← m + 1
5:    best.friend ← x
6:    best.lcp ← m
7: RETURN m
```

Fig. 1. Code for computing $lcp(x, y)$ values on the fly.

When inventing a new data structure for strings, the designer can easily realize whether it compares favorably to the known data structures, whose functionalities can be smoothly extended as a black box to strings without giving up their structural and topological properties.

2 The General Technique for Strings

We now describe our technique. The operations supported by the DS_{lcp} list are listed in Section 2.1, whose implementation is discussed later on, in Section 4. The fast computation on the fly of *lcp* values is presented in Section 2.2. The use of the latter two tools in our technique is shown in Section 2.3. We recall that, for any two strings x and y, we have $x \leq y$ in lexicographic order if and only if $x[\ell + 1] < y[\ell + 1]$, where $\ell = lcp(x, y)$. Here is why we center our discussion around the efficient computation of *lcp* values.

2.1 The DS_{lcp} List

The DS_{lcp} list stores a sequence of strings $x_1, x_2, \ldots, x_n$ in lexicographic order, each string is of unbounded-length and is referenced by a constant-space pointer (e.g., `char *p` in C language). A DS_{lcp} list L supports the following operations:

- Query $DS_{lcp}(x_p, x_q)$ in L, returning the value of $lcp(x_p, x_q)$, for any pair of strings x_i and x_j stored in L.
- Insert y in a position between two consecutive keys x_{k-1} and x_k in L. Requirements: $x_{k-1} \leq y \leq x_k$ holds, and $lcp(x_{k-1}, y)$ and $lcp(y, x_k)$ are known.
- Remove string x_i from its position in L.

Theorem 1. *A DS_{lcp} list L can be implemented in $O(n)$ space, so that querying for lcp values, inserting keys into L and deleting keys from L can be supported in constant time per operation, in the worst case.*

2.2 Computing *lcp* Values on the Fly

The DS_{lcp} list L is a valid tool to dynamically compute *lcp* values for the strings stored in L. We now examine the situation in which we have to compare a string $y \notin L$ against an arbitrary choice of strings $x \in L$. We put ourselves in the worst

situation, namely, the choice of x is unpredictable from our point of view. Even in this case, we can still compute $lcp(x, y)$ efficiently. We implicitly assume that the empty string is kept in L as the least string.

We employ two global variables, *best.friend* and *best.lcp,* which are initialized to the empty string and to 0, respectively. During the computation, they satisfy the invariant that, among all the strings in L compared so far against y, the one pointed by *best.friend* gives the maximum *lcp* value, and that value is *best.lcp.* We now have to compare y against x, following the simple algorithm shown in Fig. 1. Using L, we can compute $m = DS_{lcp}(best.friend, x)$, since both strings are in L. If $m < best.lcp$, we can infer that $lcp(x, y) = m$ and return that value. Otherwise, we may possibly extend the number of matched characters in y storing it into *best.lcp,* thus finding a new *best.friend.* It's a straightforward task to prove the correctness of the invariant (note that it works also when $x = best.friend$). Although the code can be improved by splitting the case $m \geq best.lcp$ of line 2 into two cases, it does not improve the asymptotic complexity.

Let's take an arbitrary operation that accesses some of the strings in L in arbitrary order, say, $x'_1, x'_2, \ldots, x'_g$ (these strings are not necessarily distinct and/or sorted). For a given string $y \notin L$, the total cost of computing $lcp(x'_1, y)$, $lcp(x'_2, y)$, $\ldots$, $lcp(x'_g, y)$ on the fly with the function shown in Fig. 1 can be accounted as follows. The cost of the function is constant unless we enter the body of the while loop at line 4, to match further characters while increasing the value of m. We can therefore restrict our analysis to the strings $x'_i \in L$ that cause the execution of the body of that while loop. Let's take the kth such string, and let m_k be the value of m at line 6. Note that the body of the while loop at line 4 is executed $m_k - m_{k-1}$ times (precisely, this is true since $best.lcp = m_{k-1}$, where $m_0 = 0$). Thus the cost of computing the *lcp* value for such a string is $O(1 + m_k - m_{k-1})$.

We can sum up all the costs. The strings not entering the while loop contribute each for a constant number of steps; the others contribute for $O(1 + m_k - m_{k-1})$ steps. As a result, we obtain a total cost of $O(g + \sum_k (m_k - m_{k-1})) = O(g + |y|)$ time, since $m_k \geq m_{k-1}$ and $\sum_k (m_k - m_{k-1})$ is upper bounded by the length of the longest matched prefix of y, which is in turn at most $|y|$.

Lemma 1. *The computation on the fly of any sequence of g lcp values involving a given string $y \notin L$ and some strings in L can be done in $O(g + |y|)$ time.*

Note that, if y were in L, we could obtain a bound of $O(g)$ in Lemma 1. Instead, $y \notin L$, and L helps us to reduce the cost from $O(g \cdot |y|)$ to $O(g + |y|)$.

2.3 Exploiting *lcp* Values in Comparison-Driven Data Structures

We now finalize the description of our general technique, leaving that of the implementation of the DS_{lcp} list L to Section 4.

Theorem 2. *Let $\mathcal{D}$ be a comparison-driven data structure such that the insertion of a key into $\mathcal{D}$ identifies the predecessor or the successor of that key in $\mathcal{D}$. Then, $\mathcal{D}$ can be transformed into a data structure $\mathcal{D}'$ for strings such that*

- *the space complexity of* $\mathcal{D}'$ *is* $\mathcal{S}(n) + O(n)$ *for storing* n *strings as keys (just store the references to strings, not the strings themselves), where* $\mathcal{S}(n)$ *denotes the space complexity of* $\mathcal{D}$*;*
- *each operation involving* $O(1)$ *strings in* $\mathcal{D}'$ *takes* $O(\mathcal{T}(n))$ *time, where* $\mathcal{T}(n)$ *denotes the time complexity of the corresponding operation originally supported in* $\mathcal{D}$*;*
- *each operation involving a string* y not *stored in* $\mathcal{D}'$ *takes* $O(\mathcal{T}(n)+|y|)$ *time, where* $|y|$ *denotes the length of* y*.*

Proof. The new data structure $\mathcal{D}'$ is made up of the original data structure $\mathcal{D}$ along with the DS_{lcp} list L of Section 2.1, and uses the computation on the fly described in Section 2.2. The additional space is that of L, namely, $O(n)$ by Theorem 1.

For the cost of the operations, consider first the insertion of a key y into $\mathcal{D}'$. We run the insertion algorithm supported by $\mathcal{D}$ as a black box. When this algorithm requires to compare y with a string x already in $\mathcal{D}'$, we proceed as in Section 2.2. By hypothesis, the algorithm determines also the predecessor or the successor of y. In $O(|y|)$ time, we can compute (if not yet determined) their *lcp* values, which are needed to insert y into L. The final cost is that of Lemma 1, where $g \leq \mathcal{T}(n)$. The other operations supported by $\mathcal{D}'$ have a similar analysis. If they require comparisons that involve strings already stored in $\mathcal{D}'$, each comparison can be clearly performed in constant time by Theorem 1. Hence their cost is just $O(\mathcal{T}(n))$ since $g \leq \mathcal{T}(n)$.

3 Some Applications

3.1 Suffix Sorting

As previously mentioned, suffix sorting is very useful in compressing, with block sorting methods and Burrows-Wheeler transform, and in text indexing, with suffix arrays [17]. The problem is that of sorting lexicographically the suffixes of an input string s of length n. Let $s_1, s_2, \ldots, s_n$ denote the suffixes of s, where $s_i \equiv s[i..n]$ corresponds to the ith suffix in s. We show how to apply the ideas behind Theorem 2 to the suffix sorting problem. We recall that comparing two suffixes s_i and s_j takes constant time if we known their value of $lcp(s_i, s_j)$. Again, we focus our presentation on the *lcp* computation.

We first need to augment the DS_{lcp} list L of Theorem 2 with suffix links. Let's take a snapshot of L at the end of the suffix sorting. A suffix link $sl(s_r)$ points to s_{r+1} in L, for $1 \leq r < n$. During the intermediate steps, we insert the suffixes $s_1, s_2, \ldots, s_n$ into $\mathcal{D}'$, in this order.

Before inserting s_i into $\mathcal{D}'$, the pointers $sl(s_j)$ are defined for $1 \leq j < i-1$. The current entry in L is s_{i-1}, for which we know its predecessor, x_{i-1}, and its successor, y_{i-1}, in L. Note that we cannot exploit $sl(s_{i-1})$ as s_i has still to be inserted. We also know $lcp(x_{i-1}, s_{i-1})$ and $lcp(s_{i-1}, y_{i-1})$. This invariant is trivially satisfied before inserting the first suffix, s_1 (here, $x_0 = s_0 = y_0 =$ empty string).

We use induction to describe the step for s_i. It suffices to show how to compute $lcp(z, s_i)$ on the fly, for $z \in \{s_j \mid j < i\}$, with reference to the code shown in Fig. 1. Assuming that $lcp(x_{i-1}, s_{i-1}) \geq lcp(s_{i-1}, y_{i-1})$ without loss of generality, we set $best.friend = sl(x_{i-1})$ and $best.lcp = \max\{0, best.lcp - 1\}$, before executing the sequence of calls to the function in Fig. 1 related to the insertion of s_i. When the insertion completes its task, we know the predecessor, x_i, and the successor, y_i, of s_i in L. We also know their *lcp* values. To maintain the invariant for the next step, we need to pose $sl(s_{i-1}) = s_i$.

Theorem 3. *Let $\mathcal{D}'$ be a data structure for managing strings obtained following Theorem 2. Then, all the suffixes of an input string of length n can be inserted into $\mathcal{D}'$, in space $O(n)$ and time*

$$O\left(n + \sum_{i=1}^{n} \mathcal{T}(i)\right),$$

where $\mathcal{T}(\cdot)$ denotes the time complexity of the insert operation in the original data structure $\mathcal{D}$ from which $\mathcal{D}'$ has been obtained. The suffixes can be retrieved in lexicographic order in linear time.

Theorem 3 provides an adaptive bound for input strings whose symbols have some presortedness. There are cases in which $\sum_{i=1}^{n} \mathcal{T}(i) = o(n \log n)$ for arbitrary alphabets Σ, whereas the known literature for suffix sorting reports the time complexity of $\Theta(n \log n)$ in the worst case as we are essentially sorting the alphabet symbols (a linear time bound is possible in special cases). One extreme example is an input string with all distinct characters in increasing order, for which the bound of Theorem 3 is $O(n)$. In general, we can perform suffix sorting in $O(n \log(F/n))$ time, where $0 \leq F \leq n(n-1)/2$ is the number of inversions. We obtain a new result that reuses techniques for one-dimensional keys given, for example, in [19].

3.2 Dynamic Lowest Common Ancestor *(lca)*

The lowest common ancestor problem for a tree is at the heart of several algorithms [4,14]. We consider here the dynamic version in which insertions add new leaves as children to existing nodes and deletions remove leaves. The more general (and complicated) case of splitting an arc by inserting a node in the middle of the arc is treated in [8].

We maintain the tree as an Euler tour, which induces an implicit lexicographic order on the nodes. Namely, if a node is the ith child of its parent, the implicit label of the node is i. The root has label 0. (These labels are mentioned only for the purpose of presentation.) The implicit string associated with a node is the sequence of implicit labels obtained in the path from the root to that node plus an endmarker that is different for each string (also when the string is duplicated; see the discussion below on insertion). Given any two nodes, the *lcp* value of their implicit strings gives the string implicitly represented by their *lca*. We

maintain the Euler tour with a DS_{lcp} list L in $O(n)$ space (see Section 2.1), where n is the number of nodes (the strings are implicit and thus do not need to be stored). We also maintain the dynamic data structure in [1] to find the level ancestor of a node in constant time.

Given any two nodes u and v, we compute $lca(u, v)$ in constant time as follows. We first find $d = lcp(s_u, s_v)$ using L, where s_u and s_v are the implicit strings associated with u and v, respectively. We then identify their ancestor at depth d using a level ancestor query.

Inserting a new leaf duplicates the implicit string s of the leaf's parent, and puts the implicit string of the leaf between the two copies of s thus produced in the Euler tour. Note that we satisfy the requirements described in Section 2.1 for the insert, as we know their *lcp* values. By Theorem 1, this takes $O(1)$ time. For a richer repertoire of supported operations in constant time, we refer to [8].

Theorem 4. *The dynamic lowest common ancestor problem for a tree, in which leaves are inserted or removed, can be solved in* $O(1)$ *time per operation in the worst case, using a* DS_{lcp} *list and the constant-time dynamic level ancestor.*

4 Implementation of the DS_{lcp} List

We describe how to prove Theorem 1, implementing the DS_{lcp} list L introduced in Section 2.1. We use the fact that $lcp(x_p, x_q) = \min\{lcp(x_{k-1}, x_k) \mid p < k \leq q\}$ since the strings $x_1, x_2, \ldots, x_n$ in L are in lexicographic order (here, $p < q$). In other words, storing only the *lcp* values between each key x_k and its predecessor in L, we can answer arbitrary *lcp* queries using the so-called range minima query problem [4]. The input is an unordered set of m entries (the $lcp(x_{k-1}, x_k)$s) and, for any given range $[i..j]$, we want to report the minimum among the entries from i to j (where $i = p + 1$ and $j = q$ for $lcp(x_p, x_q)$).

We are interested in discussing the dynamic version of the problem. In its general form, this is equivalent to sorting. Fortunately, we can attain constant time per operation since we impose the additional constraint that the set of entries can only vary *monotonically*. Namely, an entry e changes by replacing it with two entries e_1 and e_2, such that $e_1 \geq e_2 = e$ or $e_2 \geq e_1 = e$. This constraint is not artificial, being dictated by the requirements listed in Section 2.1 when inserting string y between x_{k-1} and x_k. A moment of reflection shows that both $e_1 \equiv lcp(x_{k-1}, y)$ and $e_2 \equiv lcp(y, x_k)$ are greater than or equal to $e \equiv lcp(x_{k-1}, x_k)$ (the entry to be changed), and at least one of them equals e.

Going straight to the benefit of monotonicity in a dynamic setting, consider the problem of maintaining the prefix minima $p_1, p_2, \ldots, p_m$ for the m entries (treating suffix minima is analogous). When inserting y as above, *just two* prefix minima can change, whereas they can all change without the monotonicity. Namely, $p_y = \min\{p_{k-1}, e_1\}$ and $p_k = \min\{p_y, e\}$, assuming $e_1 \geq e_2 = e$ and letting p_y be the prefix minimum for the entry e_1 associated with y. We use this as a key observation to obtain constant-time complexity.

We focus on insertions as deletions are weak and can be treated with partial rebuilding techniques (deletions replace two consecutive entries with the

smallest of the two and so do not change the range minima of the remaining entries). Note that the insertion of y can be viewed as either the insertion of entry e_1 to the left of entry e (when $e_1 \geq e_2 = e$) or the insertion of e_2 to the right of e (when $e_2 \geq e_1 = e$).

For implementing the DS_{lcp} list we adopt a two-level scheme. We introduce the upper level consisting of the *main tree* in Section 4.1, and the lower level populated of *micro trees* in Section 4.2. We sketch the method for combining the two levels in Section 4.3. The net result is a generalization of the structure of Dietz and Sleator that works for multidimensional keys, without relying on the well-known algorithm of Willard [26] to maintain order in a dense file (avoiding Willard's for the amortized case has been suggested in [3]). When treating information that can be represented with $O(\log n)$ bits, we will make use of table lookups in $O(1)$ time. The reader may verify that we also use basic ideas from previous work [2,4,14].

4.1 Main Tree

For the basic shape of the main tree we follow the approach of [2]. The main tree has m leaves, all on the same level (identified as level 0), each leaf containing one entry of our range minima problem. The *weight* $w(v)$ a node v is *(i)* the number of its children (leaves), if v is on level 1 *(ii)* the sum of the weights of its children, if v is on a level $l > 1$. Let $b > 4$ the *branching parameter*. We maintain the following constraints on the weight of a node v on a level l.

1. If $l = 1$, $b \leq w(v) \leq 2b - 1$.
2. If $l > 1$, $w(v) < 2b^l$.
3. If $l > 1$ and v is not the root of the tree, $w(v) > \frac{1}{2}b^l$.

From the above constraints it follows that each node on a level $l > 1$ in the main tree has between $b/4$ and $4b$ children (with the exception of the root that can have a minimum of two children). From this we can easily conclude that the height of the main tree is $h = O(\log_b m) = 0(\log m)$.

When a new entry is inserted as a new leaf v in the main tree, any ancestor u of v that does not respect the weight constraint is split into two new nodes and the new child is inserted in the parent of u (unless u is the root, in that case a new root is created). That rebalancing method has an important property.

Lemma 2 ([2]). *After a split of a node u on level $l > 1$ into nodes u' and u'', at least $b^l/2$ inserts have to be performed below u' (or u'') before splitting again.*

The nodes of the main tree are augmented with two secondary structures.

The former secondary structure is devoted to *lca* queries. Each internal node u is associated with a numeric identifier $sib(u)$ representing its position among its siblings; since the maximum number of children of a node is a constant, we need only a constant number of bits, say c_b, to store each identifier. Each leaf v is associated with two vectors, $\mathcal{P}_v$ and $\mathcal{A}_v$. Let u_i be the i-th ancestor of v (starting from the root). The i-th location of $\mathcal{P}_v$ contains the identifier $sib(u_i)$ whereas

the i-th location of $\mathcal{A}_v$ contains a pointer to u_i. Note that $\mathcal{P}_v$ occupies $h \cdot c_b = O(\log m)$ bits. These auxiliary vectors are used to find the *lca* between any two leaves v', v'' of the main tree in constant time. First, we find $j = lcp(\mathcal{P}_{v'}, \mathcal{P}_{v''})$ by table lookups; then, we use the pointer in $\mathcal{A}_v[j]$ to access the node.

The latter secondary structure is devoted to maintain some range minima. Each internal node u is associated with a doubly linked list $\mathcal{E}_u$ that contains the copies of the entries in the descendant leaves of u. The order of the copies in $\mathcal{E}_u$ is identical to that in the leaves (i.e., the lexicographical order in which the strings are maintained). As previously mentioned, we maintain the prefix minima and the suffix minima in $\mathcal{E}_u$. Then, we associate with each leaf v a vector $\mathcal{C}_v$ containing pointers to all the copies of the entry in v, each copy stored in the doubly linked lists of v's ancestors.

Because of the redundancy of information, the total space occupied by the main tree is $O(m \log m)$ but now we are able to answer a general range minima query for an interval $[i \,..\, j]$ in constant time. We first find the lowest common ancestor u of the leaves v_i and v_j corresponding to the ith and the jth entries, respectively. Let u_i be the child of u leading to v_i and u_j that leading to v_j (they must exist and we can use $\mathcal{P}_{v_i}$ and $\mathcal{P}_{v_j}$ for this task). We access the copies of the entries of i and j in $\mathcal{E}_{u_i}$ and $\mathcal{E}_{u_j}$, respectively, using $\mathcal{C}_{v_i}$ and $\mathcal{C}_{v_j}$. We then take the suffix minimum anchored in i for $\mathcal{E}_{u_i}$, and the prefix minimum anchored in j for $\mathcal{E}_{u_j}$. We also take the minima in the siblings between u_i and u_j (excluded). The minimum among these $O(1)$ minima is then the answer to our query for interval $[i \,..\, j]$.

Lemma 3. *The main tree for m entries occupies $O(m \log m)$ space, and support range minima queries in $O(1)$ time and monotone updates in $O(\log m)$ time.*

It remains to see how the tree can be updated. We are going to give a "big picture" of the techniques used, leaving the details of the most complicated aspects to the full version of the paper. We already said that deletions can be treated lazily with usual partial rebuilding techniques. We follow the same approach for treating the growth of the height h of the main tree and the subsequent variations of its two secondary structures, $\mathcal{P}$, $\mathcal{A}$, $\mathcal{E}$, and $\mathcal{C}$. From now on, let's assume w.l.o.g. that the insertions do not increase the height of the main tree.

When a new string is inserted, we know by hypothesis a pointer to its predecessor (or successor) in the lexicographical order and the pointer to the leaf v of the main tree that receives a new sibling v' and contains the entry (*lcp* value) to be changed. The creation and initialization of the vectors associated with the new leaf v' can be obviously done in $O(\log m)$ time. Then we must propagate the insertion of the new entry in v' to its ancestors. Let u be one of these ancestors. We insert the entry into its position in $\mathcal{E}_u$, using $\mathcal{C}_v$, which is correctly set (and useful for setting $\mathcal{C}_{v'}$). As emphasized at the beginning of Section 4, the monotonicity guarantees that the only prefix minima changing are constant in number and near to the new entry (an analogous situation holds for the suffix minima). As long as we do not need to split an ancestor, we can therefore perform this update in constant time per ancestor.

If an ancestor u at level l needs to split in two new nodesu' and u'' so as to maintain the invariants on the weights, we need to recalculate $O(|\mathscr{E}_u|) = O(b^l)$ values of prefix and suffix minima. By Lemma 2 we can immediately conclude that the time needed to split u is $O(1)$ in amortized sense. A lazy approach to the construction of the lists $\mathscr{E}_{u'}$ and $\mathscr{E}_{u''}$ will lead to the desired worst case constant time complexity for the splitting of an internal node. This construction is fairly technical (e.g., see [2]) and we will detail it in the full paper.

4.2 Micro Trees for Indirection

We employ micro trees for providing a level of indirection to reduce space and update time in the main tree of Section 4.1. Each micro tree satisfies the invariants 1–3 of the main tree, except that all the nodes contain $O(\log n)$ entries, with a fan out of $O(\log n)$. We guarantee that a micro tree stores $\Theta(\log^2 n)$ entries in two levels (its height is $h = 2$). We fill its nodes by starting from a doubly linked list of its $\Theta(\log^2 n)$ entries. We partition it into sublists of size $O(\log n)$, which are the leaves. Then, we take the first and the last entry in each sublist, and copy these two entries in a new sublist of size $O(\log n)$, so as to form the root. It's not difficult to maintain the weighted median entry of each sublist in $O(1)$ time per operation. This is useful to split a sublist (node) into two sublists (nodes) of equal size in $O(1)$ time (we can incrementally scan them by $O(1)$ entries at a time and find the new weighted median before they split again).

We have a secondary structure in the nodes of the micro trees, to locally support range minima, split and insert in $O(1)$ time. Each sublist is associated with a Cartesian tree [4,14]. The root is the minimum entry and its left (right) subtree recursively represents the entries to the left (right). The base case corresponds to the empty set which is represented by the null pointer. The observation in [4] is that the range minimum from entry i to entry j is given by the entry represented by $lca(i, j)$ in the Cartesian tree. We encode all this information in $O(\log n)$ bits so that it can be manipulated in $O(1)$ time with table lookups. Note that inserting entries monotonically guarantees that these entries are inserted as leaves, so we know the position of insertion in constant time. Again, we will detail it in the full paper.

4.3 Implementing the Operations

Our high-level scheme is similar to that in [10]. The main tree has $m = O(n/\log^2 n)$ leaves. Each leaf is associated with a distinct micro tree, so that the concatenation of the micro trees gives the order kept in the DS_{lcp} list. Each micro tree contributes to main tree with its leftmost and rightmost entries (actually, the range minima of its two extremal entries). A micro tree is ready to split, when the number of its keys is at least 2/3 of the maximum allowed. The ready micro trees are kept in a queue sorted by size. We take the largest such tree, split it in $O(1)$ time and insert two new entries in the main tree. However, we perform incrementally and lazily the $O(\log n)$ steps for the insertion (Lemma 3)

of these two entries. At any time only one update is pending in the main tree by an argument similar to that in [10].

References

1. S. Alstrup, J. Holm. Improved algorithms for finding level ancestors in dynamic trees. *ICALP,* 73–84, 2000.
2. L. Arge, J. S. Vitter. Optimal external memory interval management. *SIAM Journal on Computing,* 32:1488–1508, 2003.
3. M. A. Bender, R. Cole, E. M. Demaine, M. Farach-Colton, J. Zito. Two simplified algorithms for maintaining order in a list. In *ESA,* 2002.
4. M. A. Bender and M. Farach-Colton. The LCA problem revisited. *LATIN,* 88–94, 2000.
5. S.W. Bent, D.D. Sleator and R.E. Tarjan. Biased search trees. *SIAM Journal on Computing* 14 (1985), 545–568.
6. J.L. Bentley and R. Sedgewick. Fast algorithms for sorting and searching strings. In *SODA* (1997), pages 360–369.
7. H.A. Clampett. Randomized binary searching with the tree structures. *Communications of the ACM* 7 (1964), 163–165.
8. R. Cole and R. Hariharan. Dynamic LCA queries on trees. *SODA,* 235–244, 1999.
9. P. Crescenzi, R. Grossi, and G.F. Italiano. Search data structures for skewed strings. *WEA,* 2003.
10. P.F. Dietz and D.D. Sleator. Two algorithms for maintaining order in a list. *STOC,* 365–372, 1987.
11. T.F. Gonzalez. The on-line *d*-dimensional dictionary problem. *SODA,* 376–385, 1992.
12. R Grossi and G.F. Italiano. Efficient techniques for maintaining multidimensional keys in linked data structures (extended abstract). *ICALP,* 372–383, 1999.
13. R.H. Gueting and H.-P. Kriegel. Multidimensional B-tree: An efficient dynamic file structure for exact match queries. *10th GI Annual Conference,* 375–388, 1980.
14. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal of Computing,* 13:338–355, 1984.
15. S. Huddleston and K. Mehlhorn. A new data structure for representing sorted lists. *Acta Informatica* 17 (1982), 157–184.
16. R.W. Irving and L. Love. The suffix binary search tree and suffix AVL tree. In *Journal of Discrete Algorithms,* 387–408, 2003.
17. U. Manber and E.W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing* 22 (1993), 935–948.
18. K. Mehlhorn. Dynamic binary search. *SIAM J. on Computing* 8 (1979), 175–198.
19. K. Mehlhorn. *Data structures and algorithms: 1. Searching and sorting,* 1984.
20. J. Nievergelt and E.M. Reingold. Binary search trees of bounded balance. *SIAM Journal on Computing* 2 (1973), 33–43.
21. S. Roura. Digital access to comparison-based tree data structures and algorithms. *Journal of Algorithms,* 40:1–23, 2001.
22. R. Seidel and C.R. Aragon. Randomized search trees. *Algorithmica,* 1996, 464–497.
23. D. D. Sleator and R.E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM* 32 (1985), 652–686.
24. R.E. Tarjan. *Data structures and network algorithms,* SIAM (1983).
25. V. K. Vaishnavi. On *k*-dimensional balanced binary trees. *JCSS* 52 (1996), 328–348.
26. D.E. Willard. A density control algorithm for doing insertions and deletions in a sequentially ordered file in good worst-case time. *Informat. and Comput.,* 1992.

Greedy Regular Expression Matching

Alain Frisch[1,2]* and Luca Cardelli[3]

[1] École Normale Supérieure (Paris)
[2] École Nationale Supérieure des Télécommunications (Paris)
[3] Microsoft Research

Abstract. This paper studies the problem of matching sequences against regular expressions in order to produce structured values.

1 Introduction

Regular expressions play a key role in XML [W3C00]. They are used in XML schema languages (DTD, XML-Schema [W3C01], Relax-NG, …) to constrain the possible sequences of children of an element. They naturally lead to the introduction of *regular expression types* and *regular expression patterns* in XML-oriented functional languages (XDUCE [HVP00,HP03,Hos01], XQuery [BCF+03b], CDuce [BCF03a]). These works introduce new kinds of questions and give results in the theory of regular expression and regular (tree) languages, such as efficient implementation of inclusion checking and boolean operations, type inference for pattern matching, checking of ambiguity in patterns [Hos03], compilation and optimization of pattern matching [Lev03,Fri04], etc…

Our work is a preliminary step in introducing similar ideas to imperative or object-oriented languages. While XTATIC [GP03] uses a uniform representation of sequences, we want to represent them with structured data constructions that provide more efficient representation and access. As in XDUCE, our types are regular expressions: we use $\times$, $+$, *, ε to denote concatenation, alternation, Kleene star and the singleton set containing the empty sequence. But our types describe not only a set of possible sequences, but also a concrete structured representation of values. As in the Xen language [MS03], we map structural types to native .NET CLR [ECM02] types, however we define subtyping on the basis of *flattened* structures, in order to support natural semantic properties of regular language inclusion. For instance, $(\texttt{int} \times \texttt{int})$ is a set-theoretic subtype of $\texttt{int}^*$, but we need a coercion to use a value of the former where a value of the latter is expected, because the runtime representations of the two types are different. Such a coercion can always be decomposed (at least conceptually) in two phases: flatten the value of the subtype to a uniform representation, and then match that flat sequence against the super type. The matching process is a generalization of pattern matching in the sense of XDUCE [HP01].

* This work was supported by an internship at Microsoft Research.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 618–629, 2004.

This paper does not propose a language design. Instead, we study the theoretical problem of matching a flat sequence against a type (regular expression); the result of the process is a structured value of the given type. In doing so, one must pay attention to ambiguity in matching. Our contributions, thus, are in noticing that: (1) A disambiguated result of parsing can be presented as a data structure in a separate type system that does not contain ambiguities. (2) There are problematic cases in parsing values of star types that need to be disambiguated (Prop. 1). (3) The disambiguation strategy used in XDUCE and CDuce pattern matching can be characterized mathematically by what we call greedy regular expression matching. (4) There is a linear time algorithm for the greedy matching.

There is a rich literature on efficient implementation of regular expression pattern matching [Lau01] [Kea91,DF00]. There is a folklore problem with expression-based implementations of regular expression matching: they don't handle correctly the case of a regular expression t^* when t accepts the empty word. Indeed, an algorithm that would naively follow the expansion $t^* \rightsquigarrow (t \times t^*) + \varepsilon$ could enter an infinite loop. Harper [Har99] and Kearns [Kea91] propose to keep the naïve algorithm, but to use a first pass to rewrite the regular expressions so as the remove the problematic cases. For instance, let us consider the regular expression $t = (a^* \times b^*)^*$. We could rewrite it as $t' = ((a \times a^*) \times b^* + (b \times b^*))^*$. In general, the size of the rewritten expression may be exponential in the size of the original expression. Moreover, changing the regular expression changes the type of the resulting values, and the interaction with the disambiguation policy (see below) is not trivial. Therefore, we do not want to rewrite the regular expressions. Another approach is to patch the naive recognition algorithm to detect precisely the problematic case and cut the infinite loop [Xi01]. This is an *ad hoc* way to define the greedy semantics in presence of problematic regular expressions.

Our approach is different since we want to axiomatize abstractly the disambiguation policy, without providing an explicit matching algorithm. We identify three notions of problematic words, regular expressions, and values (which represent the ways to match words), relate these three notions, and propose matching algorithms to deal with the problematic case.

2 Notations

Sequences. For any set X, we write X^* for the set of finite sequences over X. Such a sequence is written $[x_1; \ldots; x_n]$. The empty sequence is []. We write $x :: s$ for the sequence obtained by prepending x in front of s and $s :: x$ for the sequence obtained by appending x after s. If s_1 and s_2 are sequences over X, we define $s_1 @ s_2$ as their concatenation. We extend these notations to subsets of X^* with $x :: X_1 = \{x :: s \mid s \in X_1\}$ and $X_1 @ X_2 = \{s_1 @ s_2 \mid s_i \in X_i\}$.

Symbols, words. We assume to be given a fixed alphabet Σ, whose elements are called symbols (they will be denoted with $c, c_1, \ldots$). Elements of Σ^* are called words. They will be denoted with w, $w_1, w', \ldots$

Types. The set of types is defined by the following inductive grammar:

$$t \in T \quad ::= \quad c \mid (t_1 \times t_2) \mid (t_1 + t_2) \mid t^* \mid \varepsilon$$

Values. The set of values $\mathcal{V}(t)$ of type t is defined by:

$$\begin{array}{ll} \mathcal{V}(c) & := \{c\} \\ \mathcal{V}(t_1 \times t_2) & := \{(v_1, v_2) \mid v_i \in \mathcal{V}(t_i)\} \\ \mathcal{V}(t_1 + t_2) & := \{e : v \mid e \in \{1,2\}, v \in \mathcal{V}(t_e)\} \\ \mathcal{V}(t^*) & := \{[v_1; \ldots; v_n] \mid v_i \in \mathcal{V}(t)\} \\ \mathcal{V}(\varepsilon) & := \{\varepsilon\} \end{array}$$

The symbol ε as a value denotes the sole value of ε as a type. We will use the letter σ to denote elements of $\mathcal{V}(t^*)$. Note that the values are structured elements, and no flattening happen automatically.

The flattening $\mathtt{flat}(v)$ of a value v is a word defined by:

$$\begin{array}{ll} \mathtt{flat}(c) & := [c] \\ \mathtt{flat}((v_1, v_2)) & := \mathtt{flat}(v_1)@\mathtt{flat}(v_2) \\ \mathtt{flat}(e : v) & := \mathtt{flat}(v) \\ \mathtt{flat}([v_1; \ldots; v_n]) & := \mathtt{flat}(v_1)@\ldots@\mathtt{flat}(v_n) \\ \mathtt{flat}(\varepsilon) & := [] \end{array}$$

We write $\mathtt{flat}(t) = \{\mathtt{flat}(v) \mid v \in \mathcal{V}(t)\}$ for the language accepted by the type t.

3 All-Match Semantics

In this section, we introduce an auxiliary definition of an all-match semantics that will be used to define our disambiguation policy and to study the problematic regular expressions. For a type t and a word $w \in \mathtt{flat}(t)$, we define

$$\mathtt{M}_t(w) := \{v \in \mathcal{V}(t) \mid \exists w'.\ w = \mathtt{flat}(v)@w'\}$$

This set represents all the possible ways to match a prefix of w by a value of type t. For a word w and a value $v \in \mathtt{M}_t(w)$, we write $v^{-1}w$ for the (unique) word w' such that $w = \mathtt{flat}(v)@w'$.

Definition 1. *A type is* problematic *if it contains a sub-expression of the form* t^* *where* $[] \in \mathtt{flat}(t)$.

Definition 2. *A value is* problematic *if it contains a sub-value of the form* $[\ldots; v; \ldots]$ *with* $\mathtt{flat}(v) = []$. *The set of non-problematic values of type* t *is written* $\mathcal{V}^{\mathtt{np}}(t)$.

Definition 3. *A word* w *is* problematic *for a type* t *if* $\mathtt{M}_t(w)$ *is infinite.*

The following proposition establishes the relation between these three notions.

Proposition 1. *Let t be a type. The following assertions are equivalent:*

1. *t is problematic;*
2. *there exists a problematic value in $\mathcal{V}(t)$;*
3. *there exists a word w which is problematic for t.*

We will often need to do induction both on a type t and a word w. To make it formal, we introduce a well-founded ordering on pairs (t, w): $(t_1, w_1) < (t_2, w_2)$ if either t_1 is a strict syntactic sub-expression of t_2 or $t_1 = t_2$ and w_1 is a strict suffix of w_2.

We write $\mathtt{M}_t^{\mathtt{np}}(w) = \mathtt{M}_t(w) \cap \mathcal{V}^{\mathtt{np}}(t)$ for the set of non-problematic prefix matches.

Proposition 2. *The following equalities hold:*

$$
\begin{aligned}
\mathtt{M}_c^{\mathtt{np}}(w) &= \begin{cases} \{c\} & \text{if } \exists w'.\ c :: w' = w \\ \emptyset & \text{otherwise} \end{cases} \\
\mathtt{M}_{t_1 \times t_2}^{\mathtt{np}}(w) &= \{(v_1, v_2) \mid v_1 \in \mathtt{M}_{t_1}^{\mathtt{np}}(w), v_2 \in \mathtt{M}_{t_2}^{\mathtt{np}}(v^{-1}w)\} \\
\mathtt{M}_{t_1 + t_2}^{\mathtt{np}}(w) &= \{e : v \mid e \in \{1,2\}, v \in \mathtt{M}_{t_e}^{\mathtt{np}}(w)\} \\
\mathtt{M}_{t^*}^{\mathtt{np}}(w) &= \{v :: \sigma \mid v \in \mathtt{M}_t^{\mathtt{np}}(w), \boxed{\mathtt{flat}(v) \neq []}, \sigma \in \mathtt{M}_{t^*}^{\mathtt{np}}(v^{-1}w)\} \cup \{[]\} \\
\mathtt{M}_{\varepsilon}^{\mathtt{np}}(w) &= \{\varepsilon\}
\end{aligned}
$$

This proposition gives a naive algorithm to compute $\mathtt{M}_t^{\mathtt{np}}(w)$. Indeed, because of the condition $\mathtt{flat}(v) \neq []$ in the case for $\mathtt{M}_{t^*}^{\mathtt{np}}(w)$, the word $v^{-1}w$ is a strict suffix of w, and we can interpret the equalities as an inductive definition for the function $\mathtt{M}_t^{\mathtt{np}}(w)$ (induction on (t, w)).

Note that if we remove this condition $\mathtt{flat}(v) \neq []$ and replace $\mathtt{M}_{\_}^{\mathtt{np}}(\_)$ with $\mathtt{M}_{\_}(\_)$, we get valid equalities.

Corollary 1. *For any word w and type t, $\mathtt{M}_t^{\mathtt{np}}(w)$ is finite.*

4 Disambiguation

A classical semantics of matching is defined by expanding the Kleene star t^* to $(t \times t^*) + \varepsilon$ and then relying on a disambiguation policy for the alternation (say, first-match policy). This gives a "greedy" semantics, which is sometimes meant as a local approximation of the longest match semantics. However, as described by Vansummeren [Van03], the greedy semantics does not implement the longest match policy. As a matter of fact, the greedy semantics really depends on the internals of Kleene-stars. For instance, consider the regular expressions $t_1 = ((a \times b) + a)^* \times (b + \varepsilon)$ and $t_2 = (a + (a \times b))^* \times (b + \varepsilon)$, and the word $w = ab$. With the greedy semantics, when matching w against t_1, the star captures *ab,* but when matching against t_2, the star captures only a.

Let t be a type. The matching problem is to compute from a word $w \in \mathtt{flat}(t)$ a value $v \in \mathcal{V}(t)$ whose flattening is w. In general, there are several different solutions. If we want to extract a single value, we need to define a disambiguation policy, that is, a way to choose a *best* value $v \in \mathcal{V}(t)$ such that $w = \mathtt{flat}(v)$. Moreover, we don't want to do it by providing an algorithm, or a set of ad hoc rules. Instead, we want to give a *declarative specification* for the disambiguation policy. To do this, we introduce a total ordering on the set $\mathcal{V}(t)$, and we specify that the best value with a given flattening is the largest value for this ordering.

We define the total (lexicographic) ordering $<$ on each set $\mathcal{V}(t)$ by:

$$\begin{array}{ll}
c < c & := \mathtt{false} \\
(v_1, v_2) < (v_1', v_2') & := (v_1 < v_1') \vee (v_1 = v_1' \wedge v_2 < v_2') \\
(e : v) < (e' : v') & := (e > e') \vee (e = e' \wedge v < v') \\
[] < \sigma' & := \sigma' \neq [] \\
v :: \sigma < v' :: \sigma' & := (v < v') \vee (v = v' \wedge \sigma < \sigma') \\
v :: \sigma < [] & := \mathtt{false} \\
\varepsilon < \varepsilon & := \mathtt{false}
\end{array}$$

This definition is well-founded by induction on the size of the values. It captures the idea of a specific disambiguation rule, namely a left-to-right policy for the sequencing, a first match policy for the alternation (we prefer the first of two alternatives, so the $1 : v$ should be larger than $2 : w$) and a greedy policy for the Kleene star.

Lemma 1. *Let t be a type and v a value in $\mathcal{V}(t)$. If v is problematic, then there exists some value $v' \in \mathcal{V}(t)$ such that $\mathtt{flat}(v) = \mathtt{flat}(v')$ and $v' > v$.*

The idea to prove this lemma is that a sequence σ corresponding to a subexpression t_0^* (with $[] \in \mathtt{flat}(t_0)$) can always be extended by appending values whose flattening is [], thus yielding strictly larger values for the ordering. Considering this lemma and Corollary 1, it is natural to restrict our attention to non problematic values. This is meaningful, because if $w \in \mathtt{flat}(t)$, then there always exist non-problematic values whose flattening is w.

Definition 4. *Let t be a type and $w \in \mathtt{flat}(t)$. We define:*
$\mathtt{m}_t(w) := \max_<\{v \in \mathcal{V}^{\mathtt{np}}(t) \mid \mathtt{flat}(v) = w\}$

The previous section gives a naive algorithm to compute $\mathtt{m}_t(w)$. We can first compute the set $\mathtt{M}_t^{\mathtt{np}}(w)$, then filter it to keep only the values v such that $v^{-1}w = []$, and finally extract the largest value from this set (if any). This algorithm is very inefficient because it has to materialize the set $\mathtt{M}_t^{\mathtt{np}}(w)$, which can be very large.

The recognition algorithm in [TSY02] or [Har99] can be interpreted in terms of our ordering. It generates the set $\mathtt{M}_t^{\mathtt{np}}(w)$ lazily, in decreasing order, and it stops as soon as it reaches the end of the input. To do this, it uses backtracking implemented with continuations. Adapting this algorithm to the matching problem is possible, but the resulting one would be quite inefficient because of backtracking (moreover, the continuations have to hold partial values, which generates a lot of useless memory allocations).

5 A Linear Time Matching Algorithm

In this section, we present an algorithm to compute $\mathtt{m}_t(w)$ in linear time with respect to the size of w, in particular without backtracking nor useless memory allocation.

This algorithm works in two passes. The main (second) pass is driven by the syntax of the type. It builds a value from a word by induction on the type, consuming the word from the left to the right. This pass must make some choices: which branch of the alternative type $t_1 + t_2$ to consider, or how many times to iterate a Kleene star t^*. To allow making these choices without backtracking, a first preprocessing pass annotates the word with enough information.

The first pass consists in running an automaton right to left on the word, and keeping the intermediate states as annotations between each symbol of the word. The automaton is build directly on the syntax tree of the regular expression itself (its states correspond to the nodes of the regular expression syntax tree). A reviewer pointed us to a previous work [Kea91] which uses the same idea. Our presentation is more functional (hence more amenable to reasoning) and is extended to handle problematic regular expressions.

5.1 Non-problematic Case

We first present an algorithm for the case when w is not problematic. Recall the following classical definition.

Definition 5. *A non-deterministic finite state automaton (FSA) with ε-transitions is a triple (Q, q_f, δ) where Q is a finite set (of states), q_f is a distinguished (final) state in Q, and $\delta \subset (Q \times \Sigma \times Q) \cup (Q \times Q)$.*

The transition relation $q_1 \xrightarrow{w} q_2$ (for $q_1, q_2 \in Q$, $w \in \Sigma^*$) is defined inductively by the following rules:

- $q_1 \xrightarrow{[]} q_2$ if $q_1 = q_2$ or $(q_1, q_2) \in \delta$
- $q_1 \xrightarrow{[c]} q_2$ if $(q_1, c, q_2) \in \delta$
- $q_1 \xrightarrow{w_1 @ w_2} q_3$ if $q_1 \xrightarrow{w_1} q_2$ and $q_2 \xrightarrow{w_2} q_3$.

We write $\mathcal{L}(q) = \{w \mid q \xrightarrow{w} q_f\}$.

From types to automata. Constructing a non-deterministic automaton from a regular expression is a standard operation. However, we need to keep a tight connection between the automata and the types. To do so, we endow the abstract syntax trees of types with a transition relation so as to turn them into automata. Formally, we introduce the set of *locations* (or nodes) $\lambda(t)$ of a type t (a location is a sequence over $\{\mathtt{fst}, \mathtt{snd}, \mathtt{lft}, \mathtt{rgt}, \mathtt{star}\}$), and for a location $l \in \lambda(t)$, we define $t.l$ as the subtree rooted at location l:

$$\begin{cases} \lambda(c) & := \{[]\} \\ \lambda(t_1 \times t_2) & := \{[]\} \cup \mathtt{fst} :: \lambda(t_1) \cup \mathtt{snd} :: \lambda(t_2) \\ \lambda(t_1 + t_2) & := \{[]\} \cup \mathtt{lft} :: \lambda(t_1) \cup \mathtt{rgt} :: \lambda(t_2) \\ \lambda(t^*) & := \{[]\} \cup \mathtt{star} :: \lambda(t) \\ \lambda(\varepsilon) & := \{[]\} \end{cases} \quad \begin{cases} t.[] & := t \\ (t_1 \times t_2).(\mathtt{fst} :: l) & := t_1.l \\ (t_1 \times t_2).(\mathtt{snd} :: l) & := t_2.l \\ (t_1 + t_2).(\mathtt{lft} :: l) & := t_1.l \\ (t_1 + t_2).(\mathtt{rgt} :: l) & := t_2.l \\ (t^*).(\mathtt{star} :: l) & := t.l \end{cases}$$

Now, let us consider a fixed type t_0. We take: $Q := \lambda(t_0) \cup \{q_f\}$ where q_f is a fresh element.

If l is a location in t_0, the corresponding state will match all the words of the form $w_1@w_2$ where w_1 is matched by $t_0.l$ and w_2 is matched by the "rest" of the regular expression (Lemma 2 below gives a formal statement corresponding to this intuition).

We define the δ relation for our automaton by using the successor function $\mathtt{succ}(\_) : \lambda(t_0) \to Q$ which formalizes this notion of "rest":

$$\begin{aligned} \delta := & \{(l, c, \mathtt{succ}(l)) \mid t_0.l = c\} \\ & \cup \{(l, \mathtt{succ}(l)) \mid t_0.l = \varepsilon\} \\ & \cup \{(l, l :: \mathtt{fst}) \mid t_0.l = t_1 \times t_2\} \\ & \cup \{(l, l :: \mathtt{lft}), (l, l :: \mathtt{rgt}) \mid t_0.l = t_1 + t_2\} \\ & \cup \{(l, l :: \mathtt{star}), (l, \mathtt{succ}(l)) \mid t_0.l = t^*\} \end{aligned} \quad \begin{cases} \mathtt{succ}([]) & := q_f \\ \mathtt{succ}(l :: \mathtt{fst}) & := l :: \mathtt{snd} \\ \mathtt{succ}(l :: \mathtt{snd}) & := \mathtt{succ}(l) \\ \mathtt{succ}(l :: \mathtt{lft}) & := \mathtt{succ}(l) \\ \mathtt{succ}(l :: \mathtt{rgt}) & := \mathtt{succ}(l) \\ \mathtt{succ}(l :: \mathtt{star}) & := l \end{cases}$$

An example for this construction will be given in the next session for the problematic case.

The following lemma relates the behavior of the automaton, the succ(_) function, and the flat semantics of types.

Lemma 2. *For any location* $l \in \lambda(t_0)$: $\mathcal{L}(l) = \mathtt{flat}(t_0.l)@\mathcal{L}(\mathtt{succ}(l))$

First pass. We can now describe the first pass of our matching algorithm. Assume that the input is $w = [c_1; \ldots; c_n]$. The algorithm computes $n + 1$ sets of states $Q_0, \ldots, Q_n$ defined as $Q_i = \{q \mid q \xrightarrow{[c_{i+1}; \ldots; c_n]} q_f\}$. That is, it annotates each suffix w' of the input w by the set of states from which the final state can be reached by reading w'.

Computing the sets Q_i is easy. Indeed, consider the automaton obtained by reversing all the transitions in our automaton (Q, q_f, δ), and use it to scan w right-to-left, starting from q_f, with the classical subset construction (with forward ε-closure). Each step of the simulation corresponds to a suffix $[c_{i+1}; \ldots; c_n]$ of w, and the subset built at this step is precisely Q_i.

This pass can be done in linear time with respect to the length of w, and more precisely in time $O(|w| \times |t_0|)$ where $|w|$ is the length of w and t_0 is the size of t_0.

Second pass. The second pass is written in pseudo-ML code, as a function build, that takes a pair (w, l) of a word and a location $l \in \lambda(t_0)$ such that $w \in \mathcal{L}(l)$ and returns a value $v \in \mathcal{V}(t_0.l)$.

```
let build(w,l) = (* Invariant:  w ∈ L(l)  *)
 match t0.l with
 | c -> c
 | t1 × t2 ->
     let v1 = build(w, l :: fst) in let v2 = build(v1^-1 w, l :: snd) in (v1, v2)
 | t1 + t2 ->
     if w ∈ L(l :: lft)
     then let v1 = build(w, l :: lft) in 1 : v1
     else let v2 = build(w, l :: rgt) in 2 : v2
 | t* ->
     if w ∈ L(l :: star)
     then let v = build(w, l :: star) in let σ = build(v^-1 w, l) in v :: σ
     else []
 | ε -> ε
```

The following proposition explains the behavior of the algorithm, and allows us to establish its soundness.

Proposition 3. *If* $w \in \mathcal{L}(l)$ *and if* t_0 *is* **non-problematic**, *then the algorithm* $\texttt{build}(w, l)$ *returns* $\max_<\{v \in \mathcal{V}(t_0.l) \mid \exists w' \in \mathcal{L}(\texttt{succ}(l)).\ w = \texttt{flat}(v)@w'\}$.

Corollary 2. *If* $w \in \texttt{flat}(t_0)$ *and if* t_0 *is* **non-problematic**, *then the algorithm* $\texttt{build}(w, [])$ *returns* $\texttt{m}_{t_0}(w)$.

Implementation. The tests $w \in \mathcal{L}(l)$ can be implemented in constant time thanks to the first pass [1] . Indeed, for a suffix w' of the input, $w' \in \mathcal{L}(l)$ means that the state l is in the set attached to w' in the first pass. Similarly, the precondition $w \in \texttt{flat}(t_0)$ can also be tested in constant time.

The second pass also runs in linear time with respect to the length of the input word (and more precisely in time $O(|w|\ |t_0|)$), because `build` is called at most once for each suffix w' of w and each location l (the number of locations is finite). This property holds because of the non-problematic assumption (otherwise the algorithm may not terminate).

Note that w is used *linearly* in the algorithm: it can be implemented as a mutable pointer on the input sequence (which is updated when the c case reads a symbol), and it doesn't need to be passed around.

5.2 Solution to the Problematic Case

Idea of a solution. Let us study the problem with problematic types in the algorithm from the previous section. The problem is in the case t^* of the algorithm, when $[] \in \texttt{flat}(t)$. Indeed, the first recursive call to `build` may return a value v such that $\texttt{flat}(v) = []$, which implies $v^{-1}w = w$, and the second recursive call

[1] If the regular expressions are 1-unambiguous (which is the case for regular expressions in DTD and XML Schema [W3C01]), the tests can be implemented directly with a look-ahead of one symbol, without the first pass.

has then the same arguments as the main call. In this case, the algorithm does not terminate.

This can also be seen on the automaton. If the type at location l accepts the empty sequence, there are in the automaton non-trivial paths of ε-transitions from l to l. The idea is to break these paths, by "disabling" their last transition (the one that returns to l) when no symbol has been matched in the input word since the last visit of the state l.

Here is how to do so. A location l is said to be a star node if $t_0.l = t^*$. Any sublocation l' is said to be scoped by l. Note that when the automaton starts an iteration in a star node (by using the ε transition $(l, l :: \mathbf{star})$), the only way to exit the iteration (and to reach the final state) is to go back to the star node l. The idea is to prevent the automaton to enter back a star node unless some symbol has been read during the last iteration. The state of the automaton includes a flag b that is set whenever a character is read. The flag is reset when an iteration starts, that is, when a transition of the form $(l, l :: \mathbf{star})$ is used. When the flag is not set, all epsilon transitions of the form $(l, \mathbf{succ}(l))$, where $\mathbf{succ}(l)$ is a star node scoping l, are disabled.

When the flag is set, this can be interpreted as the requirement: Something needs to be read in order to exit the current iteration. Consequently, it is natural to start running the automaton with the flag set, and to require the flag to be set at the final node.

From problematic types to automata. Let us make this idea formal. We write P for the set of locations l such that $\mathbf{succ}(l)$ is an ancestor of l in the abstract syntax tree of t_0 (this implies that $\mathbf{succ}(l)$ is a star node). Note that the "problematic" transitions are the ε-transition of the form $(l, \mathbf{succ}(l))$ with $l \in P$.

We now take: $Q := (\lambda(t_0) \cup \{q_f\}) \times \{0, 1\}$. Instead of (q, b), we write q^b. The final state is q_f^1. Here is the transition relation:

$$\begin{aligned}
\delta_0 := & \{(l^b, c, \mathbf{succ}(l)^1) \mid t_0.l = c\} \\
& \cup \{(l^b, l :: \mathbf{fst}^b) \mid t_0.l = t_1 \times t_2\} \\
& \cup \{(l^b, l :: \mathbf{lft}^b), (l^b, l :: \mathbf{rgt}^b) \mid t_0.l = t_1 + t_2\} \\
& \cup \{(l^b, l :: \mathbf{star}^0) \mid t_0.l = t^*\} \\
& \cup \{(l^b, \mathbf{succ}(l)^b) \mid (*)\}
\end{aligned}$$

where the condition $(*)$ is the conjunction of:

(I) $t_0.l$ is either ε or a star t^*
(II) if $l \in P$, then $b = 1$

Note that the transition relation is monotonic with respect to the flag b: if $q_1^0 \xrightarrow{w} q_2^b$, then $q_1^1 \xrightarrow{w} q_2^{b'}$ for some $b' \geq b$.

We write $\mathcal{L}(q^b) := \{w \mid q^b \xrightarrow{w} q_f^1\}$. As for any FSA, we can simulate the new automaton either forwards or backwards. In particular, it is possible to annotate a word w with a right-to-left traversal (in linear time w.r.t the length of w), so as to be able to answer in constant time any question of the form $w' \in \mathcal{L}(q^b)$ where w' is a suffix of w. This can be done with the usual subset construction.

The monotonicity remark above implies that whenever q^0 is in a subset, then q^1 is also in a subset, which allows to optimize the representation of the subsets.

The lemma above is the invariant used to prove Proposition 4.

Lemma 3. *Let* $l \in \lambda(t_0)$ *and* $L = \mathtt{flat}(t_0.l)$. *Then:*

$$\mathcal{L}(l^1) = L@\mathcal{L}(\mathtt{succ}(l)^1)$$
$$\mathcal{L}(l^0) = \begin{cases} (L\backslash\{[]\})@\mathcal{L}(\mathtt{succ}(l)^1) & \text{if } l \in P \vee [] \notin L \\ (L\backslash\{[]\})@\mathcal{L}(\mathtt{succ}(l)^1) \cup \mathcal{L}(\mathtt{succ}(l)^0) & \text{if } l \notin P \wedge [] \in L \end{cases}$$

Algorithm. We now give a version of the linear-time matching algorithm which supports the problematic case. The only difference is that it keeps track (in the flag b) of the fact that something has been consumed on the input since the last beginning of an iteration in a star. The first pass is not modified, except that the new automaton is used. The second pass is adapted to keep track of b.

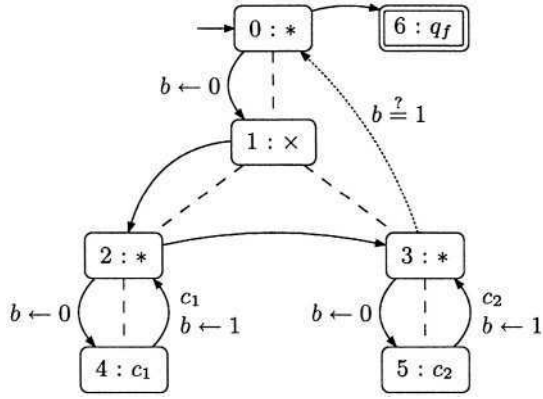
```
let build'(w, l^b) = (* Invariant:  w ∈ L(l^b)  *)
 match t0.l with
 | c -> c
 | t1 × t2 ->
     let v1 = build'(w, l :: fst^b) in
     let b' = if (v1^-1 w = w) then b else 1 in
     let v2 = build'(v1^-1 w, l :: snd^b') in
     (v1, v2)
 | t1 + t2 ->
     if w ∈ L(l :: lft^b)
     then let v1 = build'(w, l :: lft^b) in 1 : v1
     else let v2 = build'(w, l :: rgt^b) in 2 : v2
 | t* ->
     if w ∈ L(l :: star^0)
     then let v = build'(w, l :: star^0) in let σ = build'(v^-1 w, l^1) in v :: σ
       (* Invariant: v^-1 w ≠ w *)
     else []
 | ε -> ε
```

Proposition 4. *Let* $w \in \mathcal{L}(l^b)$. *Let* V *be the set of non-problematic values* $v \in \mathcal{V}(t_0.l)$ *such that* $\exists w' \in \mathcal{L}(\mathtt{succ}(l)^{b'})$. $w = \mathtt{flat}(v)@w'$ *with* $b' = 1$ *if* $\mathtt{flat}(v) \neq []$ *and* $((b = 1 \vee l \notin P) \wedge b' = b)$ *if* $\mathtt{flat}(v) = []$. *Then the algorithm* $\mathtt{build}'(w, l^b)$ *returns* $\max_< V$.

Corollary 3. *If* $w \in \mathtt{flat}(t_0)$, *then the algorithm* $\mathtt{build}'(w, []^1)$ *returns* $\mathtt{m}_{t_0}(w)$.

Implementation. The same remarks as for the first algorithm apply for this version. In particular, we can implement w and b with mutable variables which are updated in the case c (when a symbol is read); thus, we don't need to compute b' explicitly in the case $t_1 \times t_2$.

Example. To illustrate the algorithm, let us consider the problematic type $t_0 = (c_1^* \times c_2^*)^*$. The picture below represents both the syntax tree of this type (dashed lines), and the transitions of the automaton (arrows). The dotted arrow is the only problematic transition, which is disabled when $b = 0$. Transitions with no symbols are ε-transitions. To simplify the notation, we assign numbers to states.



Let us consider the input word $w = [c_2; c_1]$. The first pass of the algorithm runs the automaton backwards on this word, starting in state 6^1, and applying subset construction. In a remark above, we noticed that if i^0 is in the subset, then i^1 is also in the subset. Consequently, we write simply i to denote both states i^0, i^1. The ε-closure of 6^1 is $S_2 = \{6^1, 0^1, 3^1, 2^1, 1^1\}$. Reading the symbol c_1 from S_2 leads to the state 4, whose ε-closure is $S_1 = \{4, 2, 1, 0, 3^1\}$. Reading the symbol c_2 from S_1 leads to the state 5, whose ε-closure is $S_0 = \{5, 3, 2, 1, 0\}$.

Now we can run the algorithm on the word w with the trace $[S_0; S_1; S_2]$. The flag b is initially set. The star node 0 checks whether it must enter an iteration, that is, whether $1 \in S_0$. This is the case, so an iteration starts, and b is reset. The star node 2 returns immediately without a single iteration, because $4 \notin S_0$. But the star node 3 enters an iteration because $5 \in S_0$. This iteration consumes the first symbol of w, and sets b. After this first iteration, the current subset is S_1. As 5 is not in S_1, the iteration of the node 3 stops, and the control is given back to the star node 0. Since $1 \in S_1$, another iteration of the star 0 starts, and then similarly with an inner iteration of 2. The second symbol of w is consumed. The star node 3 (resp. 0) refuses to enter an extra iteration because $5 \notin S_2$ (resp. $1 \notin S_2$); note that $1^1 \in S_2$, but this is not enough, as this only means that an iteration could take place without consuming anything - which is precisely the situation we want to avoid.

The resulting value is $[([], [c2]); ([c1], [])]$. The two elements of this sequence reflect the two iterations of the star node 0.

Acknowledgments. We would like to express our gratitude to the reviewers of PLAN-X 2004 and ICALP 2004 for their comments and in particular for their bibliographical indications.

References

[BCF03a] Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. CDuce: An XML-centric general-purpose language. In ICFP'03, 2003.

[BCF+03b] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0: An XML Query Language. W3C Working Draft, http://www.w3.org/TR/xquery/, May 2003.

[DF00] Danny Dub and Marc Feeley. Efficiently building a parse tree from a regular expression. Acta Informatica, 37(2): 121-144, 2000.

[ECM02] ECMA. CLI Partition I - Architecture. http://msdn.microsoft.com/net/ecma/, 2002.

[Fri04] Alain Frisch. Regular tree language recognition with static information. International Conference on Theoretical Computer Science, 2004.

[GP03] V. Gapayev and B.C. Pierce. Regular object types. In Proceedings of the 10th workshop FOOL, 2003.

[Har99] Robert Harper. Proof-directed debugging. Journal of Functional Programming, 9(4):463-469, 1999.

[Hos01] Haruo Hosoya. Regular Expression Types for XML. PhD thesis, The University of Tokyo, 2001.

[Hos03] H. Hosoya. Regular expressions pattern matching: a simpler design. Unpublished manuscript, February 2003.

[HP01] Haruo Hosoya and Benjamin C. Pierce. Regular expression pattern matching for XML. In The 25th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2001.

[HP03] Haruo Hosoya and Benjamin C. Pierce. XDuce: A typed XML processing language. ACM Transactions on Internet Technology, 3(2):117-148, 2003.

[HVP00] Haruo Hosoya, Jérôme Vouillon, and Benjamin C. Pierce. Regular expression types for XML. In ICFP '00, volume 35(9) of SIGPLAN Notices, 2000.

[Kea91] Steven. M. Kearns. Extending regular expressions with context operators and parse extraction. Software - practice and experience, 21(8):787-804, 1991.

[Lau0l] Ville Laurikari. Efficient submatch addressing for regular expressions. Master's thesis, Helsinki University of Technology, 2001.

[Lev03] Michael Levin. Compiling regular patterns. In ICFP'03, 2003.

[MS03] Erik Meijer and Wolfram Schulte. Unifying tables, objects, and documents. In DP-COOL 2003, 2003.

[TSY02] Naoshi Tabuchi, Eijiro Sumii, and Akinori Yonezawa. Regular expression types for strings in a text processing language. In Workshop on Types in Programming (TIP), 2002.

[Van03] Stijn Vansummeren. Unique pattern matching in strings. Technical report, University of Limburg, 2003. http://arXiv.org/abs/cs/0302004.

[W3C00] W3C Recommendation. Extensible Markup Language (XML) 1.0, 2000.

[W3C01] W3C Recommandation. XML Schema, 2001.

[Xi01] Hongwei Xi. Dependent types for program termination verification. In Logic in Computer Science, 2001.

A $2^{O(n^{1-\frac{1}{d}}\log n)}$ Time Algorithm for d-Dimensional Protein Folding in the HP-Model

Bin Fu[1] and Wei Wang[2]

[1] Department of Computer Science, University of New Orleans, New Orleans, LA 70148 and Research Institute for Children, 200 Henry Clay Avenue, New Orleans, LA 70118
fu@cs.uno.edu
[2] Department of Chemistry and Biochemistry, University of California at San Diego, CA 92093
wwang@chem.ucsd.edu

Abstract. The protein folding problem in the HP-model is NP-hard in both 2D and 3D [4,6]. The problem is to put a sequence, consisting of two characters H and P, on a d-dimensional grid to have the maximal number of HH contacts. We design a $2^{O(n^{1-\frac{1}{d}}\log n)}$ time algorithm for d-dimensional protein folding in the HP-model. In particular, our algorithm has $O(2^{6.145\sqrt{n}\log n})$ and $O(2^{4.306n^{\frac{2}{3}}\log n})$ computational time in 2D and 3D respectively. The algorithm is derived via our separator theorem for points on a *d*-dimensional grid. For example, for a set of n points P on a 2-dimensional grid, there is a separator with at most $1.129\sqrt{n}$ points that partitions P into two sides with at most $(\frac{2}{3})n$ points on each side. Our separator theorem for grid points has a greatly reduced upper bound than that for the general planar graph [2].

1 Introduction

Proteins are composed of 20 amino acids. Two amino acids can be connected via a peptide bond. A protein sequence can be generated by using peptide bonds to connect amino acids. A protein can fold into a specific 3D structure, which is uniquely determined by the sequence of amino acids. Its 3D structure determines its function. A standard procedure to determine 3D structure is to produce a pure solution with only the protein, then crystallize it followed by x-ray crystallography. This is a very time consuming process. Therefore, protein structure prediction with computational technology is one of the most significant problems in bioinformatics.

It is much easier to identify a protein's 1D sequence than its 3D structure. In order to carry out their various functions, proteins must fold into a 3D structure. By studying how proteins fold, their functions can be better understood. The study of protein folding can help answer questions such as how a protein changes to a totally different function or how the function of a protein changes with its structure.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 630–644, 2004.

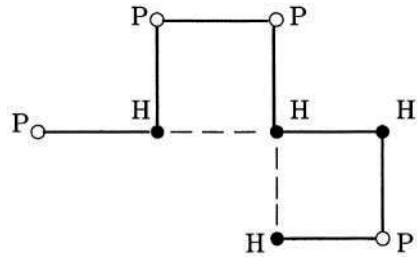


Fig. 1. The sequence PHPPHHPH is put on the 2 dimensional grid. There are 2 H-H contacts marked by the dotted lines.

A simplified representation of proteins is a lattice conformation, which is a self-avoiding sequence in Z^3. An important representative of lattice models is the HP-model, which was introduced in [14,15]. In this model, the 20 letter alphabet of amino acids is reduced to a two letter alphabet, namely H and P. H represents hydrophobic amino acids, whereas P represents polar or hydrophilic amino acids. Two monomers form a contact in some specific conformation if they are not consecutive, but occupy neighboring positions in the conformation(i.e., the distance vector between their positions in the conformation is a unit vector). A conformation with minimal energy is just a conformation with the maximal number of contacts between non-consecutive H-monomers. The folding problem in the HP-model is to find the conformation for any HP-sequence with minimal energy. This problem was proven to be NP-hard in both 2D and 3D [4,6].

Some algorithms for this problem have been developed based on the heuristic, genetic, Monte Carlo, branch and bound methods (e.g. [26,27,28,25,19,22, 12,13,21,17,23,7,3]). Although many experimental results were reported for testing sequences of small length, we have not seen any theoretical analysis about the computational time upper bound of the algorithms. Another approach is to develop polynomial time approximation algorithms for the protein folding in the HP model [10,1,18]. Hart and Istrail [10] showed a polynomial time $\frac{3}{8}$-approximation algorithm for the 3D folding in the HP model and Newman [18] derived a polynomial time $\frac{1}{3}$-approximation algorithm for the 2D problem, improving $\frac{1}{4}$-approximation algorithm in [10].

If the first letter of a HP sequence is fixed at a position of 2D (3D) plane (space), we have at least 2^{n-1} (3^{n-1}) ways and at most $3^{n-1}(5^{n-1})$ ways to put the rest of the letters on the plane (space resp.). Our algorithm computational time is bounded by $2^{O(n^{\frac{1}{2}}\log n)}$ $(2^{O(n^{\frac{2}{3}}\log n)})$ in 2D (3D resp.). As the average number of amino acids of proteins is between 400 to 600, if an algorithm could solve the the protein structure prediction with ≤ 1000 amino acids, it would be able to satisfy most of the application demand. Our effort is a theoretical step toward this target.

Our algorithm is the divide and conquer approach, which is based on our geometric separator for the points on a d-dimensional grid. Lipton and Tarjan [16] showed the well known geometric separator for planar graphs. Their result has

been elaborated by many subsequent authors. The best known separator theorem for planar graphs was proved by Alon, Seymour and Thomas [2].

Theorem 1. *[2] Any planar graph of n vertices has a vertex subset of cardinality $\leq \sqrt{4.5n}$ whose removal separates the graph into two components each having $\frac{2n}{3}$ vertices.*

Some other forms of the separator theorem were applied in deriving algorithms for some geometric problems such as the planar Travelling Salesman and Steiner Tree problems (e.g. see [24]). Those problems usually have input points with fixed geometric positions in space. A set of grid points on the plane forms a planar graph by adding edges to every two grid points with distance 1. As the input of folding problem is only a sequence of letters, their locations in space are unknown and will be determined by the algorithm. We do not know if the separator theorem like Theorem 1 can be applied to the folding problem. We derive a separator theorem for the grid points with a greatly reduced upper bound for the number of points on the separator than that for the planar graph.

Theorem 2. *For a set P of n grid points on a 2-dimensional plane, there is a line on the plane and a subset $Q \subseteq P$ of cardinality $\leq 1.129\sqrt{n}$ such that each halfplane contains at most $\frac{2}{3}n$ points of P, and every two points $p_1, p_2 \in P$ on the different sides of the line have distance > 1 unless at least one of p_1, p_2 is in Q.*

Furthermore, we also provide $O(n^2)$ possible locations to find such a line based on the folding region within a fixed $n \times n$ square. This makes it is possible to use the separator theorem in the algorithm for the folding problem even though the locations of the letters are not known.

2 An Easy Separator and Algorithm

We will show that there is a small set of letters with size $O(n^{1-\frac{1}{d}})$ on a hyper plane (denoted by $P_{r,a}$ for some $1 \leq r \leq d$ and integer a in the definition below) to partition the folding problem of n letters into 2 problems of $\leq c(d)n$ letters, where $0 < c(d) < 1$, $c(d)$ is a constant for fixed d and n is the size of the input (the number of H and P characters). The 2 smaller problems are recursively solved and their solutions are merged to derive the solution to the original problem. As the separator has only $O(n^{1-\frac{1}{d}})$ letters, there are at most $n^{O(n^{1-\frac{1}{d}})}$ cases to partition the problem. The separator in this section has a self-contained proof and implies an $n^{O(n^{1-\frac{1}{d}})}$-time algorithm for the folding problem in the HP-model.

2.1 A Balanced Separator

Let the dimensional number d be fixed. We need the following terms:

Definition 3.

- For a set A, $|A|$ is denoted as the number of elements in A.

- The integer set is represented by $Z = \{\cdots, -2, -1, 0, 1, 2, \cdots\}$. For integers i and j, *integer interval* $[i, j] = \{i, i+1, \cdots, j\}$. For integers $x_1, \cdots, x_d$, $(x_1, \cdots, x_d)$ is a d-dimensional *grid point.*
- For two points p_1, p_2 with the same dimension, $\text{dist}(p_1, p_2)$ is the Euclidean distance between them.
- For a set Σ of letters, a *Σ-sequence* is a sequence of letters from Σ. For example, *PHPPHHPH* is an $\{H, P\}$-sequence. For a sequence S of length n and $1 \le i \le n$, $S[i]$ is the i-th letter of S. $S[i, j]$ denotes the subsequence $S[i]S[i+1]\cdots S[j]$. If $[i_1, j_1], [i_2, j_2], \cdots, [i_t, j_t]$ are disjoint intervals inside $[1, n]$, we call $S[i_1, j_1], S[i_2, j_2], \cdots, S[i_t, j_t]$ *disjoint subsequences* of S. For a set of integers $A = \{i_1 < i_2 < \cdots < i_k\}$, define $S[A] = S[i_1]S[i_2]\cdots S[i_k]$.
- For a d-dimensional point $(x_1, \cdots, x_d)$, define $||(x_1, \cdots, x_d)|| = \sum_{i=1}^{d} |x_i|$.
- A *self-avoiding arrangement* f for a sequence S of length n on the d-dimensional grid is a one-to-one mapping from $\{1, 2, \cdots, n\}$ to Z^d such that $||f(i) - f(i+1)|| = 1$ for $i = 1, 2, \cdots, n-1$. For the disjoint subsequences $S[i_1, j_1], \cdots, S[i_k, j_k]$ of 5, a *partial self-avoiding arrangement* of S on $S[i_1, j_1], \cdots, S[i_k, j_k]$ is a partial function f from $\{1, 2, \cdots, n\}$ to Z^d such that f is defined on $\cup_{t=1}^{k}[i_t, j_t]$, and f can be extended to a (full) self-avoiding arrangement of S on Z^d.
- For a grid self-avoiding arrangement, its *contact map* is the graph $G_f = (1, 2, \cdots, n, E)$, where the edge set $E = \{(i, j) : |i-j| > 1 \text{ and } ||f(i)-f(j)|| = 1\}$.
- A r-plane is the set $P_{r,a} = \{(x_1, \cdots, x_{r-1}, a, x_{r+1}, \cdots, x_d) | x_1, \cdots, x_{r-1}, x_{r+1}, \cdots, x_d \in Z\}$, which has all of the elements in Z^d with the r-th element of fixed value a.
- $P_{r,<a} = \{(x_1, \cdots, x_{r-1}, x_r, x_{r+1}, \cdots, x_d) | x_1, \cdots, x_{r-1}, x_r, x_{r+1}, \cdots, x_d \in Z \text{ and } x_r < a\}$.
- $P_{r,>a} = \{(x_1, \cdots, x_{r-1}, x_r, x_{r+1}, \cdots, x_d) | x_1, \cdots, x_{r-1}, x_r, x_{r+1}, \cdots, x_d \in Z \text{ and } x_r > a\}$.
- $P_{r,\le a} = Pr_{r,<a} \cup P_{r,a}$, and $P_{r,\ge a} = P_{r,>a} \cup P_{r,a}$.
- For a set of points S in d-dimensional space and $1 \le r \le d$ and $a \in Z$, define $S(r, <a) = \{(x_1, \cdots, x_d) \in S | x_r < a\}$, $S(r, = a) = \{(x_1, \cdots, x_d) \in S | x_r = a\}$, and $S(r, > a) = \{(x_1, \cdots, x_d) \in S | x_r > a\}$.
- For $0 < c < 1$ and a set S in d-dimensional space, a $P_{r,a}$ is a *c-balanced-separator* if $|S(r, <a)| \le c \cdot |S|$ and $|S(r, >a)| \le c \cdot |S|$.
- A *rectangular region* R in d-dimensional space is the intersection of a finite number of sets $P_1, P_2, \cdots, P_k$, where $P_i = P_{r,<a}$ or $P_i = P_{r,>a}$ with $1 \le r \le d$ and $a \in Z$ for $(i = 1, \cdots, k)$.
- A rectangular region R in d-dimensional space is of *size* $m_1 \times m_2 \times \cdots \times m_d$ if $m_i = max\{x_i - x_i' | (x_1, \cdots, x_d), (x_1', \cdots, x_d') \in R\} + 1$ for $i = 1, \cdots, d$.

Lemma 4. *For a set S of n grid points in d-dimensional space, there is a $c(d)$-balanced-separator P^* that contains at most $\le c'(d)n^{1-\frac{1}{d}}$ points from S, where $0 < c(d) < 1, 0 < c'(d)$ and both $c(d)$ and $c'(d)$ are constants for a fixed dimensional number d.*

Proof. We will construct a series of sets $S = S_0 \supseteq S_1 \supseteq S_2 \supseteq \cdots \supseteq S_t$ such that $t \leq d-1$ and $|S_i| \geq \frac{1}{2}|S_{i-1}|$ for $i = 1, 2, \cdots, t$. The construction of P^* starts from Stage 0 and can go up to Stage d.

Stage 0: Let $S_0 = S$ and $r = 1$. Enter stage 1. **End of Stage 0.**

Stage r $(1 \leq r \leq d-1)$: Let Q_r contain all of the $P_{r,a}$ such that $P_{r,a}$ is a $\frac{3}{4}$-balanced-separator for S_{r-1}. At most the $\frac{1}{4}$ elements in S_{r-1} with smallest a values (for the r-th entry) stay on the left of all $\frac{3}{4}$-balanced separators and at most the $\frac{1}{4}$ elements in S_{r-1} with largest a values (for the r-th entry) stay on the right of all $\frac{3}{4}$-separators. The set $\cup_{P_{r,a} \in Q_r} P_{r,a}$ has at least $\frac{1}{2}$ elements from S_{r-1}. So, Q_r is not empty. If a $P_{r,a}$ in Q_r contains no more than $n^{1-\frac{1}{d}}$ elements from S, let $P^* = P_{r,a}$ and terminate the construction. We have $|S_{r-1}| \geq \frac{1}{2^{r-1}}|S|$ and

$$|S(r, <a)| \leq |S_{r-1}(r, <a)| + |S - S_{r-1}| \leq \frac{3}{4}|S_{r-1}| + |S| - |S_{r-1}| \quad (1)$$

$$= |S| - \frac{1}{4}|S_{r-1}| \leq (1 - \frac{1}{2^{r+1}})|S| \leq (1 - \frac{1}{2^d})|S| \quad (2)$$

Similarly, $|S(r, >a)| \leq (1 - \frac{1}{2^d})|S|$.

If every $P_{r,a} \in Q_r$ has $> n^{1-\frac{1}{d}}$ elements from S, $|Q_r| \leq n^{\frac{1}{d}}$ because $|\cup_{P_{r,a} \in Q_r} (P_{r,a} \cap S)| \leq |S| = n$ and all planes in Q_r are disjoint from each other. It is easy to see that there is an integer interval $[c_1, c_2]$ such that $Q_r = \{P_{r,a} | a \in [c_1, c_2]\}$. Let $S_r = \cup_{P_{r,a} \in Q_r} (P_{r,a} \cap S_{r-1})$. We have $S_r \subseteq S_{r-1}$ and $|S_r| \geq |S_{r-1}|/2$ (because $[c_1, c_2]$ is the set of all integers a such that $P_{r,a}$ is a $\frac{3}{4}$-balanced-separator). Let $r = r + 1$ and go to the next stage. **End of stage r.**

Stage d: Assume for each r with $1 \leq r \leq d-1$, Q_r has no plane $P_{r,a}$ with elements $\leq n^{1-\frac{1}{d}}$ from S. Hence, $|Q_r| \leq n^{\frac{1}{d}}$ for $1 \leq r \leq d-1$. If a is fixed, every $p \in P_{r,a}$ has the r-th entry equal to a. Therefore, $\{x_r | x_r$ is the $r-$th entry of some $p \in P_{r,a}$ for some $P_{r,a} \in Q_r\}$ has $\leq n^{\frac{1}{d}}$ elements since $|Q_r| \leq n^{\frac{1}{d}}$ $(1 \leq r \leq d-1)$. This implies that for every $P_{d,a}$, $|\{p | p \in P_{r,a_r}$ for some $P_{r,a_r} \in Q_r (r = 1, \cdots, d-1)$ and $p \in P_{d,a}\}| \leq (n^{\frac{1}{d}})^{d-1} = n^{\frac{d-1}{d}}$.

As $|S_{d-1}| \geq \frac{|S|}{2^{d-1}} = \frac{1}{2^{d-1}}n$, there are at least $\frac{\frac{1}{2}|S_{d-1}|}{n^{1-\frac{1}{d}}} \geq \frac{1}{2^d} \cdot n^{\frac{1}{d}}$ $P_{d,a}$s to be $\frac{3}{4}$-balanced-separator for S_{d-1}. One of them has at most $\frac{|S|}{\frac{1}{2^d} n^{\frac{1}{d}}} = 2^d n^{1-\frac{1}{d}}$ elements from S. Let P^* be such a $P_{d,a}$. As $|S_{d-1}| \geq \frac{1}{2^{d-1}}|S|$, we have

$$|S(d, <a)| \leq |S_{d-1}(d, <a)| + |S - S_{d-1}| \leq \frac{3}{4}|S_{d-1}| + |S| - |S_{d-1}| \quad (3)$$

$$= |S| - \frac{1}{4}|S_{d-1}| \leq (1 - \frac{1}{2^{d+1}})|S| \quad (4)$$

Similarly, we also have $|S(d, >a)| \leq (1 - \frac{1}{2^{d+1}})|S|$. **End of stage d.**

For a d-dimensional cube that contains n grid points, its edge length is $n^{\frac{1}{d}}$. Every hyper plane $P_{r,a}$, which intersects the cube, shares $n^{\frac{d-1}{d}}$ grid points with

the cube. This shows it is impossible to improve the separator to $o(n^{\frac{d-1}{d}})$. The next section shows that we can improve the separator by a constant factor. This lemma indicates that the balanced separator can be found among $O(dn)$ hyper-planes.

2.2 Algorithm

As we are going to describe our algorithm recursively, we use the following term to characterize the problem. A d-dimensional **Multi-Sequence Folding Problem** F is formulated as follows:

The inputs are

1. disjoint subsequences $S_1, S_2, \cdots, S_k$ of sequence S_0 ($S_t = S_0[i_t, j_t]$ for $t = 1, \cdots, k$), and
2. a rectangular region R, where all of the k $\{H, P\}$-sequences are going to be arranged, and
3. a series of k pairs of points in R: $(p_1, q_1), (p_2, q_2), \cdots, (p_k, q_k)$, in which points $p_t \in R$ and $q_t \in R$ are the positions for putting the first and last letters of S_t respectively, and
4. a set of available points to put the letters from the k sequences, and
5. a set of $\{H, P\}$ points on R, which already have letters H and P from $S_0[([1, n] - \cup_{t=1}^{k}[i_t, j_t])]$.

Output: a partial self-avoiding arrangement f of S_0 on $S_1, \cdots, S_k$ in the rectangular region R that satisfies $f(i_t) = p_t, f(j_t) = q_t (t = 1, 2, \cdots, k)$, has the maximal number of H-H contacts, and $f(i)$ is an available point for each $i \in \cup_{t=1}^{k}[i_t, j_t]$. H-H contacts may happen between two neighbor available positions, and also between an available and a non-available position after the arrangement.

A hyper-plane $P_{r,a}$ partitions a multi-sequence folding problem F into two multi-sequence folding problems F_1 and F_2 in regions $R \cap P_{r,\leq a}$ and $R \cap P_{r,\geq a}$ respectively by fixing some letters on the $P_{r,a}$ (see Figure 2). Furthermore, the available points of F_1 (F_2) are the intersection of F's available points with $P_{r,<a}$ ($P_{r,>a}$ resp.).

Algorithm

(a) Input d-dimensional multi-sequence folding problem F (as the definition).
(b) For each subset S of $\leq c'(d) \cdot n^{\frac{d-1}{d}}$ letters from $S_1, \cdots, S_k$,
every plane $P_{r,a}$ (with nonempty intersection with R) and
every arrangement of S in available points on $P_{r,a} \cap R$
(c) begin
(d) for each partition (by $P_{r,a}$) making F into problems F_1 and F_2 of size $\leq c(d)n$.
(e) begin
(f) Recursively solve F_1 and F_2.
(g) Merge the solution to F_1 and F_2 to get a potential solution for F.

(h) end
(i) end
(j) Output the solution for F with the maximal number of H-H contacts among all of the potential solutions for F.

End of the Algorithm

Lemma 5. *There is a* $(nm)^{O(n^{1-\frac{1}{d}})}$ *time algorithm for the* d*-dimensional multi-sequence folding problem with a* $m_1 \times m_2 \times \cdots \times m_d$ *rectangular region in the HP-model, where* $m = \max\{\max\{m_i | i = 1, \cdots, d\}, 2\}$ *and the dimensional number* d *is assumed to be a constant.*

Proof. By Lemma 4, the folding problem is partitioned into two problems with a separator of size $\leq c'(d) \cdot n^{1-\frac{1}{d}}$ elements. For each $1 \leq r \leq d$, we have at most m planes $P_{r,a}$ that have a non-empty intersection with the $m_1 \times m_2 \cdots \times m_d$ rectangular region. There are at most $d \cdot m$ ways to select the plane. If the plane has at most t letters, there are at most $d \cdot m \cdot n^t m^{(d-1)t} = dn^t m^{(d-1)t+1}$ ways to select the plane and letters, and put those letters on the plane. So, the loop (c)-(i) is repeated $\leq dn^t m^{(d-1)t+1}$ times.

For disjoint subsequences $S_1, \cdots, S_k$ of S_0 inside a rectangular region R, if we fix $t \leq c'(d) \cdot n^{1-\frac{1}{d}}$ letters from $S_1, \cdots, S_k$ on the hyper plane $P_{r,a}$, they are partitioned into three groups of sub-sequences of S_0 which are in $R \cap P_{r,<a}$, $P_{r,=a}$ and $P_{r,>a}$ respectively (see figure 2). For each sub-sequence from $R \cap P_{r,<a}$ or $R \cap P_{r,>a}$, we fix the positions for its two end points under all possible cases. The sub-sequences in $R \cap P_{r,<a}$ will not affect those in $R \cap P_{r,>a}$. We have at most 2^{t+1} ways to fix the end points of those sequences in $R \cap P_{r,<a}$ and $R \cap P_{r,>a}$. Therefore, the loop (e)-(h) is repeated $\leq 2^{t+1}$ times.

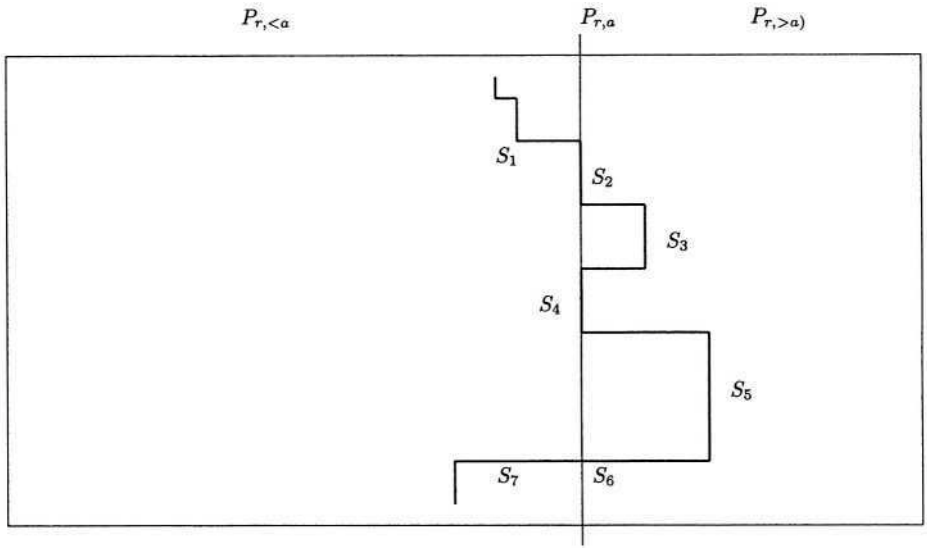


Fig. 2. The hyper-plane $P_{r,a}$ partitions a sequence into 3 groups of disjoint subsequences $\{S_1, S_7\}, \{S_2, S_4, S_6\}$ and $\{S_3, S_5\}$ in $P_{r,<a}, P_{r,a}$ and $P_{r,>a}$ respectively

We have the following recursive relationship for the total time of the algorithm:

$$T(n) \leq 2 \cdot d \cdot m^{(d-1)c'(d)n^{1-\frac{1}{d}}+1} \cdot n^{c'(d)n^{1-\frac{1}{d}}} \cdot 2^{c'(d)n^{1-\frac{1}{d}}+1} \cdot T(c(d)n),$$

where $0 < c(d) < 1$ and $0 < c'(d)$ are constants for fixed d. Expanding the inequality recursively, we have $T(n) = (nm)^{O(n^{1-\frac{1}{d}})}$.

Theorem 6. *There is a $2^{O(n^{1-\frac{1}{d}}\log n)}$ time algorithm for the d-dimensional protein folding in the HP-model for fixed d.*

Proof. The folding problem can be put into a $n \times n \cdots n$ rectangular region in d-dimensional space by fixing the two middle letters in two center neighbor points in the region. By Lemma 5, we have an $n^{O(n^{1-\frac{1}{d}})} = 2^{O(n^{1-\frac{1}{d}}\log n)}$ time algorithm.

3 Improved Separators and Algorithms

The last section shows that the d-dimensional folding problem is computable in $O(2^{e(d)n^{1-\frac{1}{d}}})$ time, where $e(d)$ is constant for fixed d. We will reduce the constant $e(d)$ in this section. Our approach is to improve the separator. The following well known fact Lemma 7 (see [20]) will be used for deriving our new separator. Our reduced upper bound for the number of points on the separator is from the fact below: For a set P of 2-dimensional grid points with the centerpoint o (see Lemma 7), a random line through o has the largest expected number of points of P with distance $\leq a$ to it when the points P are tightly arranged in the grid points inside a circle with the least radius. It is also true in dimension larger than 2.

Lemma 7. *For an n-element set P in d-dimensional space, there is a point q with the property that any half-space that does not contain q, covers at most $\frac{d}{d+1}n$ elements of P. (Such a point q is called a centerpoint of P).*

Definition 8. For a grid point (i,j) on 2-dimensional plane, its *grid square* is a 1×1 square with four corner points $(i-\frac{1}{2}, j-\frac{1}{2}), (i-\frac{1}{2}, j+\frac{1}{2}), (i+\frac{1}{2}, j-\frac{1}{2})$ and $(i+\frac{1}{2}, j+\frac{1}{2})$. A *grid cube* is a $1 \times 1 \times 1$ cube with eight corner points $\{(i+\alpha, j+\beta, k+\gamma) | \alpha, \beta, \gamma \in \{-\frac{1}{2}, \frac{1}{2}\}\}$ for a 3-dimensional grid point (i,j,k).

3.1 2-Dimension

Lemma 9. *(1) A circle of radius r contains at most $\pi(r+\frac{\sqrt{2}}{2})^2$ grid points.*

(2) A circle of radius r on a 2-dimensional plane has at least $\pi r^2 - 4\sqrt{2}\pi r$ grid points inside it.

(3) A circle of radius $\frac{1}{\sqrt{\pi}}\sqrt{n} + 4\sqrt{2}$ has at least n grid points in it.

(4) For every line segment L of length m, the number of grid points with distance $\leq a$ to at least one point of L is $\leq (2a+\sqrt{2})(m+2a+\sqrt{2})$.

(5) For every line L and fixed $a > 0$, there are at most $(2a+\sqrt{2})(\sqrt{2}n+2a+\sqrt{2})$ grid points inside a $n \times n$ square with $\leq a$ distance to L.

Proof. (1) If a grid point p is inside a circle C of radius r at center o, the 1×1 grid square with center at p is inside a circle C' of radius $r + \frac{\sqrt{2}}{2}$ at the same center o. The number of those 1×1 grid squares for the grid points inside C is no more than the area size of the circle C'.

(2) Let C_1, C, and C_2 be three circles on the plane with the same center. Their radii are $r-\sqrt{2}, r$, and $r+\sqrt{2}$ respectively. Every 1×1 grid square intersecting C boundary is outside C_1 and inside C_2. The number of grid squares intersecting C boundary is no more than $\pi(r+\sqrt{2})^2 - \pi(r-\sqrt{2})^2 = 4\sqrt{2}\pi r$.

(3) Let $r = \frac{1}{\sqrt{\pi}}\sqrt{n}+4\sqrt{2}$. It is straightforward to verify that $\pi r^2 - 4\sqrt{2}\pi r > n$.

(4) If a point p has $\leq a$ distance L, every point in the 1×1 grid square with center at p has distance $\leq a + \frac{\sqrt{2}}{2}$ to L. The number of those 1×1 squares with center at points of distance $\leq a$ to L is no more than $2(a + \frac{\sqrt{2}}{2})(m + 2a + \sqrt{2})$.

(5) The length of a line L inside an $n \times n$ square is $\leq \sqrt{2}n$. Apply (4).

Definition 10. Define $Pr_2(a, p_0, p)$ to be the probability that the point p has $\leq a$ perpendicular distance to a random line L through the point p_0.

Lemma 11. *Let $a > 0$ be a constant and $\delta > 0$ be a small constant. Let P be a set of points on 2-dimensional grid. Assume that all points of P are inside a circle of radius r with center at point o. For a random line passing through o, the expected number of points in P with distance $\leq a$ to L is bounded by $4ar + \delta r$ for all large r.*

Proof. Assume $p = (x, y)$ is a point of P and L is a random line passing through the center $o = (x_0, y_0)$. Let C be the circle of radius r and center o such that C covers all points in P. Let C' be the circle of radius $r' = r + \frac{\sqrt{2}}{2}$ and the same center o. It is easy to see every unit square with center at a point in P is inside C'. The probability that a point p has distance $\leq a$ to L is $\frac{2 \arcsin \frac{a}{\mathrm{dist}(o,p)}}{\pi}$.

Let $\epsilon > 0$ be a small constant which will be determined later. Select r_0 to be large enough such that for every point p with $dist(o, p) \geq r_0$, $\arcsin \frac{a}{\mathrm{dist}(o,p)} < (1+\epsilon)\frac{a}{\mathrm{dist}(o,p)}$ and $\frac{1}{\mathrm{dist}(o,p')} < \frac{1+\epsilon}{\mathrm{dist}(o,p)}$ for every point p' with $\mathrm{dist}(p', p) \leq \frac{\sqrt{2}}{2}$. Let P_1 be the set of all points p in P such that $\mathrm{dist}(o, p) < r_0$. By Lemma 9, the number of grid points in P_1 is no more than $\pi(r_0 + \frac{\sqrt{2}}{2})^2$. For each point $p \in P_1$, $Pr_2(a, o, p) \leq 1$. For every point $p \in P - P_1$, $Pr_2(a, o, p) = \frac{2 \arcsin \frac{a}{\mathrm{dist}(o,p)}}{\pi} \leq \frac{(1+\epsilon)2a}{\pi \mathrm{dist}(o,p)}$.

The expected number of points in P with distance $\leq a$ to a random line through the point o is

$$\sum_{p \in P} Pr_2(a, o, p) = \sum_{p \in P_1} Pr_2(a, o, p) + \sum_{p \in P - P_1} Pr_2(a, o, p) \tag{5}$$

$$= \sum_{p \in P_1} 1 + \sum_{p \in P - P_1} \frac{2 \arcsin \frac{a}{\mathrm{dist}(o,p)}}{\pi} \tag{6}$$

$$< \pi(r_0 + \frac{\sqrt{2}}{2})^2 + \sum_{p \in P - P_1} \frac{(1+\epsilon)2a}{\pi \text{dist}(o,p)} \tag{7}$$

$$\leq \pi(r_0 + \frac{\sqrt{2}}{2})^2 + \frac{2a(1+\epsilon)^2}{\pi} \int\int_{C'} \frac{1}{\text{dist}(o,p)} d_x d_y \tag{8}$$

$$= \frac{2a(1+\epsilon)^2}{\pi} \int_0^{2\pi} \int_0^{r'} \frac{\rho}{\rho} d_\rho d_\theta + \pi(r_0 + \frac{\sqrt{2}}{2})^2 \tag{9}$$

$$= 4a(1+\epsilon)^2 r' + \pi(r_0 + \frac{\sqrt{2}}{2})^2 \tag{10}$$

$$< 4ar + \delta r \quad \text{for all large } r \quad \text{by selecting } \epsilon \text{ small enough.} \tag{11}$$

We use the transformation $x = \rho\cos\theta + x_0, y = \rho\sin\theta + y_0$ to convert the integration at 8 to that at 9 above.

Lemma 12. *Let $a > 0$ be a constant and $\epsilon > 0$ be a small constant. For a set P of n grid points in a 2-dimensional grid, there is a line L such that P has at most $(\frac{4a}{\sqrt{\pi}}) \cdot \sqrt{n} + \epsilon\sqrt{n}$ points with distance $\leq a$ to L, and each halfplane divided by L has at most $\frac{2}{3}n$ points from P.*

Proof. Assume that the centerpoint is at the point o (see Lemma 7). We are going to estimate the upper bound for the expected number of points in P, which have $\leq a$ distance to a random line L through o.

Let $r = \frac{1}{\sqrt{\pi}}\sqrt{n} + 4\sqrt{2}$. By Lemma 9, the circle C with center o and radius r contains at least n grid points. Let f be a one-to-one mapping from P to the set of grid points inside C such that $f(p) = p$ for every $p \in P$ with $\text{dist}(o,p) \leq r$. Therefore, f moves those points of P outside the circle C to the inside. It is easy to see that if $\text{dist}(o,p_1) \leq \text{dist}(o,p_2)$ then, $Pr_2(a,o,p_1) \geq Pr_2(a,o,p_2)$. The expected number of points in P with $\leq a$ distance to L is $\sum_{p \in P} Pr_2(a,o,p)$.

By Lemma 11, $\sum_{p \in P} Pr_2(a,o,p) \leq \sum_{p \in P} Pr_2(a,o,f(p)) \leq 4ar + \delta r = \frac{4a}{\pi}\sqrt{n} + \epsilon\sqrt{n}$ by selecting small δ.

It is easy to see that Lemma 12 implies Theorem 2 by setting $a = \frac{1}{2}$. Assume that our input HP-sequence has n_0 letters and the optimal folding is inside a $m \times m$ square. Select a parameter $\epsilon > 0$. Add some points evenly on the four edges of the $m \times m$ square, so that every two neighbor points have distance $\leq \epsilon$. Those points are called *ϵ-regular points*. Every line segment connecting two ϵ-regular points is called a *ϵ-regular line segment*. A ϵ-regular line is a line containing two ϵ-regular points.

Lemma 13. *Let $\epsilon > 0$ be a constant. Every line segment L_1 inside the $m \times m$ square has a ϵ-regular segment L_2 such that for every point $p_1 \in L_1$, there is a point $p_2 \in L_2$ with $\text{dist}(p_1,p_2) \leq \epsilon$, and for every point $q_2 \in L_2$, there is a point $q_1 \in L_1$ with $\text{dist}(q_1,q_2) \leq \epsilon$.*

Proof. Assume E_1, E_2, E_3, and E_4 are the 4 edges of the $m \times m$ square. Assume L_1 intersects two of them inside the square at two points p_i and p_j of edges

E_i and $E_j (i \neq j)$ respectively. Select the ϵ-regular point q_i closest to p_i from the edge E_i, and q_j closet to p_j from E_j. The ϵ-regular line segment L_2 results from connecting q_i and q_j. Every point p in L_1 has another point $p' \in L_2$ with distance $\leq \max(\text{dist}(p_i, q_i), \text{dist}(p_j, q_j)) \leq \epsilon$, and every point q in L_2 has another point in $q' \in L_1$ with distance $\leq \max(\text{dist}(p_i, q_i), \text{dist}(p_j, q_j)) \leq \epsilon$.

Lemma 14. *Let a and ϵ be positive constants. Let P be a set of n points in a 2-dimensional grid. There is a ϵ-regular line L such that there are $\leq (\frac{2}{3} + \epsilon)n$ points of P on each half plane, and $\leq 4(a + \epsilon)\frac{\sqrt{n}}{\sqrt{\pi}}$ points of P to have distance $\leq a$ to L.*

Proof. Let $\delta > 0$ be a small constant. By Lemma 12, there is a line L such that the number of points of P with distance $a + \delta$ to it is bounded by $4(a + \delta)\frac{\sqrt{n}}{\sqrt{\pi}}$, and each side has at most $\frac{2}{3}n$ points in P. By Lemma 13, there is a line L' close to L such that every point in L has another point in L' with distance $\leq \delta$ and every point in L' has another point in L with distance $\leq \delta$. Every point with distance $\leq a$ to the line L' has distance $\leq a + \delta$ to L. Therefore, the number of points in P with distance $\leq a$ to L' is bounded by $4(a + \epsilon)\frac{\sqrt{n}}{\sqrt{\pi}}$, and each half plane divided by L has at most $(\frac{2}{3} + \epsilon)n$ points in P if δ is small enough.

Lemma 15. *For some constants $c_0, \epsilon > 0$, there is a $O(m^{c_0 \log n} n_0^{(6.145-\epsilon)\sqrt{n}})$ time algorithm for the 2D Multi-Sequence Folding Problem F in an $m \times m$ square, where n is the sum of lengths of input disjoint subsequences of S_0, and n_0 is the length of S_0.*

Proof. Let $a = 1/2, c = 2/3 + \delta$, and $d = \frac{4(a+\delta)}{\sqrt{\pi}}$, where $\delta > 0$ is a small constant which will be fixed later. We assume $m > 1$ and n is large. Let P be an optimal arrangement for the problem F. By the Lemma 14, there is a line L such that P has at most $d\sqrt{n}$ points to have distance $\leq 1/2$ to L, and each half plane has at most cn points from P. The letters that stay on those positions with $\leq \frac{1}{2}$ distance to L form a separator for P. For every two letters at different sides of L that have a contact (their distance is 1), at least one of them has distance $\leq \frac{1}{2}$ to L. The algorithm is based on such a separator and is similar to that in the last section to find such a optimal solution P.

The number of δ-regular points at every edge of the $m \times m$ square is bounded by $\frac{m}{\delta}$. The total number of δ-regular lines is bounded by $u_1 = \binom{4}{2}(\frac{m}{\delta})^2$. By Stirling formula, we have $(d\sqrt{n})! > \frac{(d\sqrt{n})^{d\sqrt{n}}}{2^{d\sqrt{n}}}$. There are $u_2 = \binom{n}{0} + \binom{n}{1} + \cdots \binom{n}{d\sqrt{n}} < d\sqrt{n}\frac{n^{d\sqrt{n}}}{(d\sqrt{n})!} < (\frac{2}{d})^{d\sqrt{n}} \cdot d\sqrt{n} \cdot n^{\frac{1}{2}d\sqrt{n}}$ ways to select the $\leq d\sqrt{n}$ letters from the n of them.

Assume fixed k $(\leq d\sqrt{n})$ letters $S_0[i_1], S_0[i_2], \cdots, S_0[i_k] (1 \leq i_1 < i_2 < \cdots < i_k \leq n)$ are from the disjoint subsequences of S_0. By Lemma 9, there are at most $\beta = (2a + \sqrt{2})(\sqrt{2}m + 2a + \sqrt{2})$ positions (inside the $m \times m$ square) to put the letter $S_0[i_1]$ such that it has distance $\leq a$ to L. After the first letter position

is fixed, there are at most $\prod_{j=1}^{j=k-1}(\alpha(i_{j+1}-i_j))$ ways to put the rest of them along the separation line with distance $\leq a$, where $\alpha < (2a+\sqrt{2})(1+2a+\sqrt{2})$ is a constant (by Lemma 9). Since $k \leq d\sqrt{n}$ and $1 \leq i_1 < i_2 < \cdots < i_k \leq n_0$, $\prod_{j=1}^{j=k-1}(\alpha(i_{j+1}-i_j)) \leq (\alpha(\frac{n_0}{k}))^k \leq (\frac{\alpha}{d})^{d\sqrt{n}} n_0^{d\sqrt{n}} n^{-\frac{1}{2}d\sqrt{n}}$ (We use the well known fact that for positive variables $y_1, \cdots, y_k$ and fixed h with $y_1 + \cdots + y_k \leq h$, the product $\prod_{t=1}^{k} y_k$ is maximal when $y_1 = y_2 = \cdots = y_k = h/k$). The number of ways to arrange the k letters along the separation line (with distance $\leq a$ to L) is bounded by $u_3 = \beta(\frac{\alpha}{d})^{d\sqrt{n}} n_0^{d\sqrt{n}} n^{-\frac{1}{2}d\sqrt{n}}$.

We have $T(n) \leq u_1 \cdot u_2 \cdot u_3 \cdot T(cn)$. It implies that $T(n) \leq (\frac{mn}{\delta})^{c_0 \log n} 2^{c_0\sqrt{n}} n_0^{d(\frac{1}{1-\sqrt{c}})\sqrt{n}} = O(m^{c_0 \log n} n_0^{(6.145-\epsilon)\sqrt{n}})$ by selecting constants ϵ, δ small enough, and c_0 large enough.

Theorem 16. *There is a $O(n^{6.145\sqrt{n}})$ time algorithm for the 2D protein folding problem in the HP-model.*

Proof. Fix the two middle letters on the two central neighbor positions of an $n \times n$ square. Let the folding be inside the $n \times n$ square, and apply Lemma 15.

3.2 3-Dimension

The technology used in the last section can be easily extended to the 3-dimensional grid. We give a brief proof for the case in 3-dimensional space.

Lemma 17. *Let $a = \sqrt{3}$. 1)A sphere of radius r has at least $\frac{4}{3}\pi r^3 - \frac{4}{3}\pi(6ar^2 + 2a^3)$ grid points. 2)A sphere of radius $(\frac{3}{4\pi})^{\frac{1}{3}} n^{\frac{1}{3}} + 6a$ contains at least n grid points.*

Proof. 1) Let $r_1 = r + a$, and $r_2 = r - a$. The volume difference between the sphere of radius r_1 and the sphere of radius r_2 is $\frac{4}{3}\pi(6ar^2 + 2a^3)$, which is $\geq$ the number of unit grid cubes intersecting the boundary of the sphere of radius r. 2) For $r = (\frac{3}{4\pi})^{\frac{1}{3}} n^{\frac{1}{3}} + 6a$, we have $\frac{4}{3}\pi r^3 - \frac{4}{3}\pi(6ar^2 + 2a^3) \geq n$.

Definition 18. Define $Pr_3(a, p_0, p)$ to be the probability that the point p has $\leq a$ perpendicular distance to a random plane L through the point p_0 in the 3-dimensional space.

Lemma 19. *Let $a > 0$ be a constant and $\delta > 0$ be a small constant. Let P be a set of points on a 3-dimensional grid. Assume that all points of P are inside a sphere of radius r with center at point o. For a random plane passing through o, the expected number of points in P with distance $\leq a$ to L is bounded by $4ar^2 + \delta r^2$ for all large r.*

Proof. The proof is very similar to that of Lemma 11. Let S be the sphere with radius r and center $o = (x_0, y_0, z_0)$ such that it contains all points in P. Let S' be the sphere of radius $r' = r + \frac{\sqrt{3}}{2}$ and with the same center as S. All of unit cubes with center at points in P are inside S'.

The expected number of points in P with distance $\leq a$ to a random plane through o is $\sum_{p=(x,y,z)\in P} Pr_3(a, o, p)$ which has the main part $\frac{1}{\pi}\int\int\int_{S'} \frac{2a}{\text{dist}(a,o,p)} dx dy dz$. By the transformation $x = \rho\sin\theta\cos\alpha + x_0, y = \rho\sin\theta\cos\alpha + y_0, z = \rho\sin\theta + z_0$, we have $\frac{1}{\pi}\int\int\int_{S'} \frac{2a}{\text{dist}(a,o,p)} dx dy dz = \frac{2}{\pi}\int_0^{r'}\int_0^{\pi}\int_0^{2\pi} \frac{a\rho^2\sin\theta}{\rho} d\alpha d\theta d\rho = 4ar'^2$.

Lemma 20. *Let $a > 0$ be a constant and $\epsilon > 0$ be a small constant. For a set P of n points in a 3-dimensional grid, there is a plane L such that P has at most $(4a(\frac{3}{4\pi})^{2/3}) \cdot n^{2/3} + \epsilon n^{2/3}$ points with distance $\leq a$ to L, and each half space divided by L has at most $\frac{3}{4}n$ points from P.*

Proof. By Lemma 17, the sphere of radius $(\frac{3}{4\pi})^{\frac{1}{3}} n^{\frac{1}{3}} + 6\sqrt{3}$ contains at least n grid points. Moving points of P into to the sphere, which has center at the centerpoint of P (see Lemma 7), from the outside increases the probability to have distance $\leq a$ to a random plane through the sphere center. By Lemma 19, the expected number of points in P with distance $\leq a$ to a random plane is $(4a(\frac{3}{4\pi})^{2/3}) \cdot n^{2/3} + \epsilon n^{2/3}$ for all large n via selecting small δ.

Put some regular points on each side of the six faces of an $m \times m \times m$ cube (the folding region) so that every point on each face has $\leq \epsilon$ distance to one regular point. Those points are called **ϵ-regular** points. Every 3 **ϵ-regular** points determine an **ϵ-regular** plane.

Lemma 21. *Let a and ϵ be positive constants. Let P be a set of n points in a 3-dimensional grid. There is an ϵ-regular plane such that there are $\leq (\frac{3}{4} + \epsilon)n$ points on each side, and $4(a + \epsilon)(\frac{3}{4\pi})^{2/3} n^{2/3}$ points to have distance $\leq a$ to it.*

Proof. Let L be the plane of Lemma 20. Let H be the area of intersection between plane L and the six faces of the $m \times m \times m$-cube that contains all points in P. Let p_1 and p_2 be the two points in H with the maximal distance. Let p_3 be the point in H with the largest perpendicular distance the line p_1p_2. Let p'_1, p'_2 and p'_3 be the δ-regular non-collinear points such that p'_i has distance $\leq \delta$ to p_i for $i = 1, 2, 3$. Use the δ-plane determined by p'_1, p'_2 and p'_3 (by selecting small enough δ).

Lemma 22. *For some positive constant c_0 and $\epsilon > 0$, there is a $O(m^{c_0 \log n} n^{-4.407n^{2/3}} n_0^{(8.813-\epsilon)n^{2/3}})$ time algorithm for the 3-dimensional Multi-Sequence Folding problem in an $m \times m \times m$ cube, where n is the sum of lengths of the input disjoint subsequences of S_0, and n_0 is the length of S_0.*

Proof. Let $a = 1/2, c = 3/4 + \delta$, and $d = 4(a+\delta)(\frac{3}{4\pi})^{2/3}$. As Lemma 15, let $u_1 = \binom{8}{3}(\frac{m}{\delta})^6$, $u_2 = (\frac{2}{d})^{d\sqrt{n}} \cdot d\sqrt{n} \cdot n^{\frac{1}{2}d\sqrt{n}}$ and $u_3 = \beta'(\frac{\alpha'}{d})^{2d\sqrt{n}} n^{-dn^{\frac{2}{3}}} n_0^{2dn^{\frac{2}{3}}}$, where α' and β' are similar to those α and β in the proof of Lemma 15. We have $T(n) \leq u_1 \cdot u_2 \cdot u_3 \cdot T(cn)$. This implies that $T(n) = (mn)^{c_0 \log n} 2^{c_0 n^{\frac{2}{3}}} n^{-(\frac{d}{1-c^{2/3}})n^{2/3}} n_0^{(\frac{2d}{1-c^{2/3}})n^{2/3}}$ for some constant $c_0 > 0$.

Theorem 23. *There is a $O(n^{4.306n^{2/3}})$ time algorithm for the 3-dimensional protein folding problem in the HP-model.*

Proof. Fix the two middle letters on the two central neighbor positions of an $n \times n \times n$ cube. Let the folding be inside the $n \times n \times n$ cube, and apply Lemma 22.

Acknowledgement. We are grateful to Mahdi Abdelguerfi, Padmanabhan Mahadevan and Seth Pincus for the helpful discussions during this research. We also thank the anonymous referees for helpful comments and pointing an error in the earlier version. The first author would also like to thank Chanda Yadavalli for introducing him to the area of bioinformatics.

References

[1] R. Agarwala, S. Batzoglou, V. Dancik, SE. Decatur , S. Hannenhalli, M. Farach, M. Muthukrishnan, S. Skiena, Local rules for protein folding on a triangular lattice and generalized hydrophobicity in the HP model. Journal of Computational Biology 4: 275-296, 1997.

[2] N. Alon, P.Seymour, and R.Thomas, Planar Separator, SIAM J. Discr. Math. 7,2(1990) 184-193.

[3] R. Backofen, Constraint techniques for solving the protein structure prediction problem, Proceedings of 4th International conference on principle and practice of constrain programming, 1998, Lecture Notes in Computer Science, 72-86, Springer-Verlag.

[4] B. Berger and T. Leighton, Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete, Journal of Computational Biology, 5(1998), 27-40.

[5] F. E. Cohen and M. J. E. Sternberg, On the prediction of protein structure: the significance of the root-mean-square deviation, J. Mol. Biol., 138(1980), 321-333.

[6] P. Crescenzi and D. Goldman and C. Papadimitriou and A. Piccolboni and M. Yannakakis,On the complexity of protein folding, Journal of computational biology, 5(1998), 423-465.

[7] U.Bastolla, H. Frauenkron, E. Gerstner, P.Grassberger, and Nadler, Testing a new Monte Carlo algorithm for protein folding, Protein: Structure, Function, and Genetics, 32(1998), 52-66.

[8] A. Godzik and J. Skolnick and A. Kolinski, Regularities in interaction patterns of globular proteins, Protein Engineering, 6(1993), 801-810.

[9] A. Godzik and J. Skonick and A. Kolinski, A topology fingerprint approach to inverse protein folding problem, J. Mol. Biol., 227(1992), 227-238.

[10] W. E. Hart and S. Istrail, Fast protein folding in the hydrophobic-hydrophilic model within three-eights of optimal, Proceedings 27th ACM symposium on the theory of computing, 1995.
[11] L. Holm and C. Sander, Mapping the protein universe, Science, 273(1996), 595-602.
[12] M. Khimasia and P.Coveney, Protein structure prediction as a hard optimization problem: The genetic algorithm approach, In Molecular Simulation, 19(1997), 205-226.
[13] N. Krasnogor, D. Pelta, P.M. Lopez, P.Mocciola, and E. De la Canal, Genetic algorithms for the protein folding problem: A critical view, In C.F.E. Alpaydin, editor, Proceedings of Engineering of Intelligent Systems. ICSC Academic Press, 1998.
[14] K. F. Lau and K. A. Dill, A lattice statistical mechanics model of the conformational and sequence spaces of proteins, Macromolecules, 22(1989), 3986-3997.
[15] K. F. Lau and K. A. Dill, Theory for protein mutability and biogenesis, Proc. Natl. Acad. Sci, 87(1990), 638-642.
[16] R. J. Lipton and R. Tarjan, A separator theorem for planar graph, SIAM J. Appl. Math. 36(1979) 177-189.
[17] F. Liang and W.H. Wong, Evolutionary Monte Carlo for Protein folding simulations, Journal of Chemical Physics, 115,7(2001), 3374-3380.
[18] A. Newman, A new algorithm for protein folding in the HP model, Proceedings 13th ACM-SIAM Symposium on Discrete Algorithms, 2002, 876-884.
[19] A. Patton, W.P.III, and E. Goldman, A standard ga approach to native protein conformation prediction, In Proc 6th Intl Conf Genetic Algorithms, Morgan Kauffman, 1995, 574-581.
[20] J. Pach and P.K. Agarwal, Combinatorial Geometry, Wiley-Interscience Publication, 1995.
[21] A. Piccolboni and G. Mauri, Application of evolutionary algorithms to protein prediction, In N. e. a. Kasabov, editor, Proceedings of I-CONIP'97, Springer, 1998.
[22] A.A. Rabow and H.A. Scheraga, Improved genetic algorithm for the protein folding problem by use of a cartesian combination operator. Protein Science, 5(1996), 1800-1815.
[23] R. Ramakrishnan, B. Ramachandran, and J.F. Pekney, A dynamic Monte Carlo algorithm for exploration of dense conformation spaces in heteropolymers, Journal of Chemical Physics, 106(1997), 2418.
[24] W. D. Smith and N. C. Wormald, Application of geometric separator theorems, FOCS 1998, 232-243.
[25] A. Sali, E. Shakhnovich, M. Karplus, How does a protein fold? Nature, 369(1994), 248-251.
[26] U. Unger and J. Moult, A Genetic algorithm for three dimensional protein folding simulations, In Proc 5th Intl Conf on Genetic Algorithms, 1993, 581-588.
[27] U. Unger and J. Moult, Genetic algorithms for protein folding simulations, Journal of Molecular Biology, 1993, 231(1),75-81
[28] K. Yue and K. A. Dill, Sequence-structure relationships in proteins and copolymers, Physical Review E48(1993), 2267-2278.

Nash Equilibria in Discrete Routing Games with Convex Latency Functions*

Martin Gairing[1], Thomas Lücking[1], Marios Mavronicolas[2], Burkhard Monien[1], and Manuel Rode[1]**

[1] Faculty of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany.
{gairing,luck,bm,rode}@uni-paderborn.de
[2] Department of Computer Science, University of Cyprus, P. O. Box 20537, Nicosia CY-1678, Cyprus.
mavronic@ucy.ac.cy

Abstract. We study *Nash equilibria* in a discrete routing game that combines features of the two most famous models for non-cooperative routing, the *KP model* [16] and the *Wardrop model* [27]. In our model, users share parallel links. A *user strategy* can be any probability distribution over the set of links. Each user tries to minimize its *expected latency,* where the latency on a link is described by an arbitrary non-decreasing, convex function. The social cost is defined as the sum of the users' expected latencies. To the best of our knowledge, this is the first time that *mixed* Nash equilibria for routing games have been studied in combination with *non-linear* latency functions.

As our main result, we show that for identical users the social cost of any Nash equilibrium is bounded by the social cost of the *fully mixed Nash equilibrium.* A Nash equilibrium is called fully mixed if each user chooses each link with non-zero probability. We present a complete characterization of the instances for which a fully mixed Nash equilibrium exists, and prove that (in case of its existence) it is unique. Moreover, we give bounds on the *coordination ratio* and show that several results for the Wardrop model can be carried over to our discrete model.

1 Introduction

Motivation and Framework. One of the most important concepts in non-cooperative game theory is the concept of *Nash equilibria* [22]. A Nash equilibrium is a state of the system in which no player can improve its objective by

* This work has been partially supported by the European Union within the 6th Framework Programme under contract 001907 (DELIS), by the IST Program of the European Union under contract number IST-2001-33116 (FLAGS), by funds from the Joint Program of Scientific and Technological Collaboration between Greece and Cyprus, by research funds at University of Cyprus, and by the VEGA grant No. 2/3164/23.

** International Graduate School of Dynamic Intelligent Systems

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 645–657, 2004.

unilaterally changing its *strategy.* A Nash equilibrium is called *pure* if all players choose exactly one strategy, and *mixed* if players choose probability distributions over strategies. The *coordination ratio* is the worst-case ratio of the *social cost* in a Nash equilibrium state and the minimum social cost. Of special interest to our work is the *fully mixed Nash equilibrium* where each player chooses each strategy with non-zero probability. We consider a hybridization of the two most famous models for non-cooperative routing in literature: the *KP model* [16] and the *Wardrop model* [8,27].

In the KP model, each of n *users* employs a *mixed strategy,* which is a probability distribution over m parallel *links,* to control the shipping of its *traffic.* Traffic is *unsplittable.* A *capacity* specifies the rate at which each link processes traffic. *Identical users* have the same traffic whereas the traffic of the users may vary arbitrarily in the model of *arbitrary users.* In a Nash equilibrium, each user selfishly routes its traffic on links that minimize its *individual cost:* its *expected latency cost,* given the expected network congestion caused by the other users. The *social cost* of a Nash equilibrium is the expectation, over all random choices of the users, of the maximum *latency* through a link (over all links).

In the Wardrop model, *arbitrary* networks with *latency functions* for edges are considered. Moreover, the traffic is *splittable* into arbitrary pieces. Here, unregulated traffic is modeled as a *network flow. Equilibrium flows* are flows with all paths used between a given pair of a *source* and a *destination* having the same latency. The latency functions are convex. Thus, equilibrium flows are optimal solutions to a convex program. An equilibrium in this model can be interpreted as a Nash equilibrium in a game with an infinite number of users, each carrying an infinitesimal amount of traffic from a *source* to a *destination.* The Wardrop model restricts to pure Nash equilibria. The *individual cost* of a user is the sum of the edge *latencies* on a path from the user's source to the its destination. The *social cost* of a Nash equilibrium is the sum of all individual costs.

The routing model considered in this work combines aspects of both the KP model and the Wardrop model. First, we restrict the network structure to that of the KP model (parallel links) and we assume a user's traffic to be unsplittable. On the other hand, we allow arbitrary non-decreasing and convex latency functions, whereas in the KP model latency functions are linear. In our model, the latency function of a link is a function in the total traffic of users assigned to this link. The social cost is defined as the expected sum of all user costs – as opposed to the social cost used in the KP model. Thus, as far as the generality of latency functions and the definition of social cost are concerned, we lean toward the Wardrop model, whereas the network structure and the indivisibility of each user's traffic remain as in the KP model. Restricted to *pure* Nash equilibria, our model has already been studied in [6], and restricted to *linear* latency functions in [18]. It is a particular instance of what is known as *congestion game* [21,23]. It is known that a pure Nash equilibrium always exists in this setting.

The main results of this work are the identification of the worst-case mixed Nash equilibrium and bounds on the coordination ratio. The convex latency func-

tions define a very general discrete routing game. To the best of our knowledge this is the first time that mixed Nash equilibria are studied in such a game.

Related Work. The KP model was introduced by Koutsoupias and Papadimitriou [16]. They introduced the notion of coordination ratio and analyzed the coordination ratio for some special cases. Later, Czumaj and Vöcking [7], and Koutsoupias et al. [15] gave asymptotically tight upper bounds on the coordination ratio for pure and mixed Nash equilibria. Mavronicolas and Spirakis [20] studied further the KP model and introduced the fully mixed Nash equilibrium. They showed that, in case it exists, the fully mixed Nash equilibrium is unique. Gairing et al. [12] conjecture that the fully mixed Nash equilibrium, whenever it exists, has the worst social cost among all Nash equilibria. From here on we will refer to this as the *Fully Mixed Nash Equilibrium Conjecture.* Up to now, the conjecture could be proven only for several particular cases of the KP model [12,19]. A proof of the conjecture will enable the derivation of upper bounds on the coordination ratio via studying the fully mixed Nash equilibrium.

Lücking et al. [18] considered the KP model with respect to *quadratic social cost,* defined as the sum of weighted individual costs. In this context, they proved the Fully Mixed Nash Equilibrium Conjecture in the case of identical users and identical links. This result is strongly related to results presented in this paper.

A natural problem is the effective computation of a Nash equilibrium. For general strategic games, it is still open as to whether a Nash equilibrium can be computed in polynomial time, even for two player games. Fotakis et al. [11] showed that a pure Nash equilibrium for the KP model can be computed in polynomial time using Graham's algorithm [13]. Furthermore, they proved that the problem to compute the best or worst pure Nash equilibrium is $\mathcal{NP}$-complete. Feldmann et al. [9] showed that any deterministic assignment of users to links can be transformed into a Nash equilibrium in polynomial time without increasing the social cost. In particular, combining this result with known approximation algorithms for the computation of optimal assignments [14] yields a PTAS for the problem to compute a best pure Nash equilibrium.

The Wardrop model was already studied in the 1950's [2,27], in the context of road traffic systems. Wardrop [27] introduced the concept of equilibrium to describe user behavior in this kind of traffic networks. For a survey of the early work on this model, see [3]. A lot of subsequent work on this model has been motivated by Braess's Paradox [5]. Inspired by the new interest in the coordination ratio, Roughgarden and Tardos [24,25,26] re-investigated the Wardrop model. For a survey of results, we refer to [10] and references therein.

Results. With our methods, we can only prove results for identical users. However, for this case we obtain through a very thorough analysis the following

- In the case of its existence, the fully mixed Nash equilibrium is the worst-case Nash equilibrium for any instance with convex latency functions. Therewith, we prove the Fully Mixed Nash Equilibrium Conjecture to hold for the model under consideration, whereas it remains unproven for the KP model in the general case. This broadens some recent results from [18] for a special case

of our model, where latency functions are restricted to be linear. We use an appropriate counterexample to show that the convexity assumption we are making for the latency functions cannot be relaxed.
- For arbitrary non-decreasing and non-constant latency functions, the fully mixed Nash equilibrium is unique in the case of its existence.
- We give a complete characterization of instances for which the fully mixed Nash equilibrium exists.
- For pure Nash equilibria we adapt an upper bound on the coordination ratio from Roughgarden and Tardos [26] to our (discrete) model. This bound holds for non-decreasing and non-constant latency functions. Considering polynomial latency functions with non-negative coefficients and of maximum degree d, this yields an upper bound of $d+1$.
- For identical links with latency function $f(x) = x^d$, $d \in \mathbb{N}$, the coordination ratio for mixed Nash equilibria is bounded by the $(d+1)$*'th Bell number.* This bound can be approximated arbitrarily but never reached.
- We give a $\mathcal{O}(m \log n \log m)$ algorithm to compute a pure Nash equilibrium for non-decreasing latency functions.
- For arbitrary users, computing the best-case or worst-case pure Nash equilibrium is $\mathcal{NP}$-hard even for identical links with a linear latency function.

Road Map. Section 2 introduces notations and terminology. In Section 3, the Fully Mixed Nash Equilibrium Conjecture is proven for the model we consider. The necessity of the convexity assumption is also established there. Furthermore, we determine the conditions under which the fully mixed Nash equilibrium exists. Section 4 presents bounds on coordination ratio and complexity results.

2 Discrete Routing Games

General. The number of ways a set of k elements can be partitioned into non-empty subsets is called the k-th *Bell Number* [4,28], denoted by B_k. It is defined by the recursive formula $B_0 = 1$ and

$$B_{k+1} = \sum_{0 \le q \le k} B_q \cdot \binom{k}{q} \quad \text{for all } k \ge 0. \tag{1}$$

Throughout, denote for any integer $m \ge 1$, $[m] = \{1, \ldots, m\}$.

We consider a *network* consisting of a set of m parallel *links* $1, 2, \ldots, m$ from a *source* node to a *destination* node. Each of n *network users* $1, 2, \ldots, n$, or *users* for short, wishes to route a particular amount of traffic along a (non-fixed) link from source to destination. Denote as w_i the *traffic* of user $i \in [n]$. Define the $n \times 1$ *traffic vector* $\mathbf{w}$ in the natural way. For any subset $A \subseteq [n]$ of users, denote $w_A = \sum_{i \in A} w_i$. If users are *identical,* we assume that $w_i = 1$ for all $i \in [n]$. In this case, w_A reduces to $|A|$. Assume throughout that $m > 1$ and $n > 1$.

A *pure strategy* for user $i \in [n]$ is some specific link. A *mixed strategy* for user $i \in [n]$ is a probability distribution over pure strategies; thus, a mixed strategy is a probability distribution over the set of links. The *support* of the mixed strategy

for user $i \in [n]$, denoted as $\mathsf{support}(i)$, is the set of those pure strategies (links) to which i assigns positive probability. A *pure strategy profile* is represented by an n-tuple $\langle \ell_1, \ell_2, \ldots, \ell_n \rangle \in [m]^n$; a *mixed strategy profile* is represented by an $n \times m$ *probability matrix* $\mathbf{P}$ of nm probabilities $p(i,j)$, $i \in [n]$ and $j \in [m]$, where $p(i,j)$ is the probability that user i chooses link j.

For a probability matrix $\mathbf{P}$, define *indicator variables* $I(i,j) \in \{0,1\}$, where $i \in [n]$ and $j \in [m]$, such that $I(i,j) = 1$ if and only if $p(i,j) > 0$. Thus, the support of the mixed strategy for user $i \in [n]$ is the set $\{j \in [m] \mid I(i,j) = 1\}$. A mixed strategy profile $\mathbf{P}$ is *fully mixed* [20, Section 2.2] if for all users $i \in [n]$ and links $j \in [m]$, $I(i,j) = 1$. Throughout, we will cast a pure strategy profile as a special case of a mixed strategy profile in which all strategies are pure.

System, Models and Cost Measures. Associated with every link $j \in [m]$, is a latency function $f_j : \mathbb{R}_0^+ \to \mathbb{R}_0^+$, $f_j(0) = 0$, which is non-decreasing and non-constant. Define the $m \times 1$ *vector of latency functions* Φ in the natural way. If $f_j = f$ for all $j \in [m]$, we say that the links are *identical,* otherwise they are *arbitrary.* For a pure strategy profile $\langle \ell_1, \ell_2, \ldots, \ell_n \rangle$, the *individual latency cost for user* $i \in [n]$, denoted by λ_i, is defined by $f_j(\sum_{k \in [n] : \ell_k = j} w_k)$, with $j = \ell_i$. For a mixed strategy profile $\mathbf{P}$, denote as Λ_j the *expected latency* on link $j \in [m]$, i.e.

$$\Lambda_j = \sum_{A \subseteq [n]} \prod_{k \in A} p(k,j) \cdot \prod_{k \notin A} (1 - p(k,j)) \cdot f_j(w_A).$$

The *expected latency cost* for user $i \in [n]$ on link $j \in [m]$, denoted by λ_{ij}, is the expectation, over all random choices of the remaining users, of the individual latency cost for user i had its traffic been assigned to link j; thus,

$$\begin{aligned} \lambda_{ij} &= \sum_{\langle \ell_1, \ldots, \ell_n \rangle} \prod_{k \in [n] \setminus \{i\}} p(k, \ell_k) \cdot f_j(w_i + \sum_{\substack{k \in [n] \setminus \{i\} \\ \ell_k = j}} w_k) \\ &= \sum_{A \subseteq [n] \setminus \{i\}} \prod_{k \in A} p(k,j) \prod_{k \notin A \cup \{i\}} (1 - p(k,j)) \cdot f_j(w_i + w_A). \end{aligned}$$

For each user $i \in [n]$, the *expected individual latency cost,* denoted by λ_i, is the expectation, over all links $j \in [m]$, of the expected latency cost for user i on link j; thus, $\lambda_i = \sum_{j \in [m]} p(i,j) \cdot \lambda_{ij}$. Associated with a mixed strategy profile $\mathbf{P}$ and a vector of latency functions Φ is the *social cost,* denoted by $\mathsf{SC}^{\Sigma}(\Phi, \mathbf{P})$, which is the sum, over all users, of the expected individual latency costs of the users. Thus, $\mathsf{SC}^{\Sigma}(\Phi, \mathbf{P}) = \sum_{i \in [n]} \lambda_i$. On the other hand, the *social optimum,* denoted by $\mathsf{OPT}^{\Sigma}(\Phi)$, is the least possible value, over all pure strategy profiles $\mathbf{L}$, of the social cost. Thus, $\mathsf{OPT}^{\Sigma}(\Phi) = \min_{\mathbf{L}} \mathsf{SC}^{\Sigma}(\Phi, \mathbf{L})$.

Nash Equilibria and Coordination Ratio. We are interested in a special class of mixed strategies called Nash equilibria [22] that we describe below. Say that a user $i \in [n]$ is *satisfied for the probability matrix* $\mathbf{P}$ if $\lambda_{ij} = \lambda_i$ for all links $j \in \mathsf{support}(i)$, and $\lambda_{ij} \geq \lambda_i$ for all $j \notin \mathsf{support}(i)$. Otherwise, user i is *unsatisfied.* Thus, a satisfied user has no incentive to unilaterally deviate from its mixed

strategy. $\mathbf{P}$ is a *Nash equilibrium* [16, Section 2] if and only if all users $i \in [n]$ are satisfied for $\mathbf{P}$. The *coordination ratio* is the maximum value, over all vectors of latency functions $\boldsymbol{\Phi}$ and Nash equilibria $\mathbf{P}$, of the ratio $\mathsf{SC}^{\Sigma}(\boldsymbol{\Phi}, \mathbf{P})/\mathsf{OPT}^{\Sigma}(\boldsymbol{\Phi})$.

3 Results on Fully Mixed Nash Equilibria

For the model of identical users, we now consider fully mixed Nash Equilibria. We start with a definition and a technical lemma. Both can be proven for the model of arbitrary users, and are useful several times throughout the paper.

Definition 1. *For a vector of r probabilities $p = (p_1, \ldots, p_r)$ and a function $g : \mathbb{R} \to \mathbb{R}$ define*

$$H(p, \mathbf{w}, g) = \sum_{A \subseteq [r]} \prod_{k \in A} p_k \prod_{k \notin A} (1 - p_k) \cdot g(w_A).$$

In the same way, we define a function $\tilde{H}(\tilde{p}, r, \mathbf{w}, g)$ by replacing p with a vector of r probabilities all equal to $\tilde{p}$. In the case that all users have the same traffic, we omit $\mathbf{w}$ in the parameter list. Note that w_A reduces to $|A|$ in this case.

We prove a natural monotonicity property of the function $H(p, \mathbf{w}, g)$.

Lemma 1. *For every vector of r probabilities $p = (p_1, \ldots, p_r)$ and every non-decreasing and non-constant function $g : \mathbb{R} \to \mathbb{R}$, $H(p, \mathbf{w}, g)$ is strictly increasing in each probability $p_i, \forall i \in [r]$.*

Proof. We prove, that $H(p, \mathbf{w}, g)$ is strictly increasing in p_r. The lemma then follows by symmetry of $H(p, \mathbf{w}, g)$ in all probabilities $p_i, i \in [r]$. It is

$$\begin{aligned} H(p, \mathbf{w}, g) &= \sum_{A \subseteq [r]} \prod_{k \in A} p_k \prod_{k \notin A} (1 - p_k) \cdot g(w_A) \\ &= \sum_{A \subseteq [r-1]} \prod_{k \in A} p_k \prod_{k \notin A \cup \{r\}} (1 - p_k) \cdot [g(w_A) + p_r \cdot (g(w_A + w_r) - g(w_A))] \end{aligned}$$

As $g(w_A + w_r) - g(w_A) \geq 0$ for all $A \subseteq [r-1]$ (g is non-decreasing), and $g(w_A + w_r) - g(w_A) > 0$ for some $A \subseteq [r-1]$ (g is non-constant), the claim follows. □

3.1 The Worst-Case Nash Equilibrium

We now focus on the Fully Mixed Nash Equilibrium Conjecture. We first show that for an arbitrary Nash equilibrium $\mathbf{P}$, the expected latency of a user i on a link j increases if we set all user probabilities on link j to be the average probability on that link. We then use this result to show that the expected individual latency of user i in the Nash equilibrium $\mathbf{P}$ is at most its expected individual latency in the fully mixed Nash equilibrium. By definition, this proves the Fully Mixed Nash Equilibrium Conjecture for our model. We furthermore give an example with a strictly increasing but non-convex latency function for which the Fully Mixed Nash Equilibrium Conjecture does not hold, showing that the assumption of convexity for the latency functions is essential.

Lemma 2. *Let g be convex and define $p = (p_1, \ldots, p_n)$ and $\tilde{p} = \frac{\sum_{i\in[n]} p_i}{n}$. Then $H(p,g) \leq \tilde{H}(\tilde{p}, n, g)$.*

Proof. Define a set of n probabilities $q = (q_1, \ldots, q_n)$ by $q_1 = q_2 = \frac{p_1+p_2}{2}$ and $q_i = p_i$, $\forall i \in [3,n]$. Then

$$H(p,g) = \sum_{A\subseteq[3,n]} \prod_{k\in A} p_k \prod_{k\notin A\cup\{1,2\}} (1-p_k) \cdot F(|A|, p, g),$$

where

$$\begin{aligned} F(|A|,p,g) = p_1 \cdot p_2 \cdot [g(|A|+2) - 2g(|A|+1) + g(|A|)] \\ + (p_1+p_2) \cdot [g(|A|+1) - g(|A|)] + g(|A|). \end{aligned}$$

Similarly,

$$H(q,g) = \sum_{A\subseteq[3,n]} \prod_{k\in A} q_k \prod_{k\notin A\cup\{1,2\}} (1-q_k) \cdot F(|A|, q, g).$$

It suffices to show, that $F(|A|,q,g) - F(|A|,p,g) \geq 0$. Indeed,

$$\begin{aligned} F(|A|,q,g) - F(|A|,p,g) &= (q_1 \cdot q_2 - p_1 \cdot p_2) \cdot [g(|A|+2) - 2g(|A|+1) + g(|A|)] \\ &\quad + (q_1+q_2-(p_1+p_2)) \cdot [g(|A|+1) - g(|A|)] \\ &= \left(\frac{p_1-p_2}{2}\right)^2 [g(|A|+2) - 2g(|A|+1) + g(|A|)] \geq 0, \end{aligned}$$

since g is convex. □

Lemma 3. *Consider the model of identical users and arbitrary links with non-decreasing, non-constant and convex latency functions. If there exists a fully mixed Nash equilibrium* $\mathbf{F}$, *then for every mixed Nash equilibrium* $\mathbf{P}$, $\lambda_i(\mathbf{P}) \leq \lambda_i(\mathbf{F})$ *for all* $i \in [n]$.

Proof. Define $\Theta_{ij} = \sum_{k\in[n],k\neq i} p(k,j)$ and $\tilde{p}(j) = \frac{\Theta_{ij}}{n-1}$. The claim holds if $\lambda_{ij}(\mathbf{P}) \leq \lambda_i(\mathbf{F})$, for all $i \in [n], j \in [m]$. So assume there exists $i \in [n]$ and $j \in [m]$ with $\lambda_{ij}(\mathbf{P}) > \lambda_i(\mathbf{F})$. By Lemma 2

$$\lambda_{ij}(\mathbf{P}) \leq \sum_{A\subseteq[n]\setminus\{i\}} \prod_{k\in A} \tilde{p}(j) \prod_{k\notin A\cup\{i\}} (1-\tilde{p}(j)) \cdot f_j(1+|A|), \quad \text{and}$$

$$\lambda_{ij}(\mathbf{F}) = \sum_{A\subseteq[n]\setminus\{i\}} \prod_{k\in A} p_F(j) \prod_{k\notin A\cup\{i\}} (1-p_F(j)) \cdot f_j(1+|A|),$$

where $p_F(j)$ is the probability for any user to choose link j in the fully mixed Nash equilibrium $\mathbf{F}$. Note that the upper bound on $\lambda_{ij}(\mathbf{P})$ is strictly increasing in $\tilde{p}(j)$, since f_j is non-decreasing and non-constant. Therefore, $\lambda_{ij}(\mathbf{P}) > \lambda_i(\mathbf{F})$ implies that $\tilde{p}(j) > p_F(j)$. Since $\sum_{j\in[m]} \tilde{p}(j) = \sum_{j\in[m]} p_F(j) = 1$, there exists a link k with $\tilde{p}(k) < p_F(k)$. However, this implies that $\lambda_{ik}(\mathbf{P}) < \lambda_i(\mathbf{F})$ and thus $\lambda_i(\mathbf{P}) < \lambda_i(\mathbf{F})$. □

If we look at the different model where latency functions only depend on the user and not on the link, we know that there exists a fully mixed Nash equilibrium with probabilities $p(i,j) = \frac{1}{m}$ for all $i \in [n]$ and $j \in [m]$. With the same method as in Lemma 3, we can prove that the expected individual latency of a user is bounded by its expected individual latency of this fully mixed Nash equilibrium.

Theorem 1. *Consider the model of identical users and arbitrary links with non-decreasing, non-constant and convex latency functions. If the fully mixed Nash equilibrium* $\mathbf{F}$ *exists, then for every mixed Nash equilibrium* $\mathbf{P}$, $\mathsf{SC}^{\Sigma}(\Phi, \mathbf{P}) \leq \mathsf{SC}^{\Sigma}(\Phi, \mathbf{F})$.

Proof. Follows from the definition of $\mathsf{SC}^{\Sigma}(\Phi, \mathbf{P})$ combined with Lemma 3. □

The Fully Mixed Nash Equilibrium Conjecture has been proven for the model of identical users, identical links and latency function $f(x) = x$ by Lucking et al. [18]. Theorem 1 generalizes this result to non-decreasing, non-constant and convex latency functions. We continue to prove that the convexity assumption is essential.

Proposition 1. *There exists an instance with identical users, identical links and a non-decreasing, non-convex latency function with a pure Nash equilibrium* $\mathbf{L}$ *and fully mixed Nash equilibrium* $\mathbf{F}$ *such that* $\lambda_i(\mathbf{L}) > \lambda_i(\mathbf{F})$ *for all* $i \in [n]$.

Proof. Consider an instance with $m = 2$ links and $n = 4$ users. Define f as follows: $f(1) = 1, f(2) = 2, f(3) = 2 + \epsilon, f(4) = 2 + 2\epsilon$, where $\epsilon > 0$. Then in each pure Nash equilibrium, there are exactly 2 users on each link. Let $\mathbf{L}$ be such a pure Nash equilibrium. Then $\lambda_i(\mathbf{L}) = 2$ for all $i \in [n]$. Now consider the fully mixed Nash equilibrium $\mathbf{F}$. Here $p(i,j) = \frac{1}{2}$ for all $i \in [n], j \in [m]$. Thus,

$$\lambda_i(\mathbf{F}) = \frac{1}{8}(f(1) + 3f(2) + 3f(3) + f(4)) = \frac{15}{8} + \frac{5\epsilon}{8}, \quad \forall i \in [n].$$

For $\epsilon < \frac{1}{5}$ it follows that $\lambda_i(\mathbf{L}) > \lambda_i(\mathbf{F})$ for all $i \in [n]$. □

3.2 Uniqueness of the Fully Mixed Nash Equilibrium

We first show that the probabilities of all users on a certain link are identical in a fully mixed Nash equilibrium. We then use this fact to establish uniqueness of the fully mixed Nash equilibrium.

Theorem 2 (Uniqueness of the Fully Mixed Nash Equilibrium). *Consider the model of identical users and arbitrary links with non-decreasing and non-constant latency functions. If a fully mixed Nash equilibrium* $\mathbf{F}$ *exists, then it is unique.*

3.3 Existence of Fully Mixed Nash Equilibrium

For the special case where all latency functions are equal, i.e. $f_j = f$ for all $j \in [m]$, a fully mixed Nash equilibrium always exists and has probabilities

$p(i,j) = \frac{1}{m}$ for all $i \in [n]$, $j \in [m]$. For the general case, the existence of the fully mixed Nash equilibrium is not granted, but depends on the latency functions f_j. We will now shed light on this dependence. Without loss of generality, assume the links to be ordered non-decreasingly according to $f_j(1)$. Let $g_j : [n-1] \cup \{0\} \to \mathbb{R}$ be defined by $g_j(x) = f_j(x+1)$ for all $j \in [m]$. For $k \in [m]$, $j \in [k-1]$, determine $p_j(k)$, such that $\tilde{H}(p_j(k), n-1, g_j) = f_k(1)$. Then, $\tilde{H}(p_j(k), n-1, g_j)$ is the expected individual latency of any user on link j, if $p(i,j) = p_j(k)$ for all $i \in [n]$. Note, that due to Lemma 1, $\tilde{H}(p_j(k), n-1, g_j)$ is strictly increasing in $p_j(k)$, and hence $p_j(k)$ is uniquely determined.

Definition 2. *Links k with $\sum_{j \in [k-1]} p_j(k) > 1$ are called dead links. Links k with $\sum_{j \in [k-1]} p_j(k) = 1$ are called special links.*

Lemma 4. *Consider the model of identical users and arbitrary links with non-decreasing and non-constant latency functions. If $j \in [m]$ is a dead link, then in any Nash equilibrium* **P**, *$p(i,j) = 0$ for all $i \in [n]$.*

Lemma 5. *Consider the model of identical users and arbitrary links with non-decreasing and non-constant latency functions. Let S be the set of special links. In any Nash equilibrium* **P**, *there exists at most one user i with $p(i,j) > 0$ for some $j \in S$.*

Theorem 3 (Characterization of Fully Mixed Nash Equilibria). *Consider the model of identical users and arbitrary links with non-decreasing and non-constant latency functions. There exists a fully mixed Nash equilibrium, if and only if there are no special and no dead links.*

Theorem 3 implies that if the fully mixed Nash equilibrium does not exist, then the instance contains dead or special links. But dead links are never used in any Nash equilibrium and could be removed from the instance. We now broaden the result from Theorem 3 by giving an upper bound on the social cost in the case that the fully mixed Nash equilibrium does not exist.

Theorem 4. *Consider an instance with special or dead links. Then the social cost of any Nash equilibrium* **P** *is bounded by the social cost of the fully mixed Nash equilibrium* **F** *for the instance where the links are restricted to the non-special and non-dead links.*

4 Coordination Ratio and Complexity Results

4.1 Bounds on Coordination Ratio for Special Latency Functions

We now consider the model of identical users and identical links with latency function $f(x) = x^d$, $d \in \mathbb{N}$. In this model, every pure Nash equilibrium has optimal social cost. For mixed Nash equilibria, we now show that the coordination ratio is bounded by the $(d+1)$-th *Bell Number* B_{d+1} (see Equation (1)). Due to [17], $B_{d+1} \approx (d+1)^{-\frac{1}{2}} [\gamma(d+1)]^{d+1+\frac{1}{2}} e^{\gamma(d+1)-d-2}$, where the function $\gamma(d+1)$ is defined implicitly by $\gamma(d+1) \cdot \ln(\gamma(d+1)) = d+1$.

Theorem 5. *Consider the model of identical users and identical links with latency function* $f(x) = x^d$, $d \in \mathbb{N}$. *Then,*

$$\sup_{w,P} \frac{SC^{\Sigma}(\Phi, P)}{OPT^{\Sigma}(\Phi)} = B_{d+1}.$$

4.2 Bounds on Coordination Ratio for General Latency Functions

In this section, we carry over an upper bound from Roughgarden and Tardos [26, Corollary 2.10] on the coordination ratio for splittable flows and continuous latency functions to our discrete setting. For the proof, which is a straightforward adaption of the corresponding proof in [26], we make use of the following lemma.

Lemma 6. *Let* $g_j : [n] \to \mathbb{R}$ *be a convex function for* $j \in [m]$. *Set* $X = \{x = (x_1, \ldots, x_m) \in \mathbb{N}_0^m \mid \sum_{j\in[m]} x_j = n\}$. *Then* $\sum_{j\in[m]} g_j(x_j)$ *is minimum among all* $x = (x_1, \ldots, x_m) \in X$, *if and only if*

$$g_j(x_j + 1) + g_k(x_k - 1) \geq g_j(x_j) + g_k(x_k) \quad \forall j, k \in [m].$$

Lemma 6 can be shown by the application of convex cost flows [1, Chapter 14].

Lemma 7. *Consider the model of identical users and arbitrary links with non-decreasing and non-constant latency functions. If* $x f_j(x) \leq \alpha \sum_{t=1}^{x} f_j(t)$ *for all* $j \in [m]$, *then the social cost of any pure Nash equilibrium is bounded by* $\alpha OPT^{\Sigma}(\Phi)$.

The following corollary is an example for the application of the upper bound.

Corollary 1. *Consider the model of identical users and arbitrary links. If latency functions are polynomials with non-negative coefficients and maximum degree* d, *then the coordination ratio for pure Nash equilibria is bounded by* $d + 1$.

4.3 Computation of Pure Nash Equilibrium and Optimum

In the model of identical users and identical links, the users are evenly distributed to the links in every pure Nash equilibrium, and every pure Nash equilibrium has optimum social cost. In the following, we give an algorithm to compute a pure Nash equilibrium in the model of identical users but arbitrary non-decreasing latency functions. A simple approach is to assign the users one by one to their respective best link. This greedy algorithm, also known as Graham's algorithm, can be implemented with running time $\mathcal{O}((n + m) \log m)$ if the links are kept in a priority queue according to their latency after the assignment of the next user.

ALGORITHM 1
Input: n and any assignment $x_1, \ldots, x_m$
Output: Nash Equilibrium $x_1, \ldots, x_m$

for $\delta = n, \lceil \frac{n}{2} \rceil, \lceil \frac{n}{4} \rceil, \ldots, 1$ **do**
  let t be such that $f_t(x_t + \delta)$ is minimum;
  while $\exists s \in [n]$ with $x_s \geq \delta$ and $f_s(x_s) > f_t(x_t + \delta)$ **do**
    let $s \in [m]$ be such that $x_s \geq \delta$ and
      $f_s(x_s)$ is maximum w.r.t. this requirement;
    $x_s = x_s - \delta$; $x_t = x_t + \delta$;
    let t be such that $f_t(x_t + \delta)$ is minimum;

Our algorithm takes time $\mathcal{O}(m \log n \log m)$, which is better if $m = o(\frac{n}{\log n})$. The algorithm takes as input an arbitrary initial assignment of users to links given by $x_1, \ldots, x_m$, where x_j is the number of users on link j. It transforms this assignment into a Nash equilibrium by moving chunks of users at a time. The first chunk contains all users. In each phase the chunk size is cut in half until a chunk consists of one user only. In the sequel we refer to x_j as the *load* on link $j \in [m]$.

Proposition 2. *Consider the model of identical users and arbitrary links with non-decreasing latency functions. Then Algorithm 1 computes a pure Nash equilibrium in time* $\mathcal{O}(m \log n \log m)$.

The following lemma shows that we can compute an optimal pure assignment in the same way as a Nash equilibrium, but according to other latency functions. A corresponding result holds for the case of continuous latency functions and splittable flows (see e.g. [26]).

Lemma 8. *Consider an instance of the routing model with identical users and* m *links with latency function* $f_j(x)$ *on link* j *for* $j \in [m]$, *such that* $xf_j(x)$ *is convex. Set* $h_j(x) = xf_j(x) - (x-1)f_j(x-1)$. *Let* $\mathbf{L}$ *be any pure strategy profile.* $\mathbf{L}$ *is an optimal assignment with respect to latency functions* f_j, *if and only if* $\mathbf{L}$ *is a Nash equilibrium with respect to latency functions* h_j.

Due to Lemma 8, Algorithm 1 can be used to compute an optimal pure assignment by applying it to the instance with latency functions h_j on link j.

4.4 Complexity Results

Fotakis et al. [11] proved that computing the best-case or worst-case pure Nash equilibrium in the KP model is $\mathcal{NP}$-hard. Keep in mind that in the KP model the social cost of a pure Nash equilibrium is the maximum latency on a link, whereas in our model the social cost is the sum of the individual latency costs. We now show that computing the best-case or the worst-case pure Nash equilibrium in our model is also NP-hard even for identical links with latency function $f(x) = x$.

Proposition 3. *Consider the model of arbitrary users and identical links with latency function* $f(x) = x$. *Then, computing the best-case or the worst-case pure Nash equilibrium is NP-hard.*

It is easy to see that Graham's algorithm [13] (known to work for the KP model [11]) still works for the model under consideration to compute a pure Nash equilibrium in polynomial time.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications.* Prentice Hall, 1993.
2. M. Beckmann, C.B. McGuire, and C.B. Winsten. *Studies in the Economics of Transportation.* Yale University Press, 1956.
3. M.J. Beckmann. On the theory of traffic flow in networks. *Traffic Quart,* 21:109–116, 1967.
4. E.T. Bell. Exponential numbers. *American Mathematical Monthly,* 41(7):411–419, 1934.
5. D. Braess. Über ein Paradoxon der Verkehrsplanung. *Unternehmensforschung,* 12:258–268, 1968.
6. A. Czumaj, P. Krysta, and B. Vöcking. Selfish traffic allocation for server farms. In *Proc. of the 34th Ann. ACM Symp. on Theory of Computing,* pp. 287–296, 2002.
7. A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. In *Proc. of the 13th Ann. ACM-SIAM Symp. on Discrete Algorithms,* pp. 413–420, 2002.
8. S.C. Dafermos and F.T. Sparrow. The traffic assignment problem for a general network. *Journal of Research of the National Bureau of Standards - B. Mathematical Sciences,* 73B(2):91–118, 1969.
9. R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Nashification and the coordination ratio for a selfish routing game. In *Proc. of the 30th Int. Colloq. on Automata, Languages, and Programming,* LNCS 2719, pp. 514–526, 2003.
10. R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Selfish routing in non-cooperative networks: A survey. In *Proc. of the 28th Int. Symp. on Mathematical Foundations of Computer Science,* LNCS 2747, pp. 21–45, 2003.
11. D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. In *Proc. of the 29th Int. Colloq. on Automata, Languages, and Programming,* LNCS 2380, pp. 123–134, 2002.
12. M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and P. Spirakis. Extreme nash equilibria. In *Proc. of the 8th Italian Conference on Theoretical Computer Science,* LNCS 2841, pp. 1–20, 2003. Also accepted to *Theoretical Computer Science,* Special Issue on *Game Theory Meets Theoretical Computer Science.*
13. R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics,* 17(2):416–429, 1969.
14. D.S. Hochbaum and D. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing,* 17(3):539–551, 1988.
15. E. Koutsoupias, M. Mavronicolas, and P. Spirakis. Approximate equilibria and ball fusion. *Theory of Computing Systems,* 36(6):683–693, 2003.
16. E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proc. of the 16th Int. Symp. on Theoretical Aspects of Computer Science,* LNCS 1563, pp. 404–413, 1999.
17. L. Lovász. *Combinatorial Problems and Exercises.* North-Holland, 1993.

18. T. Lücking, M. Mavronicolas, B. Monien, and M. Rode. A new model for selfish routing. In *Proc. of the 21st Int. Symp. on Theoretical Aspects of Computer Science,* LNCS 2996, pp. 547–558, 2004.
19. T. Lücking, M. Mavronicolas, B. Monien, M. Rode, P. Spirakis, and I. Vrto. Which is the worst-case nash equilibrium? In *Proc. of the 28th Int. Symp. on Mathematical Foundations of Computer Science,* LNCS 2747, pp. 551–561, 2003.
20. M. Mavronicolas and P. Spirakis. The price of selfish routing. In *Proc. of the 33rd Ann. ACM Symp. on Theory of Computing,* pp. 510–519, 2001.
21. I. Milchtaich. Congestion games with player-specific payoff functions. *Games and economic behavior,* 13:111–124, 1996.
22. J. Nash. Non-cooperative games. *Annals of Mathematics,* 54(2):286–295, 1951.
23. R.W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *Int. Journal of Game Theory,* 2:65–67, 1973.
24. T. Roughgarden. Stackelberg scheduling strategies. In *Proc. of the 33rd Ann. ACM Symp. on Theory of Computing,* pp. 104–113, 2001.
25. T. Roughgarden. The price of anarchy is independent of the network topology. In *Proc. of the 34th Ann. ACM Symp. on the Theory of Computing,* pp. 428–437, 2002.
26. T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of the ACM,* 49(2):236–259, 2002.
27. J.G. Wardrop. Some theoretical aspects of road traffic research. In *Proc. of the Institute of Civil Engineers, Pt. II, Vol. 1,* pp. 325–378, 1956.
28. H.S. Wilf. *Generatingfunctionology.* Academic Press, 1994.

Improved Results for Data Migration and Open Shop Scheduling

Rajiv Gandhi[1], Magnús M. Halldórsson[2], and Guy Kortsarz[1], and Hadas Shachnai[3]*

[1] Department of Computer Science, Rutgers University, Camden, NJ 08102.
{rajivg,guyk}@camden.rutgers.edu
[2] Department of Computer Science, University of Iceland, IS-107 Reykjavik, Iceland.
mmh@hi.is
[3] Department of Computer Science, The Technion, Haifa 32000, Israel.
hadas@cs.technion.ac.il

Abstract. The data *migration* problem is to compute an efficient plan for moving data stored on devices in a network from one configuration to another. We consider this problem with the objective of minimizing the sum of completion times of all storage devices. Kim [13] gave a 9-approximation algorithm for the problem. We improve Kim's result by giving a 5.06-approximation algorithm.
We also address the *open shop scheduling* problem, $O|r_j|\sum w_j C_j$, and show that it is a special case of the data migration problem. Queyranne and Sviridenko [18] gave a 5.83-approximation algorithm for the non-preemptive version of the open shop problem. They state as an obvious open question whether there exists an algorithm for open shop scheduling that gives a performance guarantee better than 5.83. Our 5.06 algorithm for data migration proves the existence of such an algorithm. Crucial to our improved result is a property of the linear programming relaxation for the problem. Similar linear programs have been used for various other scheduling problems. Our technique may be useful in obtaining improved results for these problems as well.

1 Introduction

The *data migration* problem arises in large storage systems, such as *Storage Area Networks* [12], where a dedicated network of disks is used to store multimedia data. As the data access pattern changes over time, the load across the disks needs to be rebalanced so as to continue providing efficient service. This is done by computing a new data layout and then "migrating" data to convert the initial data layout to the target data layout. While migration is being performed, the storage system is running suboptimally, therefore it is important to compute a data migration schedule that converts the initial layout to the target layout quickly.

* Part of this work was done while the author was on leave at Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 658–669, 2004.

This problem can be modeled as a *transfer graph* [13], in which the vertices represent the storage disks and an edge between two vertices u and v corresponds to a data object that must be transferred from u to v, or vice-versa. Each edge has a processing time (or length) that represents the transfer time of a data object between the disks corresponding to the end points of the edge. An important constraint is that any disk can be involved in at most one transfer at any time.

Several variations of the data migration problem have been studied. These variations arise either due to different objective functions or due to additional constraints. One common objective function is to minimize the *makespan* of the migration schedule, i.e., the time by which all migrations complete. Coffman *et al.* [5] introduced this problem. They showed that when edges may have arbitrary lengths, a class of greedy algorithms yields a 2-approximation to the minimum makespan. In the special case where the edges have equal (unit) lengths, the problem reduces to edge coloring of the transfer (multi)graph of the system. The best approximation algorithm known for minimum edge coloring [15] then yields an algorithm for data migration with unit edge length, whose makespan is $1.1\chi' + 0.8$, where χ' is the chromatic index of the graph.

Hall *et al.* [8] studied the data migration problem with unit edge lengths and *capacity constraints;* that is, the migration schedule must respect the storage constraints of the disks. The paper gives a simple 3/2-approximation algorithm for the problem. The papers [8,1] also present approximation algorithms for the makespan minimization problem with the following constraints: *(i)* data can only be moved, i.e, no new copies of a data object can be created, *(ii)* additional nodes can assist in data transfers, and *(iii)* each disk has a unit of spare storage. Khuller *et al.* [12] solved a more general problem, where each data object can also be copied. They gave a constant factor approximation algorithm for the problem.

Another objective function is to minimize the average completion time over all data migrations. This corresponds to minimizing the average edge completion time in the transfer graph. For the case of unit edges lengths, Bar-Noy *et al.* [2] showed that the problem is NP-hard and gave a simple 2-approximation algorithm. For arbitrary edge lengths, Halldórsson *et al.* [10] gave a 12-approximation algorithm for the problem. This was improved to 10 by Kim [13].

In this paper, we study the data migration problem with the objective of minimizing the average completion time over all storage disks. Indeed, this objective favors the individual storage devices, which are often geographically distributed over a large network. It is therefore natural to try and minimize the average amount of time that each of these (independent) devices is involved in the migration process. For the case where vertices have arbitrary weights, and the edges have unit length, Kim [13] proved that the problem is NP-hard and showed that Graham's list scheduling algorithm [6], when guided by an optimal solution to a linear programming relaxation, gives an approximation ratio of 3. She also gave a 9-approximation algorithm for the case where edges have arbitrary lengths. We show that the analysis of the 3-approximation algorithm is tight, and for the case where edges have release times and arbitrary lengths, we give a 5.06-approximation algorithm.

A problem related to the data migration problem is non-preemptive *open shop scheduling,* denoted by $O|r_j|\sum w_j C_j$ in the standard three-field notation [14]. In this problem, we have a set of jobs, $\mathcal{J}$, and a set of machines $M_1, \ldots, M_m$. Each job $J_j \in \mathcal{J}$ consists of a set of m operations: $o_{j,i}$ has the processing time $p_{j,i}$ and must be processed on M_i, $1 \leq i \leq m$. Each machine can process a single operation at any time, and two operations that belong to the same job cannot be processed simultaneously. Also, each job J_j has a positive weight, w_j, and a release time, r_j, which means that no operation of J_j can start before r_j. The objective is to minimize the sum of weighted completion times of all jobs. This problem is MAX-SNP hard [11]. Chakrabarti *et al.* [4] gave a $(5.78+\epsilon)$-approximation algorithm for the case where the number of machines, m, is some fixed constant. They also gave a $(2.89+\epsilon)$-approximation algorithm for the preemptive version of the problem and fixed number of machines. For arbitrary number of machines, Queyranne and Sviridenko [18] presented algorithms that yield approximation factors of 5.83 and 3 for the non-preemptive and preemptive versions of the problems, respectively. The approximation factor for the preemptive version was subsequently improved to $(2+\epsilon)$ by the same authors [17].

Our Contribution. We show that the open shop scheduling problem is a special case of the data migration problem. Hence, all of our positive results for data migration apply to open shop scheduling. Note that the MAX-SNP hardness of the data migration problem follows from the MAX-SNP hardness of open shop scheduling [11]. Our main result is a 5.06-approximation algorithm for the data migration problem with arbitrary edge lengths. Our algorithm is based on rounding a solution of a linear programming (LP) relaxation of the problem. The general idea of our algorithm is inspired by the work of Halldórsson *et al.* [10] in that the edges have to wait before they are actually processed (i.e., data transfer begins). Even though the high-level idea is similar, there are subtle differences that are crucial to the improved results that we present here. Our method combines solutions obtained by using two different wait functions. It is interesting to note that each solution (when all edges are released at time 0) is a 5.83-approximate solution, which is the approximation ratio obtained by Queyranne and Sviridenko [18]. To obtain an approximation ratio better than 5.83, we crucially use a property of the LP relaxation that we prove in Lemma 1. Although the LP relaxation has been used earlier [20,16,19,9,13,18], we are not aware of any previous work that uses such a property of the LP. Our technique may be useful for deriving improved results for other shop scheduling problems.

For the case where edges have unit lengths, we show, by giving a tight example, that the list scheduling analysis of Kim [13] is tight. This illustrates the limitations of the LP relaxation. Finally, we study the open shop problem under *operations completion time* criteria (cf. [18]); that is, we sum the completion times of all operations for every job. For the special case of unit length operations with arbitrary non-negative weights, we show that an algorithm of [10] yields a 1.796 approximation algorithm for the problem.

2 Relation of Data Migration and Open Shop Scheduling

In this section, we formally state the data migration and open shop scheduling problems and show that the latter is a special case of the former.

Data Migration Problem: We are given a graph $G = (V, E)$. Let $E(u)$ denote the set of edges incident on a vertex u. The vertices and edges in G are jobs to be completed. Each vertex v has weight w_v and processing time 0. Each edge e has a length, or processing time, p_e. Moreover, each edge e can be processed only after its release time r_e. All release times and processing times are non-negative integers. The completion time of an edge is simply the time at which its processing is completed. Each vertex v can complete only after all edges in $E(v)$ are completed. Since each vertex v has the processing time 0, the completion time, C_v, of v is the latest completion time of any edge in $E(v)$. The crucial constraint is that two edges incident on the same vertex cannot be processed at the same time. The objective is to minimize $\sum_{v \in V} w_v C_v$.

Open Shop Scheduling Problem: We are given a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, to be scheduled on a set of machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_j has a non-negative weight w_j; also, J_j consists of a set of m operations $o_{j,1}, \ldots, o_{j,m}$, with the corresponding processing times $p_{j,i}$, $1 \leq i \leq m$; the operation $o_{j,i}$ must be processed on the machine M_i. Each machine can process at most one operation at any time, and no two operations belonging to the same job can be processed simultaneously. The completion time C_j of each job J_j is the latest completion time of any of its operations. The objective is to minimize $\sum_{J_j \in \mathcal{J}} w_j C_j$.

The open shop scheduling problem is a special case of the data migration problem, as shown by the following reduction. Given an instance of the open shop scheduling problem, construct a bipartite graph $B = (J, M, F)$ as follows. Each vertex $j_\ell \in J$ represents a job $J_\ell \in \mathcal{J}$, and each vertex $m_i \in M$ represents a machine $M_i \in \mathcal{M}$. The edge $(j_\ell, m_i) \in F$ with processing time $p_{\ell,i}$ corresponds to the operation $o_{\ell,i}$, $1 \leq i \leq m$. Assign $w_{m_i} = 0$ to each vertex $m_i \in M$, and $w_{j_\ell} = w_\ell$ (i.e., the weight of the job J_ℓ) to each vertex $j_\ell \in J$. It is now easy to verify that any data migration schedule for B is a valid solution for the corresponding open shop problem.

In the remainder of the paper, we consider only the data migration problem, with the understanding that all of our results apply to open shop scheduling.

3 A Linear Programming Relaxation

The linear programming relaxation for the data migration problem (without release times) was given by Kim [13]. Such relaxations have been proposed earlier by Wolsey [20] and Queyranne [16] for single machine scheduling problems and by Schulz [19] and Hall *et al.* [9] for parallel machines and flow shop problems. For the sake of completeness, we state below the LP relaxation for the data migration problem.

For an edge e (vertex v) let C_e (C_v) be the variable that represents the completion time of e (resp., v) in the LP relaxation. For any set of edges $S \subseteq E$, let $p(S) = \sum_{e \in S} p_e$ and $p(S^2) = \sum_{e \in S} p_e^2$.

$$(LP) \qquad \text{minimize} \quad \sum_{v \in V} w_v C_v \tag{1}$$

$$\text{subject to:} \qquad C_v \geq r_e + p_e, \qquad \forall v \in V, e \in E(v) \tag{2}$$

$$C_v \geq p(E(v)), \qquad \forall v \in V \tag{3}$$

$$C_v \geq C_e, \qquad \forall v \in V, e \in E(v) \tag{4}$$

$$\sum_{e \in S(v)} p_e C_e \geq \frac{p(S(v))^2 + p(S(v)^2)}{2}, \qquad \forall v \in V, S(v) \subseteq E(v) \tag{5}$$

$$C_e \geq 0, \qquad \forall e \in E \tag{6}$$

$$C_v \geq 0, \qquad \forall v \in V \tag{7}$$

The set of constraints represented by (2), (3), and (4) are due to the different lower bounds on the completion times of a vertex. The justification for constraints (5) is as follows. By the problem definition, no two edges incident on the same vertex can be scheduled at the same time. Consider any ordering of the edges in $S(v)$. If an edge $e \in S(v)$ is the j-th edge to be scheduled among the edges in $S(v)$ then, setting $C_j = C_e$ and $p_j = p_e$, we get

$$\sum_{j=1}^{|S(v)|} p_j C_j \geq \sum_{j=1}^{|S(v)|} p_j \sum_{k=1}^{j} p_k = \sum_{j=1}^{|S(v)|} \sum_{k=1}^{j} p_j p_k = \frac{p(S(v))^2 + p(S(v)^2)}{2}.$$

Although there are exponentially many constraints, the above LP can be solved in polynomial time via the ellipsoid algorithm [16].

3.1 A Property of the LP

In this section, we state and prove a property of the LP that plays a crucial role in the analysis of our algorithm. Let $X(v, t_1, t_2) \subseteq E(v)$ denote the set of edges that complete in the time interval $(t_1, t_2]$ in the LP solution. Hall *et al.* [9] showed that $p(X(v, 0, t)) \leq 2t$. In Lemma 1 we prove a stronger property of a solution given by the above LP. Intuitively, our property states that if too many edges complete early, then other edges must complete late. For example, as a consequence of our property, for any $t > 0$ if $p(X(v, 0, t/2)) = t$ then $p(X(v, t/2, t)) = 0$, which means that no edges in $E(v) \setminus X(v, 0, t/2)$ complete before t in the LP solution. We now formally state and prove the lemma.

Lemma 1. *Consider a vertex v and times $t_1 > 0$ and $t_2 \geq t_1$. If $p(X(v, 0, t_1)) = \lambda_1$ and $p(X(v, t_1, t_2)) = \lambda_2$, then λ_1 and λ_2 are related by*

$$\lambda_2 \leq t_2 - \lambda_1 + \sqrt{t_2^2 - 2\lambda_1(t_2 - t_1)}$$

Proof. Using the constraint (5) of the LP relaxation for vertex v, we get

$$\frac{p(X(v,0,t_2))^2}{2} \leq \sum_{e \in X(v,0,t_1)} p_e C_e + \sum_{e \in X(v,t_1,t_2)} p_e C_e$$

$$\leq p(X(v,0,t_1))t_1 + p(X(v,t_1,t_2))t_2$$

$$\therefore (\lambda_1 + \lambda_2)^2 \leq 2\lambda_1 t_1 + 2\lambda_2 t_2$$

$$\therefore \lambda_2 \leq t_2 - \lambda_1 + \sqrt{t_2^2 - 2\lambda_1(t_2 - t_1)}$$

□

The following result of [9] follows from Lemma 1 by substituting $t_1 = 0$, $\lambda_1 = 0$, and $t_2 = t$.

Corollary 1. *For any vertex v and time $t \geq 0$, $p(X(v,0,t)) \leq 2t$.*

4 Algorithm

Note that if an edge has processing time 0, it can be processed as soon as it is released, without consuming any time steps. Hence, without loss of generality, we assume that the processing time of each edge is a positive integer.

The algorithm is parameterized by a *wait function* $W : E \to \mathcal{R}^+$. The idea is that each edge e must *wait* for W_e ($W_e \geq r_e$) time steps before it can actually start processing. The algorithm processes the edges in non-decreasing order of their completion times in the LP solution. When e is being processed, we say that e is *active.* Once it becomes active, it remains active for p_e time steps, after which it is *finished.* A not-yet-active edge can be *waiting* only if none of its neighboring edges are active; otherwise, it is said to be *delayed.* Thus, at any time, an edge is in one of four modes: *delayed, waiting, active,* or *finished.* When adding new active edges, among those that have done their waiting duty, the algorithm uses the LP completion time as priority.

The precise rules are given in the pseudocode in Fig. 1. Let $wait(e,t)$ denote the number of time steps that e has waited until the end of time step t. Let $Active(t)$ be the set of active edges during time step t. Let $\widetilde{C_e}$ be the completion time of edge e in our algorithm. The algorithm in Fig. 1, implemented as is, would run in pseudo-polynomial time, however, it is easy to implement the algorithm in strongly polynomial time.

One property of our processing rules, that distinguishes it from the wait functions used in [10] for the sum of edge completion times, is that multiple edges can wait at the same time.

We run the algorithm for two different wait functions W and choose the better of the two solutions. For any vertex v (edge e), let C_v^* (C_e^*) be its completion time in the LP solution. In the first wait function, for each edge e we choose $W_e = \lfloor \beta_1 C_e^* \rfloor$, $\beta_1 \geq 1$ and in the second one, we choose $W_e = \lfloor \beta_2 \max\{r_e, p(S_e(u)), p(S_e(v))\} \rfloor$, where $S_e(u) = \{f | f \in E(u), C_f^* \leq C_e^*\}$ and $\beta_2 \geq 1$. Note that the choice of wait functions ensures that the edges become

```
SCHEDULE(G = (V, E), W)
1   Solve the LP relaxation for the given instance.
2   t ← 0
3   Finished ← Active(t) ← ∅
4   for each e ∈ E do
5       wait(e, t) ← 0
6   while (Finished ≠ E) do
7       t ← t + 1
8       Active(t) ← {e | e ∈ Active(t − 1) and e ∉ Active(t − p_e)}
9       for each edge e ∈ Active(t − 1) \ Active(t) do
10          C̃_e ← t − 1
11          Finished ← Finished ∪ {e}              // e is finished
12      for each edge e = (u, v) ∈ E \ (Active(t) ∪ Finished)
13          in non-decreasing order of LP completion time do
14          if (Active(t) ∩ (E(u) ∪ E(v)) = ∅) and (wait(e, t − 1) = W_e) then
15              Active(t) ← Active(t) ∪ {e}        // e is active
16      for each edge e = (u, v) ∈ E \ (Active(t) ∪ Finished) do
17          if (Active(t) ∩ (E(u) ∪ E(v)) = ∅) then
18              wait(e, t) ← wait(e, t − 1) + 1    // e is waiting
19          else wait(e, t) ← wait(e, t − 1)       // e is delayed
20  return C̃
```

Fig. 1. Algorithm for Data Migration

active only after they are released. When all release times are 0, we can choose β_1 and β_2 such that $\beta_1 > 0$ and $\beta_2 > 0$.

5 Analysis

Consider a vertex x and an edge $e = (x, y)$, and recall that C^*_x and C^*_f are their completion times in the LP solution. Let $B_e(x) = \{f | f \in E(x), C^*_f > C^*_e, \widetilde{C_f} < \widetilde{C_e}\}$, i.e., edges in $E(x)$ that finish after e in the LP solution, but finish before e in our algorithm. Recall that $S_e(x) = \{f | f \in E(x), C^*_f \leq C^*_e\}$. Note that $e \in S_e(x)$. Let $\overline{S_e(x)} = S_e(x) \setminus \{e\}$. By constraint (3), we have $p(S_e(x)) + p(B_e(x)) \leq C^*_x$.

We analyze our algorithm separately for the two wait functions defined in Section 4. In each case, we analyze the completion time of an arbitrary but fixed vertex $u \in V$. Without loss of generality, let $e_u = (u, v)$ be the edge that finishes last among the edges in $E(u)$. By constraint (4), we have $C^*_{e_u} \leq C^*_u$. We analyze our algorithm for the case where all edges in $\overline{S_{e_u}(u)} \cup \overline{S_{e_u}(v)}$ finish before e_u in our algorithm. If this is not true then our results can only improve. Let $p(S_{e_u}(v)) = \lambda_{e_u} C^*_{e_u} \leq \lambda_{e_u} C^*_u, 0 \leq \lambda_{e_u} \leq 2$. The upper bound on λ_{e_u} follows from Corollary 1.

Lemma 2. $\widetilde{C_u} \leq W_{e_u} + C^*_u + p(S_{e_u}(v)) + p(B_{e_u}(v))$

Proof. Observe that when e_u is in delayed mode it must be that some edge in $\overline{S_{e_u}(u)} \cup B_{e_u}(u) \cup \overline{S_{e_u}(v)} \cup B_{e_u}(v)$ must be active. Hence, we have

$$\widetilde{C_{e_u}} \leq W_{e_u} + p(S_{e_u}(u)) + p(B_{e_u}(u)) + p(S_{e_u}(v)) + p(B_{e_u}(v))$$

$$\therefore \widetilde{C_u} \leq W_{e_u} + C^*_u + p(S_{e_u}(v)) + p(B_{e_u}(v))$$

□

Define $f(\beta_1, \lambda) = \beta_1 + 1 + \frac{\beta_1+1}{\beta_1} + \sqrt{\left(\frac{\beta_1+1}{\beta_1}\right)^2 - \frac{2\lambda}{\beta_1}}$.

Lemma 3. *If* $W_{e_u} = \lfloor \beta_1 C^*_{e_u} \rfloor$ *then* $\widetilde{C_u} \leq f(\beta_1, \lambda_{e_u}) C^*_u$

Proof. Let $e_b \in B_{e_u}(v)$ be the edge with the largest completion time in the LP solution among all the edges in $B_{e_u}(v)$. Note that when e_b is in waiting mode it must be that either e_u is waiting or an edge in $\overline{S_{e_u}(u)} \bigcup B_{e_u}(u)$ is active. Thus, we get $W_{e_b} \leq W_{e_u} + p(\overline{S_{e_u}(u)}) + p(B_{e_u}(u))$. Hence, we have that $\lfloor \beta_1 C^*_{e_b} \rfloor \leq \lfloor \beta_1 C^*_{e_u} \rfloor + p(S_{e_u}(u)) - p_{e_u} + p(B_{e_u}(u))$. Since $p_{e_u} \geq 1$, it follows that $\beta_1 C^*_{e_b} - 1 \leq \beta_1 C^*_u + C^*_u - 1$, and $C^*_{e_b} \leq \frac{\beta_1+1}{\beta_1} C^*_u$.

Substituting $t_1 = C^*_u, t_2 = \frac{\beta_1+1}{\beta_1} C^*_u$, $\lambda_1 = \lambda_{e_u} C^*_u$, and $\lambda_2 = p(B_{e_u}(v))$ in Lemma 1 and using the fact that $p(S_{e_u}(v)) \leq \lambda_{e_u} C^*_u$, we get

$$p(B_{e_u}(v)) \leq \left(\frac{\beta_1 + 1}{\beta_1} - \lambda_{e_u} + \sqrt{\left(\frac{\beta_1 + 1}{\beta_1}\right)^2 - 2\lambda_{e_u}\left(\frac{\beta_1 + 1}{\beta_1} - 1\right)} \right) C^*_u$$

$$\therefore p(S_{e_u}(v)) + p(B_{e_u}(v)) \leq \left(\frac{\beta_1 + 1}{\beta_1} + \sqrt{\left(\frac{\beta_1 + 1}{\beta_1}\right)^2 - \frac{2\lambda_{e_u}}{\beta_1}} \right) C^*_u$$

The lemma now follows from Lemma 2 and the fact that $C^*_{e_u} \leq C^*_u$. □

Define $h(\beta_2, \lambda) = (\beta_2 + 1)\max\{1, \lambda\} + \frac{\beta_2+1}{\beta_2}$.

Lemma 4. *If* $W_{e_u} = \lfloor \beta_2 \max\{r_{e_u}, p(S_{e_u}(u)), p(S_{e_u}(v))\} \rfloor$ *then* $\widetilde{C_u} \leq h(\beta_2, \lambda_{e_u}) C^*_u$

Proof. By constraints (2) and (3), $r_{e_u} \leq C^*_{e_u} \leq C^*_u$ and $p(S_{e_u}(u)) \leq C^*_u$. Also, recall that $p(S_{e_u}(v)) = \lambda_{e_u} C^*_u, 0 \leq \lambda_{e_u} \leq 2$. Hence, $W_{e_u} \leq \lfloor \beta_2 \max\{C^*_u, \lambda_{e_u} C^*_u\} \rfloor$. We will upper bound $p(S_{e_u}(v)) + p(B_{e_u}(v))$ as follows.

Let $z \in \overline{S_{e_u}(v)} \cup B_{e_u}(v)$ be the edge with the *largest waiting time,* i.e., $W_z = \max_{f \in \overline{S_{e_u}(v)} \cup B_{e_u}(v)} \{W_f\}$. When z is in waiting mode it must be that either e_u is waiting or an edge in $\overline{S_{e_u}(u)} \bigcup B_{e_u}(u)$ is active. Thus we get

$$\begin{aligned} W_z &\leq W_{e_u} + p(\overline{S_{e_u}(u)}) + p(B_{e_u}(u)) \\ &\leq \lfloor \beta_2 \max\{C^*_u, \lambda_{e_u} C^*_u\} \rfloor + p(S_{e_u}(u)) - p_{e_u} + p(B_{e_u}(u)) \\ &\leq \beta_2 \max\{C^*_u, \lambda_{e_u} C^*_u\} + C^*_u - 1 \end{aligned} \tag{8}$$

Let l be the edge with the *largest completion time* in the LP solution among the edges in $\overline{S_{e_u}(v)} \cup B_{e_u}(v)$, i.e., $C_l^* = \max_{f \in \overline{S_{e_u}(v)} \cup B_{e_u}(v)} \{C_f^*\}$. Since $W_l \leq W_z$, we have

$$\lfloor \beta_2 (p(S_{e_u}(v)) + p(B_{e_u}(v))) \rfloor \leq W_l \leq W_z \tag{9}$$

Combining (8) and (9) we get

$$\lfloor \beta_2 (p(S_{e_u}(v)) + p(B_{e_u}(v))) \rfloor \leq \beta_2 \max\{C_u^*, \lambda_{e_u} C_u^*\} + C_u^* - 1$$

$$\therefore p(S_{e_u}(v)) + p(B_{e_u}(v)) \leq \left(\max\{1, \lambda_{e_u}\} + \frac{1}{\beta_2} \right) C_u^*$$

The lemma follow by combining Lemmas 2 with the fact that $C_{e_u}^* \leq C_u^*$. □

Combining the two solutions: For convenience in notation, for each vertex u, let $\lambda_u = \lambda_{e_u}$, and omit β_1 and β_2 as parameters to the functions f and h.

Partition the vertex set into $V_0 = \{u \in V | \lambda_u \leq 1\}$ and $V_1 = \{u \in V | \lambda_u > 1\}$. For a set X of vertices, define $f(X) = \sum_{u \in X} f(\lambda_u) C_u^*$, and similarly $h(X)$. The cost of our solution will be the smaller of the two bounds given by Lemmas 3 and 4, or the smaller of $f(V) = f(V_0) + f(V_1)$ and $h(V) = h(V_0) + h(V_1)$. Observe that performance ratio is a function only of the λ-values of the vertices, with weights C_u^*; thus, we ignore other features of the instance.

The following lemma, whose proof is omitted, shows that performance analysis of our algorithm can be done by optimizing a three-variable function.

Lemma 5. *There is a worst-case instance where $h(V) = f(V)$, and all λ_u are either 0 or $\hat{\lambda}$, for some $\hat{\lambda} \geq 1$ that is a function of only β_1 and β_2.*

Let w be such that $wOPT = \sum_{v \in V_0} w_v C_v^*$. By Lemma 5, in the worst-case scenario, we have

$$w \cdot f(0) + (1 - w) \cdot f(\hat{\lambda}) = w \cdot h(0) + (1 - w) \cdot h(\hat{\lambda})$$

Solving for w, and defining $g(\lambda) = g(\beta_1, \beta_2, \lambda) = h(\beta_2, \lambda) - f(\beta_1, \lambda)$, we have that

$$w = \frac{g(\hat{\lambda})}{g(\hat{\lambda}) - g(0)}.$$

We then obtain an expression for the performance ratio of

$$\rho = w \cdot h(0) + (1 - w) \cdot h(\hat{\lambda}) = \max_{\hat{\lambda} \in (1,2]} \left(h(\hat{\lambda}) + g(\hat{\lambda}) \frac{h(0) - h(\hat{\lambda})}{g(\hat{\lambda}) - g(0)} \right). \tag{10}$$

We can optimize the best choice of parameters β_1 and β_2. When release times of jobs are non-zero, we must restrict the β-values to be at least 1, to ensure that a job does not begin executing before its release time. Setting $\beta_1 = 1.177$ and $\beta_2 = 1.0$, the worst-case is achieved at about $\hat{\lambda} = 1.838$ giving a ratio of $\rho \leq 5.0553$.

Theorem 1. *There exists a* 5.06*-approximation algorithm for the data migration problem, as well as for the open shop scheduling problem.*

When all release times are zero, we can widen the search to all non-zero values. We then obtain a slightly improved ratio of 5.03, when choosing $\beta_1 = 1.125$ and $\beta_2 = 0.8$.

6 Unit Processing Times

When all edges are released at time 0 and have unit processing times, Kim [13] showed that Graham's list scheduling algorithm [6] guided by an optimal solution to the LP relaxation (see in Section 3) gives a 3-approximate solution. The algorithm is called *Ordered List Scheduling (OLS)* [13]. The problem of obtaining a better than 3-approximate solution remained open. In Section 6.1, we show by giving a tight example that OLS cannot achieve a ratio better than 3. The tight example also illustrates the limitations of the LP solution. For the sake of completeness, we state the OLS algorithm and its analysis here. The edges are sorted in non-decreasing order of their completion times in the LP solution. At any time, an edge $e = (u, v)$ is scheduled iff no edge in $S_e(u) \cup S_e(v)$ is scheduled at that time. (Recall that $S_e(u) = \{f | f \in E(u), C^*_f \leq C^*_e\}$.) For any vertex u, if e_u is the edge that finishes last among the edges in $E(u)$, and if $\widetilde{C_u}$ is the completion time of u in OLS, then $\widetilde{C_u} \leq p(S_{e_u}(u)) + p(S_{e_u}(v))$. Combining the fact that $p(S_{e_u}(u)) = C^*_u$ along with $p(S_{e_u}(v)) \leq 2C^*_{e_u} \leq 2C^*_u$ (Corollary 1), we get $\widetilde{C_u} \leq 3C^*_u$ and hence a 3-approximation ratio.

6.1 A Tight Example

Consider a tree rooted at vertex r. Let $S = \{s_1, s_2, \ldots, s_k\}$ be the children of r. For each vertex s_i, let $L_i = \{l^i_1, l^i_2, \ldots, l^i_h\}$ be the children of s_i. Let $L = \cup_{i=1}^{k} L_i$. Let $k = (n+1)/2$ and $h = n-1$. For each vertex $u \in S$, let $w_u = \epsilon$ and for each vertex $v \in L$, let $w_v = 0$. Let $w_r = M$. For each edge e, let $C^*_e = (n+1)/2$ be its completion time in the LP solution. For each vertex $v \in L \cup \{r\}$, $C^*_v = (n+1)/2$ and for each vertex $v \in S$, $C^*_v = n$. The completion times of vertices in L do not matter as the weights of all those vertices are zero. It is easy to verify that this is an optimal LP solution. The cost of the LP solution equals

$$w_r \left(\frac{n+1}{2}\right) + \sum_{i=1}^{k} w_{s_i} n = M\left(\frac{n+1}{2}\right) + k\epsilon n = M\left(\frac{n+1}{2}\right) + \left(\frac{n(n+1)}{2}\right)\epsilon\,.$$

OLS could process the edges in the following order. At any time t, $1 \leq t \leq n-1$, OLS processes all edges in $\{(s_1, l^1_t), (s_2, l^2_t), \ldots, (s_k, l^k_t)\}$. At time $t = n+z$, $0 \leq z < (n+1)/2$, OLS processes edge (r, s_{z+1}). The cost of the solution in OLS is at least

$$w_r \left(n - 1 + \frac{n+1}{2}\right) + \sum_{i=1}^{k} w_{s_i} n = M\left(\frac{3n-1}{2}\right) + \left(\frac{n(n+1)}{2}\right)\epsilon\,.$$

For large n, if $M \gg \epsilon$, the ratio of the cost of the OLS solution to the cost of the LP solution approaches 3.

6.2 Open Shop and Sum of Operation Completion Times

Consider now the open shop problem, where each operation has unit processing time and a non-negative weight, and the objective is to minimize the weighted sum of completion times of all operations. We relate this problem to a result of [10] for the *sum coloring* problem. The input to sum coloring is a graph G, where each vertex corresponds to a unit length job. We need to assign a positive integer (color) to each vertex (job) so as to minimize the sum of the colors over all vertices. The constraint is that adjacent vertices receive distinct colors. In the weighted case, each vertex (job) is associated with a non-negative weight, and the goal is to minimize the weighted sum of the vertex colors.

In the *maximum k-colorable subgraph* problem, we are given an undirected graph G and a positive integer k; we need to find a maximum size subset $U \subseteq V$ such that $G[U]$, the graph induced by U, is k-colorable. In the weighted version, each vertex has a non-negative weight and we seek a maximum weight k-colorable subgraph. The following theorem is proved in [10].

Theorem 2. *The weighted sum coloring problem admits a* 1.796 *ratio approximation algorithm on graphs for which the maximum weight k-colorable subgraph problem is polynomially solvable.*

We can relate this theorem to the above variant of the open shop problem, by defining the bipartite graph $B = (J,M,F)$ (see in Section 2) and setting $G = L(B)$, i.e., G is the line graph of B. Recall that in $L(B)$ the vertices are the edges of B; two vertices are neighbors if the corresponding edges in B share a vertex.

In order to apply Theorem 2, we need to show that the maximum weight k-colorable subgraph problem is polynomial on $L(B)$. Note that this is the problem of finding a maximum weight collection of edges in B that is k-colorable (i.e., can be decomposed into k disjoint matchings in B). Observe that, on bipartite graphs, this problem is equivalent to the well-known weighted *b-matching* problem. In weighted b-matching, we seek a maximum weight set of edges that induces a subgraph of maximum degree at most k. Note that a bipartite graph always admits a matching touching every vertex of maximum degree (c.f. [7]). It follows, that the chromatic index of a bipartite graph is equal to its maximum degree. Since weighted b-matching is solvable in polynomial time (c.f. [3]), the same holds for the weighted k-colorable subgraph problem on $L(B)$. Hence, we have shown

Theorem 3. *Open shop scheduling of unit jobs, under weighted sum of operation completion time criteria, admits a* 1.796 *ratio approximation.*

Acknowledgments. The first author would like to thank Yoo-Ah Kim for introducing to the author the problem of data migration, and Samir Khuller, Yoo-Ah Kim, Aravind Srinivasan, Chaitanya Swamy for useful discussions.

References

1. E. Anderson, J. Hall, J. Hartline, M. Hobbes, *et al. An Experimental Study of Data Migration Algorithms.* In WAE, 145-158, 2001.
2. A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, T. Tamir. *On Chromatic Sums and Distributed Resource Allocation.* Inf. Comput. 140:183-202, 1998.
3. W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. Combinatorial Optimization. Wiley, 1998.
4. S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, J. Wein. *Improved Scheduling Problems For Minsum Criteria.* 23rd ICALP, LNCS 1099, 646-657, 1996.
5. E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. LaPaugh. *Scheduling File Transfers.* SIAM Journal on Computing, 14(3):744-780, 1985.
6. R. Graham. *Bounds for certain multiprocessing anomalies.* Bell System Technical Journal, 45:1563–1581, 1966.
7. H. Gabow and O. Kariv. *Algorithms for edge coloring bipartite graphs and multigraphs.* SIAM Journal of Computing, 11(1), February 1992.
8. J. Hall, J. Hartline, A. Karlin, J. Saia, and J. Wilkes. *On Algorithms for Efficient Data Migration.* In 12th SODA, 620–629, 2001.
9. L. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. *Scheduling to Minimize Average Completion Time: Off-line and On-line Approximation Algorithms.* Mathematics of Operations Research, 22:513-544, 1997.
10. M. M. Halldórsson, G. Kortsarz, and H. Shachnai. *Sum Coloring Interval Graphs and k-Claw Free Graphs with Applications for Scheduling Dependent Jobs.* Algorithmica, 37:187-209, 2003.
11. H. Hoogeveen, P. Schuurman, and G. Woeginger. *Non-approximability Results For Scheduling Problems with Minsum Criteria.* In 6th IPCO, LNCS 1412, 353–366, 1998.
12. S. Khuller, Y. Kim, and Y. C. Wan. *Algorithms for Data Migration with Cloning.* In 22nd PODS, 27–36, 2003.
13. Y. Kim. *Data Migration to Minimize the Average Completion Time.* In 14th SODA, 97–98, 2003.
14. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy-Kan, and D. B. Shmoys. Sequencing and Scheduling: Algorithms and Complexity. In S. C. Graves et al, eds., *Handbooks in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory,* 445–522, 1993.
15. T. Nishizeki and K. Kashiwagi. *On the 1.1 edge-coloring of multigraphs.* SIAM Journal on Discrete Mathematics, 3(3):391-410, 1990.
16. M. Queyranne. *Structure of a Simple Scheduling Polyhedron.* Mathematical Programming, 58:263-285, 1993.
17. M. Queyranne and M. Sviridenko. *A $(2+\epsilon)$-Approximation Algorithm for Generalized Preemptive Open Shop Problem with Minsum Objective.* Journal of Algorithms, 45:202-212, 2002.
18. M. Queyranne and M. Sviridenko. *Approximation Algorithms for Shop Scheduling Problems with Minsum Objective.* Journal of Scheduling, 5:287-305, 2002.
19. A. S. Schulz. *Scheduling to Minimize Total Weighted Completion Time: Performance Guarantees of LP-based Heuristics and Lower Bounds.* In 5th IPCO, LNCS 1084, 301–315, 1996.
20. L. Wolsey. *Mixed Integer Programming Formulations for Production Planning and Scheduling Problems.* Invited talk at the 12th International Symposium on Mathematical Programming, MIT, Cambridge, 1985.

Deterministic M2M Multicast in Radio Networks
(Extended Abstract)

Leszek Gąsieniec[1]*, Evangelos Kranakis[2]**, Andrzej Pelc[3]***, and Qin Xin[1]

[1] Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK,
{leszek,qinxin}@csc.liv.ac.uk
[2] School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada,
kranakis@scs.carleton.ca
[3] Dép. d'informatique, Université du Québec en Outaouais, Hull, Québec, J8X 3X7, Canada,
andrzej.pelc@uqo.ca

Abstract. We study the problem of exchanging messages within a fixed group of k nodes, in an n-node multi-hop radio network, also known as the problem of Multipoint-to-Multipoint (M2M) multicasting. While the radio network topology is known to all nodes, we assume that the participating nodes are not aware of each other's positions. We give a new fully distributed deterministic algorithm for the M2M multicasting problem, and analyze its complexity. We show that if the maximum distance between any two out of k participants is d then this local information exchange problem can be solved in time $O(d \log^2 n + k \log^3 n)$. Hence our algorithm is linear in the size of the subnetwork induced by the participating nodes and only polylogarithmic in the size of the entire radio network.

1 Introduction

Next generation wireless networks are expected to support group communication applications (such as distance learning, video conferencing, disaster recovery and distributed collaborative computing). In such applications, any of the nodes of a well-defined group may be required to send messages to all other nodes in the group. The problem of exchanging messages within a fixed group of nodes in a multi-hop network is called *M2M (multipoint-to-multipoint) multicasting.*

Broadcasting and *gossiping* are two classical problems of information dissemination in computer networks. In the broadcasting problem, we want to distribute a message from a distinguished *source* node to all other nodes in the network. In the gossiping problem, each node v in the network initially holds a message m_v, and we wish to distribute all messages m_v to all nodes in the network. In both problems, one of the main efficiency criteria is the time needed to complete the given communication task. M2M multicasting is a natural generalization of gossiping, in which information exchange concerns not all nodes of the network but only a subset of all nodes, called *participants.*

* Research supported in part by the Royal Academy of Engineering.
** Research supported in part by MITACS, and NSERC grants.
*** Research supported in part by NSERC grant and the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 670–682, 2004.

A radio network is a collection of stations, equipped with capabilities of transmitting and receiving messages. Stations will be referred to as *nodes* of the network. The network is modeled as an n-node undirected connected graph $G = (V, E)$ on the set of these nodes. Each node has a unique label drawn from set $[N] = \{0,1,.., N-1\}$ of integers, where N is bounded by some polynomial in n. An edge e between two nodes means that the transmitter of one end of e can reach the other end. Nodes send messages in synchronous *steps* (time slots). In every step every node acts either as a *transmitter* or as a *receiver*. A node acting as a transmitter sends a message which can potentially reach all of its neighbors. A node acting as a receiver in a given step gets a message, if and only if, exactly one of its neighbors transmits in this step. If at least two neighbors v and v' of u transmit simultaneously in a given step, none of the messages is received by u in this step. In this case we say that a *collision* occurred at u. It is assumed that the effect at node u of more than one of its neighbors transmitting is the same as that of no neighbor transmitting, i.e., a node cannot distinguish a collision from silence.

In this paper we consider deterministic communication algorithms that use the entire knowledge about the network topology. Such algorithms are useful in radio networks that have a reasonably stable graph of connections. As long as no changes occur in the network topology during the execution of the algorithm, the communication task will be completed successfully. Another interesting aspect of deterministic communication in known radio networks is its close relation with randomized communication in ad-hoc radio networks.

Although either broadcasting or gossiping could be used to solve M2M multicasting, the former often does not scale well while the latter may not be efficient because an application may involve only a small fraction of the total number of nodes of the underlying radio network. In this paper we address the problem of minimizing the communication time of M2M multicast in multi-hop radio networks. To the best of our knowledge, this is the first study of M2M multicast time in this communication model.

1.1 Previous Work

Most of the work devoted to radio networks is focused on the broadcasting problem. In the model with known radio network topology, Gaber and Mansour [13] showed that the broadcasting task can be completed in time $O(D+\log^5 n)$, where D is the diameter of the network. Two alternative broadcasting algorithms (superior for small diameters) can be found in [5,20]. The computation of an optimal radio broadcast schedule for an arbitrary network is known to be NP-hard, even if the underlying graph of connections is embedded into a plane [4,22].

Many authors [3,6,7,9,10,12,18,11] studied deterministic distributed broadcasting in ad-hoc radio networks, in which every node knows only its own label, using the model of directed graphs. Increasingly faster broadcasting algorithms working on arbitrary n-node (directed) radio networks were constructed, the currently fastest being the $O(n \log^2 D)$-time algorithm from [11]. (Here D is the radius of the network, i.e, the longest distance from the source to any other node). On the other hand, in [10] a lower bound $\Omega(n \log D)$ on broadcasting time was proved for directed n-node networks of radius D.

The gossiping problem was not studied in the context of radio networks of known topology, until very recent work of Gąsieniec and Potapov [15]. They study the gos-

siping problem in known radio networks, where each node transmission is limited to unit messages. In this model several optimal and almost optimal $O(n)$-time gossiping algorithms are proposed in various standard network topologies, including lines, rings, stars and trees. It is also proved that there exists a radio network topology in which gossiping (with unit messages) requires $\Omega(n \log n)$ time. Very recently, Gąsieniec *et al.* [16] studied gossiping in known radio networks with arbitrarily large messages, and several optimal gossiping algorithms were proposed for a wide range of radio topologies.

So far, the gossiping problem was mostly studied in the context of ad-hoc radio networks, where the topology of connections is unknown to nodes. In this model, Chrobak *et al.* [9] proposed a fully distributed deterministic algorithm that completes the gossiping task in time $O(n^{3/2} \log^3 n)$. For small values of the diameter D, the gossiping time was later improved by Gąsieniec and Lingas [14] to $O(nD^{1/2} \log^3 n)$. Another interesting $O(n^{3/2})$-time algorithm, a tuned version of the gossiping algorithm from [9], can be found in [24]. A very recent $O(n^{4/3} \log n)$–time gossiping algorithm has been proposed by Gąsieniec *et al.* in [17]. A study of deterministic gossiping in ad-hoc radio networks, with messages of limited size, can be found in [8]. The gossiping problem in *ad-hoc* radio networks also attracted studies based on efficient randomized algorithms. In [9], Chrobak et al. proposed an $O(n \log^4 n)$-time gossiping procedure. This time was later reduced to $O(n \log^3 n)$ [21], and very recently to $O(n \log^2 n)$ [11].

1.2 Our Results

The aim of this paper is the design of efficient algorithms for the M2M multicasting problem in radio networks. We study the complexity of this problem for k participating nodes in an n-node radio network. While the topology of the network is known to all nodes, participating nodes are not aware of each other's positions. We show that if the maximum distance between any two out of k participants is d then this information exchange problem can be solved in time $O(d \log^2 n + k \log^3 n)$ by a fully distributed deterministic algorithm. Hence our algorithm is linear in the size of the subnetwork induced by the participating nodes, and only polylogarithmic in the size of the entire radio network. Our solution is based on a novel application of the graph clustering method preserving locality [13] and on efficient adaptive collision resolution based on the concept of promoters, see section 2.1.

2 Paradigms and Tools

All multicast algorithms presented in this paper are based on the following idea. The nodes participating in the multicast process communicate with other participants via messages. Each participating node has initially one message which is the label of the node. The aim is that all participants learn labels of all other participants.

In the first part of the algorithm, the messages are gathered in one selected *meeting point*. The messages traveling towards the meeting point, from time to time compete with other messages for the same communication channel. We will guarantee the invariant that each message competes with any other message at most once. Moreover, the time spent during any particular competition with l other messages is bounded by $O(l \log^2 n)$.

Note that, although each traversing message is kept in a single copy, it leaves its trace in each visited node. In the second part of the multicast procedure, a compound message containing all individual messages is distributed to all participating nodes to inform them about the labels of the others.

Although the algorithms used for trees and for arbitrary graphs share the same general structure, they differ dramatically in details of their design. The two main differences lie in the choice of the meeting point and in the way in which the competition for the same communication channel is resolved.

In trees, the selection of the meeting point is implicit. Before the communication process is started, one node is chosen as the root of the tree. During the multicast process, all messages generated by the participating nodes traverse towards this root. The meeting point corresponds to the first node which is visited by all messages. In fact, the meeting point is the lowest common ancestor (LCA) of all participating nodes, with respect to the chosen root of the tree. Note that the distance between the LCA and all participating nodes is always limited to d. Each competition is resolved with the help of a system of synchronized descending selectors.

In arbitrary graphs, the choice (computation) of the meeting point is much more complex. Not knowing the position of participating nodes, we cannot fix the meeting point in advance, since – in the worst case – messages would have to travel along the diameter of the entire network before meeting each other. Instead, we propose a new clustering concept, that allows us to group all participating nodes in one of the clusters with a relatively small diameter, comparable with d. Each cluster has its own meeting point and a BFS spanning tree rooted in it. In each cluster, similarly as in the case of trees, we try to move all messages from the participating nodes towards the meeting point. However, efficient traversal limited to branches of the BFS tree is not always possible. This is due to the fact that in the cluster there exist edges outside of the BFS tree that potentially cause a lot of conflicts. Thus the competition is becoming much harder. In order to overcome this problem, we propose a special algorithm that resolves conflicts between competing messages. This algorithm is based on a novel use of descending selectors, combined with broadcasting and gossiping procedures.

2.1 Resolving Competition

The main difficulty occurring in radio communication is the presence of collisions. It has been shown before, see, e.g., [10,9], that the most efficient tools designed for collision resolution are based on combinatorial structures possessing a *selectivity property.* We say that a set R hits a set Z on element z, if $R \cap Z = \{z\}$, and a family of sets $\mathcal{F}$ hits a set Z on element z, if $R \cap Z = \{z\}$ for at least one $R \in \mathcal{F}$. In [10] we can find a definition of a family of subsets of set $\{0, 1, \ldots, N-1\} \equiv [N]$ which hits each subset of $[N]$ of size at most $k \leq N$ on all of its elements. They refer to this family as *strongly* k*−selective* family. They also prove the existence of such a family of size $O(k^2 \log N) = O(k^2 \log n)$. In [9] we find a definition of a family of subsets of set $\{0, 1, \ldots, N-1\} \equiv [N]$ which hits each subset of $[N]$ of size at most k on at least $k/2$ distinct elements, where $N \geq k \geq 1$. They call it a k*−selector* and prove the existence of such a family of size $O(k \log N) = O(k \log n)$.

In what follows we show how to cope with collisions occurring during the competition process with a help of selective families and selectors.

Promoting messages in unknown stars. Assume k nodes from $V' = \{v_1, v_2, \ldots, v_k\}$ are immediate neighbors (not aware of each other) of another node w, i.e., they form a star with a center in w, and they all compete (at some stage of the algorithm) to move their message to w. The process of moving messages from nodes in V' to w is called *a promotion.* It is known, that the mechanism based on the selector idea allows a fraction (e.g., a half) of the nodes in V' to deliver their messages to w in time $O(k \log n)$ [9]. Let $S(k)$ represent the collision resolution mechanism based on selectors. Note that $S(k)$, if applied in undirected networks, can be supported by the *acknowledgment of delivery* mechanism in which each transmission from the neighbors of w is alternated with an acknowledgement message coming from the central node w. If during the execution of $S(k)$ a transmission towards w is successful, i.e., one of $v_i \in V'$ succeeds in delivering its message, the acknowledgement issued by w and returned to all nodes in V' contains the label of the successful node; otherwise the acknowledgement is null. Let $\mathbf{S}(k)$ be the mechanism with the acknowledgement feature based on $S(k)$. In other words, the use of $\mathbf{S}(k)$ allows to exclude from further transmissions all nodes in V' that have managed to deliver their message to w during the execution of $\mathbf{S}(k)$. Note that the duration of $\mathbf{S}(k)$ is $O(k \log n)$, see [9].

Let $S^*(i)$ be the communication mechanism based on concatenation (superposition) of i selectors $S(2^i), S(2^{i-1}), \ldots, S(2^1)$. We will call it later as a *descending selector.* The descending selector extended by the acknowledgement mechanism, i.e., the concatenation of $\mathbf{S}(2^i), \mathbf{S}(2^{i-1}), \ldots, \mathbf{S}(2^1)$, forms *a promoter* and it is denoted by $\mathbf{S}^*(i)$. Note that the duration of $\mathbf{S}^*(i)$ is $O(2^i \log n)$.

Lemma 1. *If $V' = \{v_1, v_2, \ldots, v_k\}$ is a set of neighbors of w, and all nodes in V' use the same promoter $\mathbf{S}^*(i)$, where $k \leq 2^i$, then all nodes in V' deliver their messages to w in time $O(2^i \log n)$.*

Proof. The proof is done by induction, and is based on the fact that after the execution of each $\mathbf{S}(2^j)$, for $j = i, \ldots, 1$, the number of competing nodes in V' is $\leq 2^{j-1}$.

Promoting messages in unknown bipartite graphs. Assume that we have a connected bipartite graph B in which nodes are partitioned into two sets U and L. In our further considerations, sets U and L will correspond to two adjacent BFS levels, upper and lower respectively, in a subgraph of G. While, in general, nodes in U and L are not aware of the presence of each other, we assume here that each node $x \in L$ is associated with exactly one of its neighbors (called later a *parent*) $y \in U$ and this relation is known to both of them. Note that a node in U can be a parent of several nodes in L, thus $|U| \leq |L| = l$. We assume also, that initially only nodes in L are aware of their presence in B, i.e., their parents must be informed about it by the children. In what follows we show how to move all messages available at nodes of L, to a single node in U in time $O(l \log^2 n)$. We first assume that the size l is known in advance. As in the case of stars, we call the process of moving messages from L to U a promotion. The promoting algorithm works in 5 stages.

procedure ENHANCED-PROMOTION(L);

1. All nodes in L contact their parents;
   (level U is formed).
2. All nodes belonging to B take part in leader election choosing a node r among all nodes in U;
   (node r is going to collect all messages initially stored in L).
3. Node r initiates broadcasting to all other nodes in B;
   (the broadcast tree (with unidirectional edges) rooted in r is created).
4. Each node (except the root r) contacts its parent in the broadcasting tree;
   (bidirectional edges are now available in the broadcast tree).
5. The root r sends a token visiting all nodes of the broadcasting tree to collect all messages from L and place them in r;
   (all messages are gathered in r).
6. The root r sends a token visiting all nodes of the broadcasting tree, in order to confirm successful delivery of every competing message.

Step 1 is based on a single use of the promoter $\mathbf{S}^*(i)$, for $i - 1 < \log l \leq i$. Even if promoters are designed primarily for promoting nodes in stars, they also prove to be useful in the case of bipartite graphs (with established parent/child relation). As before, we say that a node $x \in L$ contacts its parent y successfully, when all other nodes in L remain silent. This means that the acknowledgement which is later sent by y, will not collide with other messages. The time of step 1 is $O(l \log n)$.
Step 2 is based on the leader election algorithm from [9] combined with the very recent fast deterministic broadcasting algorithm in [19]. The election algorithm works in time $O(l \log^2 n)$.
Step 3 is based on the broadcasting algorithm presented in [6] and works in time $O(l \log n)$
Step 4 is analogous to Step 1. This gives the time complexity $O(l \log n)$.
Steps 5 and 6 are implemented as a simple tree (e.g., pre-order) traversal in time $O(l)$, for details see [6].

Thus the total time of the algorithm is bounded by $O(l \log^2 n)$.

2.2 Graph Clustering Preserving Locality

The main purpose of the clustering method is to obtain a representation of a large graph as a collection of its much smaller subgraphs (clusters), while preserving local distances between the nodes.

Let $G = (V, E)$ be a graph representing a radio network. Initially we pick an arbitrary node c in V that becomes a *central node* in G. The radius of G is the maximum distance D between c and any other node. The clustering method groups nodes belonging to some connected subgraphs G', in the same cluster C. If the diameter of G' is d, the diameter of C is at most $O(d \log n)$.

Definition 1. *Let l_j be the j^{th} BFS level in a graph G with respect to a central node c, i.e., $l_j = \{v | dist(c, v) = j\}$.*

Definition 2. *A partition $\pi(x)$ of the graph G is a division ofG into super-levels, such that, each super-level is composed of $4d$ consecutive BFS levels, where the first super-level starts from an arbitrary but fixed BFS level l_x (note that levels $l_0, l_1, \ldots, l_{x-1}$ are excluded from the partition $\pi(x)$). More formally, the ith super-level in $\pi(x)$ is $G_i(x) = \{v | v \in l_j, (i-1-x)\cdot 4d \leq j \leq (i-x)\cdot 4d - 1\}$, for $i = 1, 2, \cdots, \lceil \frac{D-x}{4d} \rceil$, where D is the radius of G with respect to the central node c. Given a super-level $G_i(x)$, its top level is $l_{(i-1-x)\cdot 4d}$, and its bottom level is $l_{(i-x)\cdot 4d-1}$. Note that $G_i(x)$ is not necessarily connected.*

Definition 3. *For each node u belonging to the top level of $G_i(x)$, we define the pre-cluster $S_u^{(i)}$, which contains all nodes in $G_i(x)$ at distance $\leq 4d$ from u.*

Definition 4. *The clusters are obtained by growing appropriate pre-clusters, according to the mechanism used in the Cover Algorithm presented in [13]. In short, the growing algorithm is performed in $O(\log n)$ stages. In each stage $i = 1, \ldots, \log k$ a collection of clusters C_*^i (each at distance 2 apart) is created as follows. We start with an arbitrary (yet available) pre-cluster which forms a core of a new cluster C_0^i. At each step of the extension procedure we add to the cluster C_0^i a new layer of pre-clusters that intersect with C_0^i or are at distance at most 1 from C_0^i. Note that this extension is successful only if the number of new nodes coming with the new pre-clusters is at least as big as the number of nodes in the pre-clusters already present in the cluster C_0^i. If this condition is not met, the extension of the cluster C_0^i is terminated, i.e., the construction of C_0^i completes without augmenting nodes available in the just considered layer of pre-clusters. Instead, the pre-clusters in the new layer are moved for consideration in stage $i+1$. The process of growing clusters $C_1^i, C_2^i, \ldots$ is performed similarly, and it continues as long as we have at least one pre-cluster that neither forms a part of any cluster constructed in stages $1, \ldots, i$, nor has been moved for consideration in stage $i+1$.*

Lemma 2. *The clusters have the following properties:*

1. *Each cluster is a union of some pre-clusters,*
2. *Each pre-cluster is a member of exactly one cluster.*
3. *Each cluster is a connected sub-graph of G.*
4. *The diameter of each cluster is $O(d \log n)$, and*
5. *There is a $O(\log n)$-**colouring** of the clusters, such that, clusters having the same color are at distance ≥ 2 apart.*

Proof. Properties 1,2, and 3 follow directly from the construction of the clusters. Property 4 is based on the fact that each pre-cluster has diameter $\leq 4d$ and that during construction of any cluster the number of new layers of pre-clusters is limited to $\log n$, since each extension by a new layer of pre-clusters at least doubles the number of nodes in the pre-clusters of currently constructed cluster. Property 5 follows from the fact that during each round we construct clusters at distance 2 apart. Note also that the number of rounds is bounded by $\log n$. This is because in each round at least half of the nodes available in pre-clusters is used to build the clusters of the same color. This is a consequence of arguments used in the proof of Property 4.

Definition 5. *The 2-partition of the graph G comprises two different partitions:* $\pi(0)$ *which starts at the super-level* $G_1(0)$*, and* $\pi(2d)$ *which starts at the super-level* $G_1(2d)$*.*

Lemma 3. *In at least one of the partitions of the 2-partition, there exists at least one cluster that contains all* k *participating nodes and the shortest paths between them. Moreover, in this partition, any other cluster containing some (or all) of the* k *points, is colored differently.*

Proof. Let v be one of the k points. According to our definition of the 2-*partition,* we can prove that the node v must fall into the central $2d$ BFS levels of a super-level in one of the partitions, except for the case when v belongs to the first d BFS levels (when all k points belong to the cluster based on the central node c). Thus, there exists a node p at the top level of the corresponding super-level $G_i(\cdot)$, which is at distance $dist(p, v) \leq 3d$ from the node v. Since all other participating nodes are at distance $\leq d$ from v, there exists a pre-cluster (which constitutes a part of a cluster) $S_p^{(i)}$ which contains the entire set of k participating nodes. The second part of the lemma follows from the fact that clusters having the same color cannot overlap.

3 Efficient M2M Multicast

We start this section with the presentation of a M2M multicasting procedure designed for radio networks with the tree topology. M2M multicast in trees works in time $d + O(k \log^2 n)$-time. We later present a more complex M2M multicast procedure which works in an arbitrary topology in time $O(d \log^2 n + k \log^3 n)$.

3.1 M2M Multicast in Trees

Our M2M multicast algorithm is based on the following principle. The participating nodes make aware other nodes (including all other participants) about their presence by distributing appropriately aimed messages. These are initially gathered in a selected, *central node,* and then distributed to all other participating nodes.

The outline of the multicast algorithm is presented below.

procedure TREE-MULTICAST(T)

1. All nodes agree on the root r of the tree T;
   (the nodes of the tree T are now divided into BFS levels with respect to the distance from the root r*).*
2. Messages issued by the participating nodes traverse, level by level, towards r;
   (traces left by the messages at the intermediate nodes meet eventually, at the latest in r*).*
3. The first node that is visited by all k messages, called the *meeting point,* distributes the compound message back towards all participating nodes;
   (this completes the multicast process).

Step 1., is straightforward. Since all nodes know the topology of G (including the labels of nodes), they use the same deterministic algorithm to choose the root r (e.g., the node with the smallest label). There is no communication involved in this step.
Step 2. is based on synchronized use of promoters and certain properties of rooted trees. Note that during the traversal, a message may meet other messages and compete, e.g., for the access to the same parent in the BFS tree. There may also be collisions caused by simultaneous transmissions at adjacent BFS levels. The latter problem can be solved by enforcing an extra rule that nodes at BFS level j (at distance j from the root r) execute their transmissions in steps i, where $i = j \pmod 3$. This slows down the whole process only by a multiplicative constant 3. The problems caused by the competition of messages require more careful consideration. When the control messages traverse towards the root r of the tree T, each successful transmission must be always confirmed (see the definition of promoters in section 2.1). If the acknowledgement arrives, the transmission is considered to be successful. Otherwise, a special promotion mechanism is switched on, which is designed to deal with the message competition. In what follows we assume that a message uses different (interleaved) time slots for *fast transmissions* (associated with an immediate acknowledgement) and slow transmissions (associated with the competition).

In the promotion mechanism, we use exactly $\log k$ promoters $\mathbf{S}^*(1), \ldots, \mathbf{S}^*(\log k)$ that are run "simultaneously" and periodically. The "simultaneous" execution of promoters of different sizes is done by the time multiplexing, i.e., the execution of two consecutive transmission steps in any $\mathbf{S}^*(i)$ is interleaved with the execution of single steps of every other promoter. Moreover the execution of the promoters of different sizes is synchronized, i.e., a single execution of the promoter $\mathbf{S}^*(i)$ corresponds to two executions of the promoters $\mathbf{S}^*(i-1)$, for any $i = 2, \ldots, \log k$. Any message traversing towards the root r, when it enters the promotion mechanism at some BFS level, it starts using promoter S* (2) as soon as it is available, i.e., when the new execution of S* (2) is scheduled. At the end of the execution of S*(2), if the message is not promoted to the next level, it starts using promoter S*(4) as soon as it is available. This means that it may wait $|\mathbf{S}^*(2)|$ time steps before the new execution of S*(4) takes place. In general, the message can wait for the execution of $\mathbf{S}^*(i)$ at most $|\mathbf{S}^*(i-1)|$ time steps. Note that, when the number of competing messages is bounded by 2^i, all messages are promoted after the execution of $\mathbf{S}^*(2^i)$. Since the running time of all previously used (smaller) promoters and the waiting time is bounded by $(2|\mathbf{S}^*(1)| + \cdots + 2|\mathbf{S}^*(i-1)|) \cdot O(\log n)$ (including time multiplexing), the total time used to promote the competing messages is $O(2^i \log^2 n)$.

Lemma 4. *The last message enters the meeting point (the lowest common ancestor (LCA) of all participating nodes, with respect to r) in time $O(d + k \log^2 n)$.*

Proof. Note that the *lowest common ancestor* (LCA) of all participating nodes (with respect to r) is at distance at most d from each of them. Consider a single message. When it moves towards the root (in fact, towards the meeting point LCA), it traverses each edge in two time units, if there is no competition. The time complexity related to this type of transmissions can be bounded by $O(d)$. If at any time the message competes with some other l messages, it is promoted to the next BFS level in time $O(l \log^2 n)$. Note that two messages competing once will never compete against each other again,

since later on, they travel along the same path towards the root of the tree. This means that the total time spent by a message on competing with other messages is bounded by $O(k \log^2 n)$. Thus the last message arrives at the meeting point in time $O(d + k \log^2 n)$.

Step 3. is a simple broadcasting procedure that distributes the compound message to all nodes (including all participants) within distance d from the meeting point. Since there are no collisions in radio broadcasting in trees, the compound message is distributed to all participating nodes in time at most d.

Theorem 1. *The M2M multicast problem in radio networks with a tree topology can be solved in time* $O(d + k \log^2 n)$.

3.2 M2M Multicast in Arbitrary Graphs

In this section we show how to perform M2M multicast in arbitrary radio networks in time $O(d \log^2 n + k \log^3 n)$. The algorithm is based on the clustering method introduced in section 2.2, on efficient promotion of messages in bipartite graphs, see section 2.1, and some other observations.

In view of the clustering method, there exists at least one (and at most $\log n$) cluster(s) with diameter $\leq d \log n$ that contain(s) all k participating nodes. In what follows, we consider computation performed inside a single cluster. Recall that simultaneous execution of transmissions in clusters having the same color does not cause collisions between the clusters, because all clusters of the same color are at distance at least 2 apart. In order to avoid collisions between clusters in different colors, we execute computation for different colors in $O(\log n)$ (number of colors) different stages. This gives an $O(\log n)$ slowdown in comparison with an execution in a single cluster. Note that having the partition into clusters ready, we could now perform the M2M multicast in time $O(k \cdot d)\text{polylog}n$, applying a leader election algorithm and broadcasting k times. However, our intention is to design a $O((k + d)\text{polylog}n)$ algorithm (thus linear in the size of the subnetwork induced by the participating nodes and only polylogarithmic in the size of the entire radio network). The computation in a cluster C of the 2-partition is performed as follows.

procedure GRAPH-MULTICAST(C)

1. Select a leader in C which becomes the root r of a spanning BFS tree T;
   (after this step the nodes in C are partitioned into BFS levels with respect to the distance from the root r).
2. Messages sent by the participating nodes travel, level by level, towards the root r;
   (note that, in the case of a competition, a message may be muted to the next BFS level via (a sequence) of edges, including those not belonging to the BFS tree T).
3. The root r distributes the compound message to all participating nodes;
   (This completes the multicast process).

Step 1. does not involve communication, since the topology of G is known to every node. Thus the division of G into clusters can be computed locally and independently in each node of G.

Step 2. uses two types of moves. Some moves towards the root are performed along the edges of the BFS tree. However, such simple moves are feasible only in the case when the traversing messages are not involved in any competition. As soon as a traversing message starts to compete (i.e., it does not receive the acknowledgement of the successful transmission), it enters the system of promotion procedures, which is based on the concept of the ENHANCED-PROMOTION procedure, see section 2.1.

The promotion algorithm in arbitrary graphs is more complex than its tree counterpart, due to the presence of external edges (with respect to the BFS tree) that cause more collisions during transmissions. This time, the competition does not always concern a single node that is a joint parent of nodes containing the competing messages. In fact, some nodes containing traversing messages and their parents may form a connected bipartite subgraph B of G (with partitions U and L at adjacent BFS levels). Regardless of the latter difference, we would like to use a similar amortization argument, while assessing the time complexity of the multicast algorithm. Indeed, we show that if at any BFS level, l messages are involved in the competition (within a bipartite graph B), all messages from the set L will be moved to a single node in U in time $O(l \log^2 n)$. Thus if two messages compete once in some bipartite graph, they will never compete against each other again. Similarly as in the case of trees, the promoting algorithm is based on simultaneous (interleaved) and periodic execution of the procedure ENHANCED-PROMOTION(i), for $i = 1, \ldots, \log k$, that deals with sets of competing messages of size $2^1, 2^2, \ldots, 2^{\log k}$, respectively. Recall that in section 2.1 we explained how to promote competing messages in bipartite graphs, when the size of the set of competing messages is known. In what follows we explain how this assumption can be dropped and shed more light on details of the promotion algorithm used at any BFS level.

At any BFS level, when a message m traversing towards the root r enters the promotion mechanism, it waits for the first available execution of the procedure ENHANCED-PROMOTION(1). Similarly as in trees, if the promotion was not successful (the number of competitors was too large), message m waits for the next (complete) execution of the procedure ENHANCED-PROMOTION(2), and so on, for all consecutive powers of two $\leq \log k$. Note that in trees, since all messages compete for the same parent, any message promoted to the next level, will never be obstructed by its former competitors again. We would like to use the same invariant in the case of general graphs too. Thus we insist that all messages competing in a bipartite graph eventually meet in one of the nodes of the set U. Moreover, we will exclude from promotion all messages that managed to gather in one node of U, if not all their competitors in the bipartite graph B managed to do so. This is to guarantee that a pair of messages that competed once will never compete again.

Recall that, upon the completion of procedure **ENHANCED-PROMOTION**(i), the acknowledgement confirming a successful promotion of all competing messages is sent across the connected component of the bipartite graph B. If the acknowledgement does not arrive (e.g., when the graph B is larger then 2^i), all nodes in B know that they have to use the next available execution of the procedure **ENHANCED-PROMOTION**$(i+1)$. However, if the confirmation arrives, the competing messages are still not sure whether all messages in B were properly discovered.

Indeed, there might be several connected components $B_1, B_2, \ldots, B_m$ of B, satisfying $B_1 \cup B_2 \cup \cdots \cup B_m = B$, that are not aware of each other at the end of the execution

of ENHANCED-PROMOTION(i). This happens when, for some reason, all internal transmissions in each B_i are not interrupted by local transmissions in other components. This can be checked in the following way. Every component B_i has its leader l_i whose label will play the role of a label of the whole component B_i. The pattern of transmissions used in each B_i is based on the combination of the concept of strongly 2-selective family [10] and of Steps 5 and 6 in the ENHANCED-PROMOTION procedure. One set R in the strongly 2-selective family, in relation to the label l_i, is replaced by either the whole execution of Steps 5 and 6 in the ENHANCED PROMOTION procedure (**if** $l_i \in R$) or by a continuous sequence of *noisy calls* (**if** $l_i \notin R$), meant to blur communication in the neighboring component. Note that if the component B_i is connected by an edge with some other component B_j, there will be a step in the application of the strongly 2-selective family when the bit associated with B_i is set to 1 and the bit associated with B_j is set to 0 (and vice versa). In this case the traversal of the message in the component B_i will be interrupted, which is enough to figure out that B_i does not form the whole graph of competitors. The cost of Steps 4 & 5 is bounded by $O(2^i)$ and the number of steps in the strongly 2-selective family is $O(\log n)$. Thus the cost of this test (including time multiplexing) is bounded by $O(2^i \log^2 n)$.

In Step 3, the distribution of the compound message is performed with the help of a broadcasting procedure from [20] in time $O(d \log n + \log^2 n)$.

Theorem 2. *The M2M multicast problem in arbitrary radio networks can be solved in time* $O(d \log^2 n + k \log^3 n)$.

4 Conclusion

In this paper we gave an $O(d \log^2 n + k \log^3 n)$-time algorithm for solving the M2M multicast problem for a group of k participating nodes with maximum distance d in an arbitrary radio network consisting of n nodes. Our approach uses a clustering technique for partitioning the radio network and a new algorithm for promoting messages in clusters. Interesting problems left for further investigation include (1) improving the upper bounds of our algorithms, (2) developing locality-sensitive multicast algorithms for the case when the nodes of the network have only limited (e.g., local) knowledge of the topology, and (3) investigating how efficient updating affects performance of multicast in mobile radio systems.

References

1. S. Banerjee, S. Khuller, A Clustering Scheme for Hierarchical Control in Multi-hop Wireless Networks, in Proc. *INFOCOM 2001,* pp 1028-1037.
2. R. Bar-Yehuda, O. Goldreich, and A. Itai, On the time complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization, *Journal of Computer and System Sciences,* 45 (1992), pp 104-126.
3. D. Bruschi and M. Del Pinto, Lower bounds for the broadcast problem in mobile radio networks, *Distributed Computing* 10 (1997), pp 129-135.
4. I. Chlamtac and S. Kutten, *On broadcasting in radio networks-problem analysis and protocol design, IEEE Transactions on Communications* 33 (1985), pp 1240-1246.

5. I. Chlamtac and O. Weinstein, The wave expansion approach to broadcasting in multihop radio networks, *IEEE Trans. on Communications* 39 (1991), pp 426-433.
6. B. Chlebus, **L. Gąsieniec,** A. Gibbons, A. Pelc and W. Rytter, Deterministic broadcasting in unknown radio networks, *Distributed Computing* 15 (2002), pp 27-38.
7. B. Chlebus, **L. Gąsieniec,** A. Ostlin, and M. Robson, Deterministic Radio Broadcasting, in Proc. *27th Int. Colloq. on Automata, Languages and Programming,* ICALP'00, pp 717-728.
8. M. Christersson, **L. Gąsieniec** and A. Lingas, Gossiping with bounded size messages in ad-hoc radio networks, in Proc. *29th International Colloquium on Automata, Languages and Programming,* ICALP'02, pp 377-389.
9. M. Chrobak, **L. Gąsieniec** and W. Rytter, Fast Broadcasting and Gossiping in Radio Networks, *Journal of Algorithms* 43(2), 2002, pp 177-189.
10. A.E.F. Clementi, A. Monti and R. Silvestri, Selective families, superimposed codes, and broadcasting on unknown radio networks, in Proc. *12th Ann. ACM-SIAM Symposium on Discrete Algorithms,* SODA'01, pp 709-718.
11. A. Czumaj and W. Rytter, Broadcasting algorithms in radio networks with unknown topology, in Proc. *44th Ann. Symp. on Foundations of Computer Science,* FOCS'03, pp 492-501.
12. G. DeMarco and A. Pelc, Faster broadcasting in unknown radio networks, *Information Processing Letters* 79, 2001, pp 53-56.
13. I. Gaber and Y. Mansour, *Broadcast in radio networks,* in Proc. *6th Ann. ACM-SIAM Symp. on Discrete Alg.,* SODA'95, pp 577-585. Also, *Journal of Algorithms,* 46(1), 2003, pp 1-20.
14. **L. Gąsieniec** and A. Lingas, On adaptive deterministic gossiping in ad hoc radio networks, *Information Processing Letters* 2(83), 2002, pp 89-94.
15. **L. Gąsieniec** and I. Potapov, *Gossiping with unit messages in known radio networks,* in Proc. *2nd IFIP Int. Conference on Theoretical Computer Science,* TCS'02, pp 193-205.
16. **L. Gąsieniec,** I. Potapov and Q. Xin, Time efficient gossiping in known radio networks, to appear in Proc. *11th Colloq. on Struct. Inform. and Comm. Complexity,* SIROCCO'04.
17. **L. Gąsieniec,** T. Radzik and Q. Xin, Faster deterministic gossiping in *ad-hoc* radio networks, to appear in Proc. *9th Scandinavian Workshop on Algorithm Theory,* SWAT'04.
18. D. Kowalski and A. Pelc, Faster deterministic broadcasting in ad hoc radio networks, in Proc. *20th Ann. Symp. on Theor. Aspects of Comp. Science,* STACS'03, pp 109-120.
19. D. Kowalski and A. Pelc, Broadcasting in undirected ad hoc radio networks, in Proc. *22nd ACM Symposium on Principles of Distributed Computing,* PODC'03, pp 73-82.
20. D. Kowalski and A. Pelc, Centralized deterministic broadcasting in undirected multi-hop radio networks, manuscript 2004.
21. D. Liu and M. Prabhakaran, On Randomized Broadcasting and Gossiping in Radio Networks, in Proc. *8th Annual International Conference on Computing and Combinatorics,* COCOON'02, pp 340-349.
22. A. Sen and M.L. Huson, *A new model for scheduling packet radio networks,* in Proc. *15th Ann., Joint Conference of the IEEE Comp. and Comm. Soc.,* 1996, pp 1116-1124.
23. P.J. Slater, E.J. Cockayne and S.T. Hedetniemi, Information Dissemination in Trees, *SIAM Journal on Computing,* 10, 1981, pp 892-701.
24. Y. Xu, An $O(n^{1.5})$ deterministic gossiping algorithm for radio networks, *Algorithmica,* 36(1), 2003, pp 93-96.

Syntactic Control of Concurrency*

D.R. Ghica, A.S. Murawski, and C.-H.L. Ong

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
{drg.andrzej,lo}@comlab.ox.ac.uk

Abstract. We consider a finitary procedural programming language (finite data-types, no recursion) extended with parallel composition and binary semaphores. Having first shown that may-equivalence of second-order open terms is undecidable we set out to find a framework in which decidability can be regained with minimum loss of expressivity. To that end we define an annotated type system that controls the number of concurrent threads created by terms and give a fully abstract game semantics for the notion of equivalence induced by typable terms and contexts. Finally, we show that the semantics of all typable terms, at any order and in the presence of iteration, admits a regular-language representation and thus the restricted observational equivalence is decidable.

1 Introduction

Game semantics has emerged as a powerful paradigm for giving semantics to a spectrum of programming languages ranging from purely functional languages to those with non-functional features such as control operators and references [1,2, 3,4,5]. Ghica and McCusker [6] found that the game semantics of a second-order fragment of a procedural language can be captured by regular languages, demonstrating a new, semantics-directed, approach to software model-checking [7]. Ghica has subsequently extended the approach to a call-by-value language with arrays [8], Hoare-style assertions [9] and specification [10].

In this paper we propose a game-based framework for compositional model checking of concurrent programs. We have developed a fully-abstract game model for ICA, a concurrent language based on Idealized Algol extended with parallel command composition $(C \,||\, C)$ and binary semaphores (**sem**) manipulated by blocking primitives **grab** and **release** [11]. However, the model seems unsuitable for model-checking applications. We can show that observational equivalence, even at second order in the absence of recursion, is not decidable. The sources of non-finitary behaviour are the free identifiers of first or higher-order types, which correspond to procedures using an argument in an unbounded number of concurrent threads of computation.

In the game model, active threads at any moment correspond to *pending questions* in a play. Hence, we constrain plays by placing bounds on the allowable number of pending questions and enforce these restrictions syntactically

* Work funded by British EPSRC, Canadian NSERC and St. John's College, Oxford.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 683–694, 2004.

using a type system augmented with resource bounds. The key differences between this type system and the standard type system, are the "linearization" of application and parallel composition, i.e. requiring the environments of the two sub-terms to be disjoint. We also revise the *contraction rule* to count the number of contracted occurrences of a variable. We call this type system *Syntactic Control of Concurrency* (SCC); it is a generalization of *Serially Reentrant Algol* (SRIA), a type system introduced by Abramsky to identify higher-order terms of a sequential language denotable by "pointer-free" finitary game models [12].

The bounds imposed on the number of pending questions by SCC can be seen as a kind of *assume-guarantee* reasoning (see e.g. [13]): bounds on the behaviour of the *Opponent* represent *assumptions* on the behaviour of the environment, while bounds on the behaviour of the *Proponent* represent *guarantees* on the behaviour of the system. Typability can be seen as composition, made possible by the fact that the guarantees and the assumptions match. Unsurprisingly, not all terms of the original language admit a resource-bounding typing.

Resource-sensitive type systems are an area of research with numerous applications; the examples mentioned below are only entry points to a vast literature. The nature of the controlled resource is usually duration [14] or space [15]; applications of such systems are as diverse as execution in embedded systems [16], memory management [17], compilation to hardware [18] or proof-carrying code [19]. Type systems have also been used to control more abstract resources, such as variable usage for improved compilation [20] or interference effects for specification and verification [21].

The motivation behind SCC is to isolate (open) terms with finitary models for the purpose of automated verification. The notion of resource in SCC, which we may call *active threads of computation,* has a computational meaning, but it is primarily motivated by the game-semantic analysis of the language [11]. The main thrust of the paper is thus semantic; we plan to investigate the type-theoretic issues of SCC separately.

2 SCC: A Resource-Bounding Type System

Theorem 1. *May-equivalence of second-order ICA terms is undecidable.*

Proof. Using the game semantic model of [11] we can show that observational equivalence of terms in the second-order fragment of ICA can be reduced to the halting problem for Minsky machines, which is known to be undecidable [22]. □

The simulation above is possible because free identifiers $\mathbf{com} \to \mathbf{com}$ correspond to functions that investigate the argument an arbitrary number of times (possibly in parallel). Therefore the key to regaining decidability is to restrict the number of times an argument is used concurrently. However, we need not restrict the number of sequential uses, to allow for iteration and all sorts of interesting procedural programs.

The type system is thus for the recursion-free fragment with **while**-loops. Divergence, $\Omega_{\mathbf{com}}$, can then be defined to be **while 1 do skip.** Types are generated by the following grammar:

$$\beta ::= \mathbf{com} \mid \mathbf{exp} \mid \mathbf{var} \mid \mathbf{sem} \qquad \theta ::= \beta \mid \gamma \to \theta \qquad \gamma ::= \theta^n.$$

The numbers that label the left-hand side of a function type will be called *resource bounds.* An occurrence m of a resource bound in a type θ is an *assume* (resp. *guarantee*) if it occurs in the left-hand scope of an even (resp. odd) number of $\to$'s in θ. Formally, m is an assume (a guarantee) in θ iff $\theta = \mathcal{A}[m]$ ($\theta = \mathcal{G}[m]$):

$$\mathcal{G}[\,] ::= \theta_1^{[\,]} \to \theta_2 \mid \theta^n \to \mathcal{G}[\,] \mid \mathcal{A}[\,]^n \to \theta, \qquad \mathcal{A}[\,] ::= \theta^n \to \mathcal{A}[\,] \mid \mathcal{G}[\,]^n \to \theta.$$

Assumes and guarantees will turn out to correspond to the Opponent/Player polarity in game semantics. For instance, 3 in $(\mathbf{com}^3 \to \mathbf{com})^4 \to \mathbf{com}$ is an assume and 4 is a guarantee.

Assumes concern the behaviour of the program context and guarantees that of the program. The assumes of a typing judgement $\theta_1^{n_1}, \ldots, \theta_k^{n_k} \vdash M : \theta$ are the assumes in θ along with the guarantees in $\theta_1, \ldots, \theta_k$. The guarantees of a typing judgement are the guarantees of θ, the assumes in $\theta_1, \ldots, \theta_k$ and $n_1, \ldots, n_k$.

We use types of this form to approximate the maximum number of concurrent sub-threads of computation at any moment. This estimate is subject to assumes on the environment. Intuitively, if a program has a type θ, then provided the environment behaves according to the assumes, the program's behaviour satisfies the guarantees. In this spirit we introduce a sub-typing relation which can be taken to correspond to weakening the constraints imposed by SCC:

$$\beta \leq \beta \qquad \frac{m \leq n}{\theta^n \leq \theta^m} \qquad \frac{\gamma_2 \leq \gamma_1 \quad \theta_1 \leq \theta_2}{\gamma_1 \to \theta_1 \leq \gamma_2 \to \theta_2}.$$

Intuitively, a subtype gives a less precise approximation: higher on the behaviour of the program and lower for the environment. In the latter case, the bound is considered inferior because it applies to a weaker behaviour of the environment.

The SCC typing rules are given in Fig. 1. Typing judgements are of the form $\Gamma \vdash_r M : \theta$, where $\Gamma = x_1{:}\theta_1^{n_1}, \ldots, x_k{:}\theta_k^{n_k}$; we write $n\Gamma = x_1{:}\theta_1^{n \cdot n_1}, \ldots, x_k{:}\theta_k^{n \cdot n_k}$. Note that the typing rules make a distinction between parallel and sequential composition. Parallel composition and application have "linear" rules, in which the context are required to be disjoint, as opposed to the rules for sequential operators ($\square$ can stand for ; , :=, etc.) including branching and iteration. The contraction rule has been modified so that the assumed bounds on the contracted variable are accumulated into the new variable.

SCC enjoys the standard syntactic properties of a typed lambda calculus (basis, generation, subterm, substitution and subject reduction lemmas) [23].

Remark 1. The rule for application is also "linear," requiring disjoint environments for the function and the application. The reason is that call-by-name application is a straitjacketed form of concurrency in which the computation carried out by the function is interleaved with that of its argument, albeit in a highly constrained fashioned. For instance, if F is a first-order function, any computation arising in an application $F(M)$ also arises in the parallel composition $\cdots F(\cdots) \cdots \,||\, \cdots M \cdots$, where the ellipses stand for code manipulating semaphores so that the right interleaving of effects is enforced [11]. The restriction of application to disjoint environments is also used in SRIA [12].

$$x : \theta^1 \vdash_r x : \theta$$

$$\frac{\Gamma \vdash_r M : \theta}{\Gamma, x : \gamma \vdash_r M : \theta}$$

$$\frac{\{\Gamma \vdash_r M_1 : \theta_1\} \qquad \Gamma \vdash_r M_2 : \theta_2}{\Gamma \vdash_r \{M_1\} \square M_2 : \theta_3}$$

$$\frac{\Gamma \vdash_r M : \theta^n \to \theta' \qquad \Delta \vdash_r N : \theta}{\Gamma, n\Delta \vdash_r MN : \theta'}$$

$$\frac{\Gamma \vdash_r M : \mathbf{exp} \qquad \Gamma \vdash_r M_1, M_2 : \beta}{\Gamma \vdash_r \mathbf{if}\ M\ \mathbf{then}\ M_1\ \mathbf{else}\ M_2 : \beta}$$

$$\frac{\Gamma, x : \mathbf{var}^n \vdash_r M : \mathbf{com}, \mathbf{exp}}{\Gamma \vdash_r \mathbf{newvar}\ X := m\ \mathbf{in}\ M : \mathbf{com}, \mathbf{exp}}$$

$$\frac{\Gamma \vdash_r M : \theta \qquad \theta \leq \theta'}{\Gamma \vdash_r M : \theta'}$$

$$\frac{\Gamma, x : \theta^m, y : \theta^n \vdash_r M : \theta'}{\Gamma, x : \theta^{m+n} \vdash_r M[x/y] : \theta'}$$

$$\frac{\Gamma \vdash_r C_1 : \mathbf{com} \qquad \Delta \vdash_r C_2 : \mathbf{com}}{\Gamma, \Delta \vdash_r C_1 \,||\, C_2 : \mathbf{com}}$$

$$\frac{\Gamma, x : \gamma \vdash_r M : \theta}{\Gamma \vdash_r \lambda x.M : \gamma \to \theta}$$

$$\frac{\Gamma \vdash_r B : \mathbf{exp} \qquad \Gamma \vdash_r C : \mathbf{com}}{\Gamma \vdash_r \mathbf{while}\ B\ \mathbf{do}\ C : \mathbf{com}}$$

$$\frac{\Gamma, S : \mathbf{sem}^n \vdash_r M : \mathbf{com}, \mathbf{exp}}{\Gamma \vdash_r \mathbf{newsem}\ S := m\ \mathbf{in}\ M : \mathbf{com}, \mathbf{exp}}$$

Fig. 1. SCC typing rules

Example 1. For any $n \in \mathbb{N}$ we have

1. $\vdash_r \lambda f x.f(f(x)) : (\mathbf{com}^n \to \mathbf{com})^{n+1} \to (\mathbf{com}^{n^2} \to \mathbf{com})$
2. $\vdash_r \lambda f x.f(x); f(x) : (\mathbf{com}^n \to \mathbf{com})^1 \to (\mathbf{com}^n \to \mathbf{com})$
3. $\vdash_r \lambda f x.f(x) \,||\, f(x) : (\mathbf{com}^n \to \mathbf{com})^2 \to (\mathbf{com}^{2n} \to \mathbf{com})$
4. $\vdash_r \lambda f.f(f\mathbf{skip}) : (\mathbf{com}^n \to \mathbf{com})^{n+1} \to \mathbf{com}$
5. $\vdash_r \lambda g.g(\lambda x.g(\lambda y.x)) : ((\mathbf{com}^n \to \mathbf{com})^n \to \mathbf{com})^{n+1} \to \mathbf{com}$

Not all ICA terms are typable in SCC. For example, the application of the term 5 to term 4 above is untypable, even using subsumption. However, as seen in Lemma 1 below, there is a wide class of typable terms.

Given the bounds for environment, SCC can be used to certify bounds for the program. We say that a term $\Gamma \vdash M : \theta$ is r-typable if for any assignment of assumes to Γ and θ there exists an assignment of guarantees such that when we adorn Γ, θ with these bounds we get Γ', θ' such that $\Gamma' \vdash_r M : \theta'$. We shall write $\eta^{\mathbf{a}}$, $\eta^{\mathbf{g}}$ respectively for the two assignments. Since not all terms are typable, not all terms are r-typable. Therefore our approach will not apply to all higher-order terms. The Lemma below, which is by no means exhaustive, shows that there is a rich supply of terms which are r-typable.

Lemma 1. *Any term with β-redexes of order at most 1 (i.e. the argument is of base type or a first-order function) is r-typable. In particular, β-normal terms of all types are r-typable.*

Using SCC we can define a new *observational approximation* relation $\sqsubseteq\!\!\approx_r$ using typable terms and contexts along with their bounds. Suppose $\Gamma \vdash_r M_1, M_2 : \theta$. In what follows we write $\vdash_r \mathcal{C}[M_i]$ to mean that $\mathcal{C}[M_i]$ is typable using (possibly several copies of) the given type derivation of $\Gamma \vdash_r M_i$, up to appropriate renaming of variables. We define $\Gamma \vdash_r M_1 \sqsubseteq\!\!\approx_r M_2$ to hold iff for all contexts $\mathcal{C}[-]$ such that $\vdash_r \mathcal{C}[M_i] : \mathbf{com}$ we have: $\mathcal{C}[M_1] \Downarrow$ implies $\mathcal{C}[M_2] \Downarrow$. Similarly, we write $\Gamma \vdash_r M_1 \cong_r M_2$ iff $\Gamma \vdash_r M_1 \sqsubseteq\!\!\approx_r M_2$ and $\Gamma \vdash_r M_2 \sqsubseteq\!\!\approx_r M_1$. In particular, the definition applies to the terms for which the above lemma holds. Note that no

bound needs to be placed on the way M_i is used in $\mathcal{C}[M_i]$, the bounds concern only the way its free identifiers are trapped in context. In the definition of $\lesssim_r$ we require $\Gamma \vdash_r M_i : \theta$ to have the same annotations. But if two terms are typable with the same assumed bounds, it is always possible to type them with the same guaranteed bounds by sub-typing.

Example 2 ([24]). Consider the terms

$$M_1 \equiv \mathbf{newvar}\, x := 0 \,\mathbf{in}\, p(x := !x + 1; x := !x + 1);\ \mathbf{if}\ even(!x)\ \mathbf{then}\ \Omega_{\mathbf{com}}$$
$$M_2 \equiv \mathbf{newvar}\, x := 0 \,\mathbf{in}\, p(x := !x + 2);\ \mathbf{if}\ even(!x)\ \mathbf{then}\ \Omega_{\mathbf{com}}$$

with $p : \mathbf{com} \to \mathbf{com}$. Brookes has shown that in *sequential* Algol they are observationally equivalent, whereas in *concurrent* Algol they are not. In SCC we have $p : (\mathbf{com}^1 \to \mathbf{com})^1 \vdash_r M_1 \cong_r M_2$; but for any (assumed) bound $n > 1$, $p : (\mathbf{com}^n \to \mathbf{com})^1 \vdash_r M_1 \not\cong_r M_2$. The reason is that the assumed bound of 1 only allows identifier p to be bound to a procedure which uses its argument sequentially. For example, context $\mathcal{C}[-] = (\lambda p.[-])(\lambda c.c \,||\, c)$ cannot trap $p : \mathbf{com}^1 \to \mathbf{com}$. On the other hand, context $\mathcal{C}[-] = (\lambda p.[-])(\lambda c.c; c)$ can trap $p : \mathbf{com}^n \to \mathbf{com}$ for any n. A formal proof of this example is immediate once the connection with game semantics is established in Sec. 4.

3 The Game Model

In [11] we have given a game model which is fully abstract for $\lesssim$ and $\cong$ (the standard notions of observational approximation and equivalence in ICA). We use the model to interpret the annotations from the type system and to show how the model can be used to reason about $\lesssim_r, \cong_r$.

We give a sketch of the model. An *arena* A is a triple $\langle M_A, \lambda_A, \vdash_A \rangle$ where M_A is a set of *moves*, $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ is a function determining for each $m \in M_A$ whether it is an *Opponent* or a *Proponent move*, and a *question* or an *answer*. We write $\lambda_A^{OP}, \lambda_A^{QA}$ for the composite of λ_A with respectively the first and second projections. $\vdash_A$ is a binary relation on M_A, called *enabling*, satisfying: if $m \vdash_A n$ for no m then $\lambda_A(n) = (O, Q)$, if $m \vdash_A n$ then $\lambda_A^{OP}(m) \neq \lambda_A^{OP}(n)$, and if $m \vdash_A n$ then $\lambda_A^{QA}(m) = Q$. If $m \vdash_A n$ we say that m *enables* n. We shall write I_A for the set of all moves of A which have no enabler; such moves are called *initial*. Note that an initial move must be an Opponent question.

The *product* $(A \times B)$ and *arrow* $(A \Rightarrow B)$ arenas are defined by:

$$M_{A \times B} = M_A + M_B \qquad M_{A \Rightarrow B} = M_A + M_B$$
$$\lambda_{A \times B} = [\lambda_A, \lambda_B] \qquad \lambda_{A \Rightarrow B} = [\langle \lambda_A^{PO}, \lambda_A^{QA} \rangle, \lambda_B]$$
$$\vdash_{A \times B} = \vdash_A + \vdash_B \qquad \vdash_{A \Rightarrow B} = \vdash_A + \vdash_B + \{ (b, a) \mid b \in I_B \text{ and } a \in I_A \}$$

where $\lambda_A^{PO}(m) = O$ iff $\lambda_A^{OP}(m) = P$.

An arena is called *flat* if its questions are all initial (consequently the P-moves can only be answers). In arenas used to interpret base types all questions are initial and P-moves answer them as detailed in the table below.

Arena	O-question	P-answers	Arena	O-question	P-answers
$[\![\mathbf{com}]\!]$	run	ok	$[\![\mathbf{exp}]\!]$	q	$0, \ldots, N$
$[\![\mathbf{var}]\!]$	$read$	n	$[\![\mathbf{sem}]\!]$	$grab$	ok
	$write(n)$	ok		$release$	ok

A *justified sequence* in arena A is a finite sequence of moves of A equipped with pointers. The first move is initial and has no pointer, but each subsequent move n must have a unique pointer to an earlier occurrence of a move m such that $m \vdash_A n$. We say that n is (explicitly) justified by m or, when n is an answer, that n answers m. Note that interleavings of several justified sequences may not be justified sequences; instead we shall call them *shuffled sequences.* If a question does not have an answer in a justified sequence, we say that it is *pending* (or *open*) in that sequence. In what follows we use the letters q and a to refer to question- and answer-moves respectively, m denotes arbitrary moves and m_A a move from M_A. Not all justified sequences are considered "valid" plays:

Definition 1. *The set P_A of* positions *(or* plays*) over A consists of the justified sequences s over A which satisfy the two conditions below.*

FORK : *In any prefix $s' = \cdots\ q \overset{\frown}{\cdots}\ m$ of s, the question q must be pending before m is played.*

WAIT : *In any prefix $s' = \cdots\ q \overset{\frown}{\cdots}\ a$ of s, all questions justified by q must be answered.*

For two shuffled sequences s_1 and s_2, $s_1 \amalg s_2$ denote the set of all interleavings of s_1 and s_2. For two sets of shuffled sequences S_1 and S_2, $S_1 \amalg S_2 = \bigcup_{s_1 \in S_1, s_2 \in S_2} s_1 \amalg s_2$. Given a set X of shuffled sequences, we define $X^0 = X$, $X^{i+1} = X^i \amalg X$. Then $X^{\circledast}$, called *iterated shuffle* of X, is defined to be $\bigcup_{i \in \mathbb{N}} X^i$.

Definition 2. *A* strategy σ *on A (written $\sigma : A$) is a prefix-closed subset of P_A, which is* O-complete *(i.e. if $s \in \sigma$ and $so \in P_A$, where o is an (occurrence of an) O-move, then $so \in \sigma$).*

Strategies $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ are composed in the standard way, by considering all possible interactions of positions from τ with shuffled sequences of $\sigma^{\circledast}$ in the shared arena B and then hiding the B moves.

The model consists of *saturated* strategies only: the saturation condition stipulates that all possible (sequential) observations of (parallel) interactions must be present in a strategy: actions of the environment can always be observed earlier if possible, actions of the program can always be observed later. To formalize this, for any arena A a preorder $\preceq$ on P_A is defined, as the least transitive relation $\preceq$ satisfying $s_0 \cdot o \cdot s_1 \cdot s_2 \preceq s_0 \cdot s_1 \cdot o \cdot s_2$ and $s_0 \cdot s_1 \cdot p \cdot s_2 \preceq s_0 \cdot p \cdot s_1 \cdot s_2$ for all s_0, s_1, s_2 where o is an O-move and p is a P-move. In the above pairs of positions moves on the lhs of $\preceq$ have the same justifier as on the rhs.

Definition 3. *A strategy σ is* saturated *iff $s \in \sigma$ and $s' \preceq s$ imply $s' \in \sigma$.*

Arenas and saturated strategies form a Cartesian closed category $\mathcal{G}_{\mathrm{sat}}$ in which $\mathcal{G}_{\mathrm{sat}}(A, B)$ consists of saturated strategies on $A \Rightarrow B$. The identity strategy is defined by "saturating" the alternating positions $s \in P_{A_1 \Rightarrow A_2}$ such that $\forall\, t \sqsubseteq_{\mathrm{even}} s,\ t \upharpoonright A_1 = t \upharpoonright A_2$, which gives rise to the behaviour of an unbounded buffer. Other elements of the syntax are represented by the least saturated strategies generated by the plays from the table below:

;	$q_1\ run\ ok\ q_0\ a_0\ a_1$	$\|\|$	$run_2\ run_0\ run_1\ ok_0\ ok_1\ ok_2$
:=	$run_2\ q_1\ n_1\ write(n)_0\ ok_0\ ok_2$	!	$q\ read\ n\ n$
grab	$run_1\ grab_0\ ok_0\ ok_1$	**release**	$run_1\ release_0\ ok_0\ ok_1$

newvar $X := n$ $\quad q\,q\,(read\,n)^*\,(\sum_{i=0}^{N}(write(i)\ ok\ (read\,i)^*))^*\,a\,a$

newsem $S := 0$ $\quad q\,q\,(grab\ ok\ release\ ok)^*\,(grab\ ok + \epsilon)\,a\,a$

newsem $S := 1$ $\quad q\,q\,(release\ ok\ grab\ ok)^*\,(release\ ok + \epsilon)\,a\,a.$

As shown in [11], $\mathcal{G}_{\mathrm{sat}}$ is fully abstract for $\cong$ in the sense mentioned below. Let $\mathsf{comp}(\sigma)$ be the set of non-empty complete plays of a strategy σ.

Theorem 2. $\Gamma \vdash M_1 \precsim M_2 \iff comp([\![\Gamma \vdash M_1]\!]) \subseteq comp([\![\Gamma \vdash M_2]\!])$.

4 The Game Model Revisited

In order to analyze the positions induced by terms in more detail we shall define a more restricted games framework where plays can form a subset of P_A as opposed to the full P_A. In particular we are going to dissect the possibilities for the function space game $A \Rightarrow B$. To do that we introduce an auxiliary notion of games in which shuffled sequences are allowed (cf. [25]).

Definition 4. *A* bounded *game $\underline{A}$ is a pair $\langle A, R_A \rangle$ where A is an arena and R_A is a prefix-closed subset of $P_A^{\circledast}$.*

We also refer to the elements of R_A as plays and write $\mathsf{comp}(R_A)$ for the set of complete plays in R_A (those in which all questions are answered). The games of $\mathcal{G}_{\mathrm{sat}}$ can be viewed as bounded games where $R_A = P_A$. Bounded games can be combined using a number of constructions.

$$\underline{A} \times \underline{B} = (A \times B, R_A + R_B) \qquad \underline{A} \otimes \underline{B} = (A \times B, R_A \amalg R_B) \qquad !\underline{A} = (A, R_A^{\circledast})$$
$$\underline{A} \multimap \underline{B} = (A \Rightarrow B, \{\, s \in P_{A \Rightarrow B}^{\circledast} \mid s \upharpoonright A \in R_A,\ s \upharpoonright B \in R_B \,\})$$

and $\underline{A} \Rightarrow \underline{B} = !\underline{A} \multimap \underline{B}$. We also have $!\underline{A} \otimes !\underline{B} = !(\underline{A} \times \underline{B})$. Note that where $R_A = P_A$, $R_B = P_B$ the $\times$ and $\Rightarrow$ constructions coincide with the previous ones.

Let us now define $!_s\underline{A} = (A, (\mathsf{comp}(R_A))^* \cdot R_A)$, i.e. $R_{!_s\underline{A}} \subseteq R_{!\underline{A}}$ and $!_s$ is an impoverished, sequential, version of ! where a new "thread" of R_A can be started only when the previous one is completed. An important case of $!_s\underline{A}$ which we use in the following is when $\underline{A}$ is well-opened, i.e. each play in R_A

can contain only one occurrence of an initial move, namely, the first move of the play (all games interpreting ICA types are of that kind). Then $!_s\underline{A}$ contains plays which might have many occurrences of initial moves, but only one occurrence of an initial question can be open (pending) at any time. Similarly, $\bigotimes_{1\leq i\leq n} !_s\underline{A}$ contains plays with at most n pending questions; we shall write $\underline{A}^n$ for it. We use this construction to specify restricted function spaces: instead of $A \Rightarrow B = !A \multimap B$ we consider $A^n \multimap B$. These restrictions turn out to give the correct interpretation of the bounds inferred by the type system given before.

Regardless of whether we deal with standard ICA type or typing judgements (annotated with bounds or not) $[\![\cdots]\!]$ stands for the usual interpretation in $\mathcal{G}_{\mathrm{sat}}$ (i.e. the information about bounds is completely ignored by $[\![\cdots]\!]$). We introduce the notation $[\![\cdots]\!]_\eta$ for bound-sensitive semantic interpretation.

Let $\Gamma \vdash_r M : \theta$, where $\Gamma = \theta_1^{n_1}, \ldots, \theta_k^{n_k}$. In $\mathcal{G}_{\mathrm{sat}}$ it is standardly interpreted by a strategy for the game $[\![\Gamma \vdash \theta]\!] = [\![\theta_1]\!] \times \ldots \times [\![\theta_k]\!] \Rightarrow [\![\theta]\!]$ or, equivalently, $![\![\theta_1]\!] \otimes \ldots \otimes ![\![\theta_k]\!] \multimap [\![\theta]\!]$. Suppose η represents a vector of resource bounds consistent with $\Gamma \vdash_r M : \theta$. It is not necessary that η includes all the bounds used in the resource-sensitive type judgement. Then the corresponding bounded game, denoted by $[\![\Gamma \vdash \theta]\!]_\eta$, is defined inductively in the same way as $[\![\Gamma \vdash \theta]\!]$ except that whenever a bound n is specified by η (for an occurrence of $\rightarrow$ or θ_i), we use $A^n \multimap B$ and A^n instead of respectively $A \Rightarrow B = !A \multimap B$ and $!A$.

Example 3. Suppose we have $x_1 : (\mathbf{com}^9 \rightarrow \mathbf{sem})^5, x_2 : (\mathbf{exp}^3 \rightarrow \mathbf{com})^7 \vdash M : \mathbf{exp}^7 \rightarrow \mathbf{var}$. The complete vector of resource bounds is (9,5,3,7,7). Let η stand for the distinguished bounds (–, 5,3, –, 7). Then

$$[\![\mathbf{com} \rightarrow \mathbf{sem}, \mathbf{exp} \rightarrow \mathbf{com} \vdash \mathbf{exp} \rightarrow \mathbf{var}]\!]_\eta$$
$$= (![\![\mathbf{com}]\!] \multimap [\![\mathbf{sem}]\!])^5 \otimes !([\![\mathbf{exp}]\!]^3 \multimap [\![\mathbf{com}]\!]) \multimap ([\![\mathbf{exp}]\!]^7 \multimap [\![\mathbf{var}]\!])$$

This notation is flexible enough to handle assumes, guarantees or combined assume-guarantee resource bounds in a uniform way.

Now we are ready to interpret the bounds given by the type system using the game model. Let us define the restriction $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathbf{a}}}$ of the semantics according to the assumed bounds to be $[\![\Gamma \vdash M : \theta]\!] \cap R_{[\![\Gamma \vdash \theta]\!]_{\eta^{\mathbf{a}}}}$, i.e. $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathbf{a}}}$ is $[\![\Gamma \vdash M : \theta]\!]$ in which O-moves are restricted to those allowed by the $A^n \multimap B$ games consistent with the bounds in $\eta^{\mathbf{a}}$. More precisely, for each occurrence m of an initial move from such B Opponent will not be allowed to play an initial move from A justified by m if the current position already contains n pending questions justified by m. The guaranteed bounds given by SCC are then sound in that they are correct approximations of the shape of positions explored by P when O behaves according to $\eta^{\mathbf{a}}$, i.e. the positions are not only in $R_{[\![\Gamma \vdash \theta]\!]_{\eta^{\mathbf{a}}}}$ but also in $R_{[\![\Gamma \vdash \theta]\!]_{\eta^{\mathbf{a}}\eta^{\mathbf{g}}}}$, where by $\eta\eta'$ we mean the two combined constrain vectors.

Theorem 3. $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathbf{a}}} \in R_{[\![\Gamma \vdash \theta]\!]_{\eta^{\mathbf{a}}\eta^{\mathbf{g}}}}$

The theorem can be proved by induction on the derivation of $\Gamma \vdash_r M : \theta$ in conjunction with the Lemma below, which validates the induction (note that the original definition of $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathbf{a}}}$ relies on the full interpretation $[\![\Gamma \vdash M : \theta]\!]$).

Lemma 2. *The definition of* $[\![\Gamma \vdash_r M : \theta]\!]_{\eta^a}$ *is compositional:* $[\![\Gamma \vdash_r M : \theta]\!]_{\eta^a}$ *can be defined directly by induction on the structure of* $\vdash_r$ *derivations. Moreover, the inductive clauses are identical to those for* $[\![\cdots]\!]$.

Crucially, given $\dfrac{\Gamma \vdash_r M : \theta^n \to \theta' \qquad \Delta \vdash_r N : \theta}{\Gamma, n\Delta \vdash_r MN : \theta'}$ where $\eta_1^a, \eta_2^a, \eta_3^a$ represent the assumed bounds of the respective three judgements, in order to calculate $[\![\Gamma, n\Delta \vdash_r MN : \theta']\!]_{\eta_3^a}$ one only needs positions from $[\![\Gamma \vdash_r M : \theta^n \to \theta']\!]_{\eta_1^a}$ and $[\![\Delta \vdash_r N : \theta]\!]_{\eta_2^a}$. The above is an important step in our method, because it allows the inductive definition of restricted denotations: full denotations are much more complicated that the resource-restricted ones. The sets of complete plays induced by the restricted denotations $\mathsf{comp}([\![\Gamma \vdash_r M : \theta]\!]_{\eta^a})$ provide a fully abstract model of $\sqsubseteq\!\!\approx_r$.

Lemma 3. *Suppose* $\Gamma \vdash_r M_1, M_2 : \theta$ *and let* η^a *be the final assignment of assumed bounds. Then* $\mathsf{comp}([\![\Gamma \vdash_r M_1 : \theta]\!]_{\eta^a}) \subseteq \mathsf{comp}([\![\Gamma \vdash_r M_2 : \theta]\!]_{\eta^a})$ *implies* $\Gamma \vdash M_1 \sqsubseteq\!\!\approx_r M_2 : \theta$.

Proof. Suppose $\vdash_r C[M_i] : \mathbf{com}$ (i=1,2) and $C[M_1]\Downarrow$. Then by the soundness of $\mathcal{G}_{\mathrm{sat}}$ [11] $\mathsf{comp}([\![C[M_1]]\!]) \neq \emptyset$. By Lemma 2 $[\![C[M_1]]\!]$ can be defined inductively through $[\![\Gamma \vdash M_1]\!]_{\eta^a}$, so because $\mathsf{comp}([\![\Gamma \vdash_r M_1]\!]_{\eta^a}) \subseteq \mathsf{comp}([\![\Gamma \vdash_r M_2]\!]_{\eta^a})$ we also have $\mathsf{comp}([\![C[M_2]]\!]) \neq \emptyset$. Thus again, by the adequacy of $\mathcal{G}_{\mathrm{sat}}$ $C[M_2]\Downarrow$, so indeed $M_1 \sqsubseteq\!\!\approx_r M_2$. □

To prove the converse we need to strengthen the definability result from [11] to ensure that terms corresponding to positions are also typable. This means that we cannot simply regard justification pointers as indicating parallel threads of computation and have to sequentialize threads where possible. Below we illustrate the difference between the two definability algorithms.

Example 4. Let us consider a position in the game for $\mathbf{com}^2 \to \mathbf{com}$:

$run \cdot run_1 \cdot run_1 \cdot ok_1 \cdot run_1 \cdot ok_1 \cdot run_1 \cdot ok_1 \cdot ok_1 \cdot ok$. The algorithm from [11] would return $\lambda x.\mathbf{newvar}\ x_0, x_3, x_5, x_7, x_8 := 0\ \mathbf{in}\ x_0 := 1; M; WAIT_9$, where $M \equiv (P_1 \,||\, P_2 \,||\, P_4 \,||\, P_6)$, $P_1 \equiv WAIT_1; x; x_3 := 1$, $P_2 \equiv WAIT_2; x; x_5 := 1$, $P_4 \equiv WAIT_4; x; x_8 := 1$, $P_6 \equiv WAIT_6; x; x_7 := 1$, but the term does not have the required type $\mathbf{com}^2 \to \mathbf{com}$. The refined version produces $M \equiv (P_1; P_4) \,||\, (P_2; P_6)$ instead. The term $WAIT_i$ tests whether all variables x_j with indices less than i are set to 1 and diverges if they are not.

Lemma 4. *Suppose* θ *is a type with constraints* η *and* $s \in R_{[\![\theta]\!]_\eta}$. *Then there exists a term* $\vdash_r M : \theta$ *such that* $[\![M]\!]$ *is the least saturated strategy containing* s.

Theorem 4. *Using the same assumes as above.* $\Gamma \vdash M_1 \sqsubseteq\!\!\approx_r M_2 : \theta$ *implies* $\mathsf{comp}([\![\Gamma \vdash_r M_1 : \theta]\!]_{\eta^a}) \subseteq \mathsf{comp}([\![\Gamma \vdash_r M_2 : \theta]\!]_{\eta^a})$.

5 Regular Representation

In this section we show how sets of complete plays $\mathsf{comp}(\llbracket \Gamma \vdash_r M : \theta \rrbracket_{\eta^a})$ can be represented faithfully as regular languages and compared by checking language equivalence. The main difficulty to be addressed is the need to represent pointers.

For any bounded game θ, we represent the positions of $R_{\llbracket\theta\rrbracket_{\eta^a\eta^g}}$ using an alphabet $\mathcal{A}(\theta)$ defined as follows: $\mathcal{A}(\beta) = M_{\llbracket\beta\rrbracket}$, $\mathcal{A}(\gamma \to \theta) = \mathcal{A}(\gamma) + \mathcal{A}(\theta)$ and $\mathcal{A}(\theta^n) = \{ m^i \mid m \in \mathcal{A}(\theta),\ 1 \leq i \leq n \}$. Thus, elements of $\mathcal{A}(\theta)$ can be seen as moves of $\llbracket\theta\rrbracket$ decorated with a vector $\vec{i} = (i_1, \ldots, i_k)$ of labels produced by the last clause. The letters $m^{\vec{i}}$ will be used to encode occurrences of m in positions from $R_{\llbracket\theta\rrbracket_{\eta^a\eta^g}}$ subject to two invariants. If a question q has several open occurrences then each of them will be represented by a different vector. Let an occurrence of a question q be represented by $q^{\vec{i}}$; if an occurrence of another question m is justified by the above occurrence of q, then m is represented as $m^{j\vec{i}}$ for some $j \in \mathbb{N}$.

We explain below how each position from the game under question will be represented so that the invariants are satisfied and only letters from $\mathcal{A}(\theta)$ are used. Note that the initial moves of θ occur without labels in $\mathcal{A}(\theta)$. They will also be represented as such in positions (this never leads to ambiguities since positions have unique initial moves). Given a representation of s a representation of sm is calculated as follows.

- If m is an answer to an occurrence of q represented by $q^{\vec{i}}$ then m is represented as $m^{\vec{i}}$.
- If m is a question justified by an occurrence of q represented as $q^{\vec{i}}$, then there exists a sub-game $G_m^n \multimap G_q$ of $\llbracket\theta\rrbracket_{\eta^a\eta^g}$ such that q, m are initial moves of respectively G_q, G_m. Since sm is a position of $\llbracket\theta\rrbracket_{\eta^a\eta^g}$ there can be at most $n-1$ open questions in s that are justified by the same occurrence of q and, hence, represented as $q^{j\vec{i}}$ in s. Thus one of the labels from $\{1, \ldots, n\}$, say k, has not been used. Then we represent m as $m^{k\vec{i}}$ (any such k will do).

Note that, thanks to the labels, justification pointers can be uniquely reconstructed from the representation so it is faithful. However, it is not unique because of the arbitrary choice of k. We will say that a representation is *canonical* if k is always chosen to be the least k available. The notion of canonicity is crucial to comparing representations of positions as they will provide the link between language equivalence and program equivalence.

Given a set S of strings over $\mathcal{A}(\theta)$ representing a set of plays (e.g. a strategy) on $R_{\llbracket\theta\rrbracket_{\eta^a\eta^g}}$ we write $\mathsf{can}(S)$ for the canonization of that representation.

Lemma 5. *If S is regular so is $\mathsf{can}(S)$.*

Proof. Given an automaton accepting S one construct one for $\mathsf{can}(S)$. The number of open questions in any position of $R_{\llbracket\theta\rrbracket_{\eta^a\eta^g}}$ is uniformly bounded. Hence, with the help of finite memory we can keep track of all labels of open questions during the runtime of the automaton and relabel the accepted letters as required in a canonical representation. Since only finite store is needed, all this can be

done by a finite automaton, so $\mathsf{can}(S)$ is also regular. The formal construction proceeds by annotating the states of the original automaton with all possible configurations of the finite memory. □

Let $\Gamma = \theta_1^{n_1}, \ldots, \theta_k^{n_k}$. We will show that the canonical representation of $\mathsf{comp}(\llbracket \Gamma \vdash_r M : \theta \rrbracket_{\eta^a})$, which we denote simply by $\llbracket \Gamma \vdash_r M : \theta \rrbracket$, is a regular language over $\mathcal{A} = \mathcal{A}(\theta_1^{n_1}) + \ldots + \mathcal{A}(\theta_k^{n_k}) + \mathcal{A}(\theta)$. Many of the definitions, especially for the imperative part of the language have the same flavour as those for Idealized Algol [6] so we focus on the more difficult cases below. Sometimes the operation on regular languages will have to be followed by an explicit conversion to canonical form.

Let $(\!|\Gamma, \Delta \vdash_r C : \mathbf{com}|\!)$ be defined by $\llbracket \Gamma, \Delta \vdash_r C \rrbracket = run \cdot (\!|\Gamma, \Delta \vdash_r C|\!) \cdot ok$. Then we take $(\!|\Gamma, \Delta \vdash_r C_1 \,||\, C_2|\!)$ to be $(\!|\Gamma \vdash_r C_1|\!) \amalg (\!|\Delta \vdash_r C_2|\!)$ (which preserves canonicity). Contraction is defined through renaming of labels associated with y. The labels $1, \ldots, n$ are replaced with $m+1, \ldots, m+n$. This induces a homomorphism on the language so the result is still regular but needs canonization. We define $\mathrm{id}_{\mathbf{com}}$ by $\{ run \cdot run^1 \cdot ok^1 \cdot ok \}$. For other base types the definition is analogous. We extend it to function types $\theta^n \to \theta'$ as follows. Let $\mathrm{id}_{\theta'} = \sum_{q,a}(q \cdot q^1 \cdot \mathrm{id}_{\theta'}^{q,a} \cdot a^1 \cdot a)$. Then $\mathrm{id}_{\theta^n \to \theta'} = \mathsf{can}(\sum_{q,a}(q \cdot q^1 \cdot (\amalg_{i=1}^n (\mathrm{id}_\theta^{i1})^* \amalg \mathrm{id}_{\theta'}^{q,a}) \cdot a^1 \cdot a))$, where id_θ^{j1} is id_θ in which each move $m^{\vec{i}}$ is replaced with $m^{\vec{ij}1}$.

For application it is crucial that canonical representations interact as the interaction has to be represented in the same way both by the function and by the argument. Let $\Delta = \theta_1^{n_1}, \ldots, \theta_k^{n_k}$. For $i = 1, \ldots, n$ let $\tilde{N}_i$ be the same as $\llbracket \Delta \vdash_r N : \theta \rrbracket$ except that the moves from the θ-component are additionally decorated with the label i while the original labels of moves from θ_j $(1 \leq j \leq k)$ (i.e. $1, \ldots, n_j$) are replaced respectively with $(i-1)n_j + 1, \ldots, in_j$. Clearly, these operations preserve regularity. Then we can define $\llbracket \Gamma, \Delta \vdash MN : \theta' \rrbracket$ to be $\mathsf{can}((\tilde{M} \cap \tilde{N}) \setminus \mathcal{A}(\theta^n))$ where $\tilde{N} = \mathcal{A}(\Gamma)^* \amalg \mathsf{can}(\amalg_{i=1}^n (\tilde{N}_i)^*) \amalg \mathcal{A}(\theta')^*$ and $\tilde{M} = \llbracket \Gamma \vdash M : \theta^n \to \theta' \rrbracket \amalg \mathcal{A}(\theta_1^{n \cdot n_1})^* \amalg \ldots \amalg \mathcal{A}(\theta_k^{n \cdot n_k})^*$. Finally, no changes are needed to interpret subsumption.

Theorem 5. *$\llbracket \Gamma \vdash_r M \rrbracket$ is a canonical representation of $\mathsf{comp}(\llbracket \Gamma \vdash_r M : \theta \rrbracket_{\eta^a})$.*

Theorem 6. *$\underset{\sim}{\sqsubset}_r$ and $\cong_r$ are decidable.*

6 Further Work

The previous section establishes that there is a finite-state representation of terms of SCC, and that it can be used, in principle, for model checking using a method similar to [7]. Lemma 1 and the various examples we give suggest that the restrictions imposed by the tighter typing discipline are not onerous. However, to claim a fully automated verification (and certification) procedure the issue of automated type inference must be investigated. Finally, only by incorporating these theoretical results in a model-checking tool (FDR seems a good candidate [26]) can we evaluate the practicality of the method.

References

1. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Information and Computation **163** (2000)
2. Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF: I, II and III. Information and Computation **163** (2000)
3. Abramsky, S., McCusker, G.: Linearity, sharing and state. In: Proceedings of 1996 Workshop on Linear Logic. ENTCS **3**., Elsevier (1996)
4. Laird, J.: Full abstraction for functional languages with control. In: LICS 12. (1997)
5. Abramsky, S., Honda, K., McCusker, G.: A fully abstract game semantics for general references. In: LICS 13. (1998)
6. Ghica, D.R., McCusker, G.: Reasoning about Idealized ALGOL using regular languages. In: ICALP 27. LNCS **1853**. (2000)
7. Abramsky, S., Ghica, D.R., Murawski, A.S., Ong, C.-H.L.: Applying game semantics to compositional software modeling and verification. In: TACAS 10. LNCS **2988** (2004)
8. Ghica, D.R.: Regular language semantics for a call-by-value programming language. In: Proceedings of MFPS 17. ENTCS **45** (2001)
9. Ghica, D.R.: A regular-language model for Hoare-style correctness statements. In: Proceedings of the Verification and Computational Logic (2001)
10. Ghica, D.R.: A Games-based Foundation for Compositional Software Model Checking. PhD thesis, Queen's University School of Computing, Canada (2002)
11. Ghica, D.R., Murawski, A.S.: Angelic semantics of fine-grained concurrency. In: FOSSACS 7. LNCS **2987** (2004)
12. Abramsky, S.: Beyond Full Abstraction: model-checking for Algol-like languages Marktoberdorf International Summer School 2001. (lecture slides)
13. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49** (2002) 672–713
14. Hofmann, M.: Linear types and non-size-increasing polynomial time computation. In: LICS 14 (1999)
15. Hofmann, M.: A type system for bounded space and functional in-place update. Nordic Journal of Computing **7** (2000)
16. Hughes, J., Pareto, L.: Recursion and dynamic data-structures in bounded space: Towards embedded ML programming. ICFP 4. ACM SIGPLAN Notices **34** (1999)
17. Tofte, M.: Region inference for higher-order functional languages. LNCS **983** (1995)
18. Mycroft, A., Sharp, R.: A statically allocated parallel functional language. In: ICALP 27. LNCS **1853** (2000)
19. Necula, G.C.: Proof-carrying code. In: POPL 24 (1997)
20. Wansbrough, K., Jones, S.L.P.: Once upon a polymorphic type. In: POPL 26 (1999)
21. Reynolds, J.C.: Syntactic control of interference. In: POPL 5 (1978)
22. Minsky, M.: Computation: Finite and Infinite Machines. Prentice Hall (1967)
23. Barendregt, H.P.: Lambda calculi with types. In Abramsky, S., Gabbay, D.M., Maibaum, T.S.E., eds.: Background: Computational Structures. Vol. 2 of Handbook of Logic in Computer Science. Oxford University Press (1992)
24. Brookes, S.: The essence of Parallel Algol. In: LICS 11 (1996)
25. McCusker, G.: Games and Full Abstraction for a Functional Metalanguage with Recursive Types. Distinguished Dissertations. Springer-Verlag Limited (1998)
26. Roscoe, W.A.: Theory and Practice of Concurrency. Prentice-Hall (1998)

Linear-Time List Decoding in Error-Free Settings
(Extended Abstract)

Venkatesan Guruswami[1] and Piotr Indyk[2]

[1] Department of Computer Science & Engg., University of Washington, Seattle, WA 98195.
venkat@cs.washington.edu
[2] Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA 02139.
indyk@mit.edu

Abstract. This paper is motivated by the program of constructing list-decodable codes with *linear-time encoding and decoding algorithms* with rate comparable to or even matching the rate achieved by the best constructions with polynomial encoding/decoding complexity. We achieve this for three basic settings of list decoding, and view these as the first promising steps in the above general program.

First is a setting, which we call "mixture recovering", where for each position, the symbols of ℓ codewords are given in a scrambled order, and the goal is to recover each of the ℓ codewords. This was one of the first models studied by Ar *et al* in their influential paper [5] and they gave a polynomial time solution with rate $1/\ell$ using Reed-Solomon codes. We propose an elegant expander-based construction with rate $\Omega(1/\ell)$ with linear-time encoding/decoding complexity.

Second is the setting of "list-recovering" where the input is a set of ℓ possibilities for the value at each coordinate of the codeword and the goal is to find all the consistent codewords. We give an explicit linear-time encodable/decodable construction which achieves rate that is polynomial in $1/\ell$ (the best rate known for polynomial decoding complexity is $\Omega(1/\ell)$).

Third is the setting of decoding from erasures where a certain fraction of the symbols are erased and the rest are received intact. Here, for every $\varepsilon > 0$, we present an explicit construction of binary codes of rate $\Omega(\varepsilon^2 \log^{-O(1)}(1/\varepsilon))$ which can be encoded and list decoded from a fraction $(1-\varepsilon)$ of erasures in linear time. This comes very close to the best known rate of $\Omega(\varepsilon^2)$ for polynomial decoding complexity. For codes over larger alphabets, we can even approach the optimal rate of $\Omega(\varepsilon)$ with linear time algorithms — specifically, we give linear-time list decodable codes of rate $\tilde{\Omega}(\varepsilon^{1+1/\alpha})$ over alphabet size 2^α to recover from a fraction $(1-\varepsilon)$ of erasures.

1 Introduction

List decoding is an area of research in algorithmic coding theory that has seen much recent activity. The basic principle behind list decoding is to deal with "high noise" situations where unambiguous recovery of the message is impossible, and to report a small list consisting of all candidate messages. An example is the noise model where a certain fraction ρ of symbols can be adversarially corrupted for some $\rho > 1/2$; in such a case where there is more noise than correct information, unambiguous decoding is impossible, but surprisingly there are now codes and list decoding algorithms known

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 695–707, 2004.

for an arbitrary error fraction $\rho < 1$ that can pin down the possible messages to a small list (whose size can be a constant independent of the message length).

Mathematically, we recall that a code is simply a map $C : \Sigma^k \to \Sigma^n$ where Σ is the alphabet of the code. The ratio k/n, called *rate,* quantifies the efficiency of the code in terms of the amount of redundancy it adds, and is one of the key parameters of a code. While we would like the rate to be as high as possible, this usually conflicts with the noise-tolerance feature desired in the code. The central pursuit in algorithmic coding theory can therefore be summarized as constructing codes with good or optimal rate vs. noise-resilience trade-off, together with asymptotically fast (eg. linear time) algorithms to perform encoding as well as decoding (under the noise model in question).

In this paper, we are interested in codes with asymptotically optimal, namely linear complexity, encoding and list decoding algorithms, under various (adversarial) noise models. Arguably the most important noise-model is one where up to an arbitrary fraction ρ of symbols could be corrupted. For error thresholds $\rho < 1/2$, unambiguous decoding is possible and a couple of years back, the authors, building upon the works of [13,15, 4], met the goal of the above program by giving linear time encodable/decodable codes with near-optimal rate [9]. For $\rho > 1/2$, one must resort to list decoding, and progress on decoding algebraic codes [5,14,11] led to polynomial time list decodable codes for any $\rho < 1$ with rate $(1 - \rho)^2$, and the encoding/decoding times were subsequently improved to $O(n \log^{O(1)} n)$ [7,2]. The algebraic nature of these codes seemed to preclude further improvement of the runtime to linear in the block length, since the decoding algorithms involve some basic algebraic subroutines for which linear time algorithms have remained elusive for decades.

In a recent paper [10], the authors overcome this difficulty and show how to construct expander-based codes that are encodable and list-decodable from a fraction $(1 - \varepsilon)$ of errors for any $\varepsilon > 0$. The main technical contribution of that paper are codes that are *list-recoverable* in linear time. In the list-recovering setting, each position of the received word holds a *list* of several (say, l) different symbols. The goal of the decoding process is to recover a codeword c, such that for "most" positions i, the symbol c_i is contained in the i-th list of the received word. In [10], the authors construct such codes (equipped with linear-time encoding and decoding procedures) and then show, via a simple transformation, how one can get linear-time list-decodable codes using the list-recoverable codes.

Unfortunately, a major drawback of the codes of [10] is their rate, which is inversely doubly-exponential in l. This results in list-decodable codes (for decoding radius $(1-\varepsilon)$) with rate doubly exponentially small in $1/\varepsilon$, which is quite inferior when compared with the $\Omega(\varepsilon^2)$ rate achievable via algebraic codes like Reed-Solomon codes. At the same time, it is quite likely that better rate for linear time codes can be obtained through better understanding of list-recoverable codes.

Motivated by this, in this paper we investigate list-recoverability in its simplest setting that requires that *all* symbols of c must be included in the corresponding lists (we call it an *error-free* scenario). We show that, for this case, one can construct linear-time codes whose rate degrades polynomially in $1/l$, which compares much better with the $\Omega(1/l)$ rate achieved by the best known constructions with polynomial decoding complexity (like Reed-Solomon codes). This gives hope for this line of research to eventually meet

the grand goal of constructing linear-time list-decodable codes with rate not much worse than, and perhaps even comparable to, the rate of the best known polynomial time list-decodable codes.

We feel that error-free list-recoverable codes are interesting in their own right and serve as an initial, yet non-trivial, test-bed for new list decoding techniques. Moreover, they are useful in constructing erasure codes. We recall that erasures are a noise model where symbols in an arbitrarily chosen subset of certain size get erased and the rest are received intact. While erasures are typically easier to deal with than errors, erasure codes can also achieve better trade-offs. Therefore, obtaining optimal erasure codes does not just follow from codes developed for errors, and different techniques, streamlined to take advantage of the simpler model, might be required to deal with erasures. We use our result on error-free list-recoverable codes to construct binary codes, that are linear-time list-decodable from a fraction $(1-\varepsilon)$ of *erasures*. The codes have rate $\Omega(\varepsilon^2/\log^{O(1)}(1/\varepsilon))$, which is only off by polylogarithmic factors compared to the best known rate $\Omega(\varepsilon^2)$ for constructions with polynomial encoding/decoding complexity [8]. (The best rate possible is $\Omega(\varepsilon)$ but we do not know any explicit constructions achieving such a rate.) Moreover, for codes over larger alphabets, we can even approach the optimal rate of $\Omega(\varepsilon)$ with linear time algorithms — specifically, we give linear-time list decodable codes of rate $\tilde{\Omega}(\varepsilon^{1+1/a})$ over alphabet size 2^a to recover from a fraction $(1-\varepsilon)$ of erasures. Thus, we are able to "add on" linear encoding/list-decoding complexity without significant sacrifice in the rate of the code.

We also construct linear-time codes with rate that **matches** (up to constant factors) the best known polynomial time constructions for a simpler model of error-free list-recovering, which we call *mixture recovering*. In fact, our rate is *optimal* up to constant factors for the alphabet size we achieve (and we believe it should also be optimal irrespective of alphabet size). The problem of mixture recovering is the following. Let $C \subseteq [q]^n$ be a q-**ary** code of block length n. For some l codewords $c_1, c_2, \ldots, c_l$, we are given as input, for each i, $1 \leq i \leq n$, the multiset $\{(c_1)_{|i}, (c_2)_{|i}, \ldots, (c_l)_{|i}\}$ where $(c_j)_{|i}$ is the i'**th** symbol of c_j. In other words, the i'**th** symbols are given scrambled in an adversarially chosen order, for each i. The goal is to recover the codewords $c_1, c_2, \ldots, c_l$.[1] While this is admittedly a rather simplistic situation, it is worth pointing out this model was defined and considered by Ar, Lipton, Rubinfeld, and Sudan [5], and it spurred the subsequent successful research on more complicated noise models in [5,14]. Thus, it is interesting that for this model we can now, finally, achieve the parameters of [5] with linear time encoding/decoding algorithms, and it is our hope that similar success will ensue for linear complexity decoding under more complicated models.

In addition to their coding-theoretic relevance, mixture recoverable codes are also closely related to a well-studied concept in combinatorics that has numerous applications, viz. *superimposed codes* or *cover-free families*. An (N, l)-**superimposed** code is a family of N subsets of a universe U such that no set is contained in the union of l other sets. The goal is to have the universe size $|U|$ as small as possible. Note that in such a family, given the union of at most l sets, it is possible to correctly identify the sets involved in the union — the "decoding" problem for superimposed codes corresponds to precisely this

[1] The code should also have the combinatorial property that no other "spurious" codeword c should satisfy $c_{|i} \in \{(c_1)_{|i}, (c_2)_{|i}, \ldots, (c_l)_{|i}\}$ for every i.

task. Non-constructively, it is known that $|U| = O(l^2 \log N)$ is possible, and the best explicit constructions (based on Reed-Solomon and algebraic-geometric codes) achieve a size of $O(l^2 \log^2 N)$ and $O(l^3 \log N)$ together with polynomial time decoding. Using our new mixture-recoverable codes, we give an explicit construction of size $O(l^4 \log N)$ equipped with linear time decoding.

Connection to Zero-Error Capacity. We also mention an interesting relation between error-free list-recoverable codes and the zero-error capacity of a combinatorial channel under list decoding [6]. In the latter model, it is assumed that the communication channel can replace any symbol a by any symbol from a small set $N(a)$. Such a channel can be modeled by a graph G with edges of the form $\{a, b\}$, $b \in N(a)$. Our result implies existence of linear-time codes which work for *any* graph G with bounded degree; the rate of our codes is inversely polynomial in the degree bound.

Overview of the techniques. The basic approach that we use in this paper is similar to the approach of our earlier works [9,10]. In particular, our codes are constructed using "high-quality" expander graphs. Our result on mixture recoverable codes (Section 4) is obtained by a novel analysis of the well-known "ABNNR" expander code scheme [3] when the "left code" is an erasure-decodable code of large distance. The decoding algorithm ends up being quite simple and elegant. For our result on error-free list-recovering (Section 3, we use several layers of expander graphs cascaded together, similar to the codes in [10]. The decoding proceeds recursively from right to left. Each layer enables reducing the length of lists of candidate symbols. Finally, the codewords are decoded using uniquely-decodable codes.

Using just these techniques, however, would only result in codes with rate $2^{-O(l)}$. This is due to the fact that, using the approach of [10], one could reduce the list length by only an *additive* factor per layer. In contrast, in this paper we show, by a careful analysis, how to reduce the list length by a *constant multiplicative* factor per layer. This allows us to reduce the number of layers to $O(\log l)$ and achieve the claimed rate.

2 Preliminaries

Decoding. For any alphabet set Σ, and any two vectors $x, y \in \Sigma^n$, we use $D(x, y)$ to denote the Hamming distance between x and y (i.e., the number of positions on which x and y differ).

The notion of list-recoverability used in this paper is as follows. Recall that a code C of block length n over alphabet Σ is simply a subset of Σ^n, and elements of C are called codewords. The (minimum) distance, say d, of the code is smallest Hamming distance between a pair of distinct codewords; the *relative distance* is defined to be the normalized quantity d/n. Such a code C is said to be (ρ, l, L)-*(list) recoverable,* if for any sequence $\mathcal{L}$ of n lists $L_1 \ldots L_n$, where $L_j \subseteq \Sigma$, $|L_j| \leq l$ for at least ρ fraction of j and $L_j = \Sigma$ for remaining j's, there are at most L codewords $c \in C$ such that $c \in L_1 \times \ldots \times L_n$. We say that C is (ρ, l, L)-*(list) recoverable* in time $T(n)$, if there is a procedure which finds the list of (at most L) such codewords in time $T(n)$ given the lists $L_1, \ldots, L_n$.

Expanders. All code constructions in this paper use *expanders*. A bipartite graph $G = (A, B, E)$ is an (α, α')-**expander,** if for any $X \subseteq A$, $|X| \geq \alpha|A|$, the set $\Gamma(X)$ of X's neighbors in B has size at least $\alpha'|B|$. However, in addition to this notion, we will use more general *isoperimetric* properties of graphs. Specifically, we will make use of the following fact.

Fact 1 *Let $G = (V, E)$ be a* ***d-regular*** *graph on n nodes with second eigenvalue λ. Then the set of edges between a pair of subsets of vertices X and Y, denoted E(X, Y), satisfies the inequality:*

$$\left| \frac{|E(X,Y)|}{d|X|} - \frac{|Y|}{n} \right| \leq \frac{\lambda}{d} \sqrt{\frac{|Y|}{|X|}} . \tag{1}$$

It is known [12] how to construct graphs (called *Ramanujan graphs*) which achieve $\lambda = 2\sqrt{d-1}$.

Expander codes. For the purpose of constructing codes using expanders, we use the following scheme, first proposed in [3]. Assume we are given a code $C \subseteq \{0 \ldots q-1\}^n$, and a bipartite graph $G = (A, B, E)$ with $|A| = |B| = n$ and with right degree d. Given these two components, we construct the code C' in the following way. For any $x \in \{0 \ldots q-1\}^n$, define $G(x)$ to be a vector $y \in (\{0 \ldots q-1\}^d)^n$ created as follows. For $j \in B$ let $\Gamma_k(j)$ be the k-**th** neighbor of j in A, for $k = 1 \ldots d$. The j-**th** symbol y_j of y is defined as $\langle x_{\Gamma_1(j)}, \ldots, x_{\Gamma_d(j)} \rangle$. In other words, we "send" a copy of each symbol x_i along all edges going out of the vertex i, and the symbol y_j is obtained by concatenating all symbols "received" by j. The code $C' = G(C)$ is now obtained by taking all vectors $G(c)$ for $c \in C$. When talking about such a construction, we will refer to A and B as the left and right sides respectively, the code C which "sits" on the side A will be accordingly called the "left" code.

It is easy to see [3] that if C has minimum distance αn, and G is an (α, α')-**expander,** then the minimum distance of C' is at least $\alpha' n$. Thus, the construction has "distance amplification" property. The price for that is the decrease in rate (by a factor $1/d$ compared to the rate of C) and larger alphabet (of size q^d).

3 Linear-Time List-Recoverable Codes with Polynomial Rate

In this section we will construct linear-time $(1-\alpha, l, l)$-**(list)** recoverable codes with rate $l^{-O(1)}$. The fraction of erasures, α, is to be thought of as small but an absolute constant, and the parameter that grows is l. (So when we say the rate is $\Omega(l^{-O(1)})$, the hidden constants could depend on α.) We will specify exact values for the various parameters towards the end of this section after setting up constraints that they must obey.

When $l = 1$, we just need a linear time code to correct a fraction α of erasures, and we know such constructions for any $\alpha < 1$ [4].

Our construction of a $(1-\alpha, l, l)$-**recoverable** code will be recursive. We will assume that we have a construction of such a code $C_{l'}$ for list length $l' = (1-\gamma)l$ ($\gamma > 0$ is yet another absolute constant to be specified later), and from that we will construct an $(1-\alpha, l, l)$-**recoverable** code C_l. We will lose a further constant factor in the rate and doing so $O(\log l)$ times in the recursion will give us rate polynomially small in $1/l$, and we will maintain linear time encoding and list recovering algorithms.

3.1 The Recursive Construction

We now describe this recursive construction. Let $C_{l'}$ be a code over an alphabet $\Sigma_{l'}$ of size $Q_{l'}$, with rate $r_{l'}$ and block length n. Assume that $C_{l'}$ can be encoded in linear time and can be $(1-\alpha, l', l')$-list recovered as well as unique decoded from a fraction $(1-\alpha)$ of erasures in linear time. To construct C_l which will be $(1-\alpha, l, l)$-recoverable, we need two expander graphs:

- an $n \times n$ bipartite graph G_1 with degree d_1, such that every set of αn left nodes has at least $(1-\beta)n$ neighbors on the right side, where $0 < \beta < 0.1$ will be yet another absolute constant. (This is the "erasure-reducing" graph.) Note that there are explicit graphs with this property with $d_1 = O(1/\alpha\beta)$.
- a bipartite graph $G_2 = (V, E, E')$ constructed as follows. Take a d_2-regular graph $G = (V, E)$ that has the following expansion properties: for any $X, Y \subset V$, $|X|, |Y| \leq \frac{n}{10}$, the size of the cut $|E(X, V-X-Y)|$ is at least $|X|d_2/2$.[2] The constant d_2 is an "absolute" constant, and in particular does not depend on l or any other constants, and indeed it is easily seen by Fact 1 that a Ramanujan graph of degree $d_2 \geq 25$ will have such a property. Then G_2 is the "edge-vertex incidence graph" of G, i.e., for any $v \in V, e \in E$, we have $(v, e) \in E'$ iff v is incident to e. Note that the square graph $(G_2)^2$ projected on V is equal to the graph G. Also, observe that G_2 has left degree d_2 and right degree 2.

Given the above components, we construct C_l as $C_l = G_2(G_1(C_{l'}))$. Denote $N = |E| = nd_2/2$. The block length of C_l equals N, its rate is $r_{l'}/(d_2 d_1)$, and its alphabet is $\Sigma_l = \Sigma_{l'}^{2d_1}$. Let $\Sigma = \Sigma_{l'}^{d_1}$ denote the alphabet of the "intermediate" code $G_1(C_{l'})$. By the construction, it is easy to check that the following properties can be ensured:

1. The intermediate code $G_1(C_{l'})$ can be *unique* decoded in linear time from a fraction 0.9 of erasures. This follows in the obvious way by pushing symbols back from the right side of G_1 to its left side, thus obtaining a received word for $C_{l'}$. By the expansion property of G_1, this will have at most a fraction α of erasures, which can be removed by running the erasure decoding algorithm for $C_{l'}$.
2. The relative distance of C_l is large enough to ensure that it is combinatorially $(1-\alpha, l, l)$-recoverable, i.e., the output list size only needs to be l.
3. The rate of C_l is $1/l^{O(1)}$. Specifically, it is equal to $\Theta\left((\alpha\beta)^{\log_{1/(1-\gamma)} l}\right)$.

3.2 The Decoding

Let $\mathcal{L}$ be a collection of N lists $L_1, L_2, \ldots, L_N$ of which at most αN are equal to Σ_l (i.e. are erasures) and the remaining have size at most l each. Our goal is to output a list of all codewords c in C_l for which $c_j \in L_j$ for all j.

Consider the way symbols are distributed by G_2. For each edge (i, j) in G_2, let $L_2(i, j) \subseteq \Sigma$ denote the set of symbols that L_j "suggests" for the i-th symbol of the left codeword. More formally, $L_2(i, j)$ contains symbols a_k, such that $\langle a_1, \ldots a_d \rangle \in L_j$

[2] The constant 10 is picked just for definiteness; other (smaller) absolute constants would probably work just as fine.

and $\Gamma_k(j) = i$. Note that $L_2(i,j)$ is a set, so duplicates are removed. For each i (left node of G_2), define $K_i = \cap_{\{i,j\} \in E} L_2(i,j)$.

Let $I \subseteq \{1, 2, \ldots, n\}$ be the set of indices i such that $|K_i| \leq l(1-\gamma)$.

Case1: $|I| > \beta n$.

Let T be the set of left nodes of G_1 which have at least one neighbor in I. By the expansion property of G_1, we have $|T| \geq (1-\alpha)n$. For each $i' \in T$, define $\tilde{L}_{i'}$ to be the symbols corresponding to position i' in the list K_i where $i \in I$ is an arbitrary node for which (i', i) is an edge in G_1. Note that each $\tilde{L}_{i'}$ has at most l elements. For $i' \notin T$, define $\tilde{L}_{i'}$ to be $\Sigma_{l'}$, the alphabet of the code C_l. We are now in a position to complete the decoding using the algorithm for C_l, since at most a fraction α of positions are erased (i.e. have $\tilde{L}_{i'} = \Sigma_{l'}$).

Note that the strong vertex-expansion property of G_1 enables us to go from a fraction β of non-erasures to a fraction α of erasures.

The above dealt with the case when a left node of G_2 had some neighbor whose list size was at most l' (as opposed to l), so we could recurse. Now we focus on the other case, in which we have to work some more before being able to recurse.

Case 2: $|I| \leq \beta n$.

In this case, the decoding proceeds as follows. Impose an arbitrary ordering on elements in $L_2(i,j)$ and K_i. Consider a graph $HH = (A, B, E'')$, where $A = V \times \{1 \ldots l\}$, $B = E \times \{1 \ldots l\}$ (recall that E is the right vertex set of G_2). The set E'' is constructed as follows: we put an edge between $(i,t) \in A$ and $(j,s) \in B$ iff the following conditions are satisfied:

- (i,j) is an edge in G_2,
- $(L_2(i,j))_s = (K_i)_t$, and there is no other s' with the same property for the given value of t

Consider any codeword $c \in G_1(C_{l'})$ such that $G_2(c) \in L_1 \times \ldots \times L_N$. Our algorithm uses the graph HH to recover c. The intuition behind the construction is as follows. The set A represents the choices of symbols for the decoded codeword of $G_1(C_{l'})$. Specifically, let $V(c) = \{(i,t) : c_i = (K_i)_t\}$. Selecting elements of A as members of $V(c)$ corresponds to making a choice about the symbols of c. Similarly, selecting an element $(j,s) \in B$ corresponds to making a choice about an element $a \in L_i$, interpreted as $G_2(c)_i = a$. Thus, the decoding problem can be phrased as finding sets $V \subset A$ and $W \subset B$ such that the codeword induced by V is "consistent" with the list elements induced by W. Our algorithm will generate *all* such set pairs (that are "significantly" different from each other), and in this way decode any desired c.

In the following, instead of dealing with the graph HH itself, we will deal with its square. More specifically, we will consider $H = (HH)^2_{|A}$. The decoding algorithm is as follows:

1. Compute all connected components $S_1 \ldots S_k$ of H that have size at least $\frac{n}{10}$. We will ensure that $\sum_i |S_i| \geq \delta l n$, for certain constant $\delta > 0$.
2. For each S_r, $1 \leq r \leq k$, do the following:

a) Construct a vector c' such that $c'_i = (K_i)_t$ iff $(i,t) \in S_r$ (we will prove in Lemma 1 that there can be most one such value of t for each i, so this is well defined). Set all unspecified values of c' as erasures.
b) Run the erasure decoding algorithm for $G_1(C_{l'})$ that can correct a fraction 0.9 of erasures to recover a codeword $c'' \in G_1(C_{l'})$, if any, that is consistent with c'.
c) If $G_2(c'')_j \in L_j$ for at least $(1-\alpha)N$ values of $j \in \{1, 2, \ldots, N\}$, then output $G_2(c'')$.

3. The above dealt with the codewords that were consistent with the large components. We now deal with any possible remaining codewords.
a) Compute $Q_i = \{(K_i)_t : (i,t) \notin \bigcup_{r=1}^{k} S_r\}$
b) Compute $K'_i = K_i - Q_i$. Observe that $\sum_i |K'_i| \leq nl(1-\delta)$.
c) Let $I' = \{i : |K_i| \leq l(1-\delta)/(1-\beta)\}$. Observe that $|I| \geq \beta n$. We will ensure that $(1-\delta)/(1-\beta) \leq 1-\gamma$.
d) Proceed as in Case 1 of the algorithm with I' playing the role of I.

3.3 Proof of Correctness and Runtime Analysis

In the following, we show correctness of the algorithm, and analyze its running time. The correctness can be established by the following two lemmas. The first lemma implies that the received word c' is well-defined in Step 3(a) as well as the correctness of step 4(b) of the algorithm. Specifically, it follows that for every i, $c'_i \notin Q_i$, and therefore $c'_i \in K'_i$. Thus, we can continue decoding recursively as in Case 1 described earlier.

Lemma 1. *For c and any S_i as above, we either have $S_i \subset V(c)$ or $S_i \cap V(c) = \emptyset$.*

Proof: It suffices to show that if $(i,t), (i',t')$ are connected in H, and $c_i = (K_i)_t$, then $c_{i'} = (K_{i'})_{t'}$. The latter fact can be easily shown by induction on the length of path connecting (i,t) and (i',t') in H. For both the base case and the inductive step, it suffices to show that if $\langle (i,t), (j,s) \rangle, \langle (i',t'), (j,s) \rangle \in HH$ and $c_i = (K_i)_t$, then $c_{i'} = (K_{i'})_{t'}$ This can be done as follows. Firstly, by construction of HH, there is no other $s' \neq s$ such that $L_2(i,j)_{s'} = c_i$. This implies $G_2(c)_j$ is equal to the s-th element L_j. By the construction of HH, this means that $(K_{i'})_{t'} = c'_i$. □

It remains to show the guarantee mentioned in Step 1 of the algorithm. Let $S_1 \ldots S_t$ be all connected components of H, sorted in the decreasing order of their size. Then k is the largest index such that $|S_k| \geq \frac{n}{10d_2}$.

Lemma 2. *Let $\delta = 1 - 4\big(1 - (1 - \alpha d_2 - \beta)(1 - 2\gamma)\big)$. Then $\sum_{i \leq k} |S_i| \geq \delta nl$.*

Proof: The main idea of the proof is as follows. Firstly, we show that H has "fair" expansion properties. Specifically, we show that there is a graph H_l which is an expander, such that H can be obtained from H_l by removing only very few edges. It will follow that H cannot have many small connected components. Firstly, we establish a lower bound on the number of edges in H, which will enable us to prove that H is obtained from H_l by removing only few edges.

Claim. There are at least $nd_2l((1 - \alpha d_2 - \beta)(1 - 2\gamma) - 1/2)$ edges in H.

Proof: Define $J \subseteq \{1, 2, \ldots, n\}$ to be the set of i's for which at least one of the d_2 neighbors, say j, of i in G_2 has an erasure at that location, i.e. $L_j = \Sigma_l$. Clearly, $|J| \leq 2\alpha N = \alpha d_2 n$. Consider any $i \notin I \cup J$, and j such that (i, j) is an edge in G_2. Since $K_i \subset L_2(i, j)$ for any j, it follows that for every $t \leq |K_i|$ there exists s such that $L_2(i, j)_s = (K_i)_t$. Since $|K_i| \geq (1 - \gamma)l$ and $|L_2(i, j)| \leq l$ (former because $i \notin I$ and latter because $i \notin J$), it follows that for at most $2\gamma l$ of t's the corresponding s is not unique. Thus, the edge set E'' of HH contains at least $(1 - 2\gamma)l$ edges of the form $(i, \cdot) - (j, \cdot)$ for each i, j as above. Therefore, HH has at least $(1 - \alpha d_2 - \beta)(1 - 2\gamma) n d_2 l$ edges. Now, since H is defined as $(HH)^2_{|A}$, and each node on the right hand side B of HH has degree at most 2, it follows that each missing edge from HH can cause at most one edge to be missing from H. It follows that the number of edges in H is at least

$$nd_2 l/2 - nd_2 l(1 - (1 - \alpha d_2 - \beta)(1 - 2\gamma)) = nd_2 l((1 - \alpha d_2 - \beta)(1 - 2\gamma) - 1/2) \,. \square$$

In the next step, we investigate the "ideal" setting, where the graph H has the maximum possible $nd_2 l/2$ edges. For this purpose we need the following definition.

Definition 1. *A graph $H_l = (V \times \{1 \ldots l\}, E_l)$ is a l-copy of a graph $G = (V, E)$, if for every edge $\{i, j\} \in E$ there is a permutation $\pi : \{1 \ldots l\} \to \{1 \ldots l\}$ such that all edges $\{(i, t), (j, \pi(t))\}$ are present in E_l, for $t = 1 \ldots l$, and these are the only edges in H_l. (Note that the definition allows the permutations π for different edges to be different.)*

Claim. The graph H is a subgraph of an l-copy H_l of G, obtained by removing at most $nd_2 l\big(1 - (1 - \alpha d_2 - \beta)(1 - 2\gamma)\big)$ edges.

Proof: Follows from the easily verified facts that if (i, t) and (i', t') are adjacent in H, then (i, i') is an edge in G, and there is no other $s' \neq t'$ for which (i, t) is adjacent to (i', s') in H. $\square$

An l-copy of an expander graph is an expander, as formalized by the following claim.

Claim. For any $X \subset A \times \{1 \ldots l\}$, $|X| \leq \frac{n}{10}$, the number of edges in H_l that are incident to exactly one vertex in X is at least $d|X|/2$.

Proof: Let $X_1, X_2, \ldots, X_l$ be the "projections" of X based on the second coordinate, i.e. $X_j = \{x \in V \mid (x, j) \in X\}$. By the expansion properties of G, since $|\bigcup_j X_j| \leq n/10$, for each j, the number of edges leaving X_j and landing outside $\bigcup_{r \neq j} X_r$ is at least $d_2|X_j|/2$. It follows that at least $d_2|X|/2$ edges of H have exactly one endpoint in X. $\square$

Now we can proceed with the proof of the Lemma. Consider the "small" connected components $S_{k+1} \ldots S_t$. In H_l, there would be $|S_i|d_2/2$ edges going out of each S_i. Thus, the number of edges that appear in H_l but not in H is at least $\frac{1}{2}\sum_{i>k} |S_i|d_2/2$. This number must be smaller than $nd_2 l\big(1 - (1 - \alpha d_2 - \beta)(1 - 2\gamma)\big)$. Therefore

$$\sum_{i>k} |S_i| \leq 4nl\big(1 - (1 - \alpha d_2 - \beta)(1 - 2\gamma)\big) \,,$$

which completes the proof of Lemma 2. $\square$

Theorem 1. *There exists an absolute constant $\alpha_0 \geq 0$ ($\alpha_0 = 0.002$ will work), such that for every integer $l \geq 1$, there exists a polynomial time constructible family of codes with rate $1/l^{O(1)}$ over an alphabet of size $2^{l^{O(1)}}$, such that a code of block length N in the family is encodable in time $O(Nl)$ and $(1-\alpha_0, l, l)$-list recoverable in time $O(Nl)$.*

Proof: First we need to fix the values of the constants $\alpha, \beta, \gamma, \delta$ that satisfy

$$1 - \delta = 4(1 - (1 - \alpha d_2 - \beta)(1 - 2\gamma)) \tag{2}$$

$$1 - \gamma \geq (1 - \delta)/(1 - \beta) \tag{3}$$

This can be satisfied e.g., by setting $\alpha = 0.1/d_2$, and $\gamma = \beta = 0.05$ (which yields $\delta = 0.23$). Note that we have not tried to optimize the parameters and just wanted to get the qualitative statement of rate that is only polynomially small in $1/l$.

The rate and encoding time follows from earlier analysis. The claim about alphabet size follows by unwinding the recurrence $\Sigma_l = \Sigma_{l'}^{O((\alpha_0\beta)^{-1})} = \Sigma_{(1-\gamma)l}^{O(1)}$. For the decoding time, observe that at the recursive level where the list size goes down from l to $(1-\gamma)l$, we perform at most $10l$ erasure decodings each taking $O(nd_2)$ time (since each of the large connected components has size at least $n/10$, there are at most $10l$ of them), followed by a single tail recursive call to a decoding for list size $(1-\gamma)l$. Solving this recurrence, the total decoding time is $O(nd_2l) = O(Nl)$, as claimed. □

4 Linear-Time Linear-Rate Mixture Recoverable Codes

In this section we show that the construction presented in the previous section can be (significantly) simplified and used to construct linear-time mixture-recoverable codes that match the best known rate.

Definition 2. *A code C is said to be* l-**mixture** recoverable, *if for every sequence of distinct $c_1 \ldots c_l \in C$, for a sequence $\mathcal{L}$ of n multisets $L_1 L_2 \ldots L_n$, where $L_j = \{(c_1)_i \ldots (c_l)_i\}$, there is no other codeword $c \in C \setminus \{c_1, c_2, \ldots, c_l\}$ such that $c \in L_1 \times \ldots \times L_n$. The algorithmic version of the problem is defined in an analogous way, where given the multisets L_i, $1 \leq i \leq n$, where each L_i has the i'th symbols of $c_1, c_2, \ldots, c_l$ in some order, the goal is to find the codewords $c_1, c_2, \ldots, c_l$.*

It is easy to see that if C is a code with relative distance greater than $1 - 1/l$, then C is (combinatorially) l-**mixture** recoverable; the challenge is to come up with "unscrambling" or decoding algorithms to find the l codewords *efficiently*. We show below how to construct codes that are l-**mixture** recoverable with rate $\Omega(1)/l$ and that are equipped with linear-time encoding and decoding procedures. This matches the rate of the best known explicit constructions that had polynomial decoding complexity (the first such result was due to Ar *et al* [5]), while providing optimal decoding time.

Theorem 2. *For every integer $l \geq 1$, there exists a polynomial time constructible family of codes over an alphabet of size $2^{O(l)}$ with rate $\Omega(1/l)$ that are encodable and* l-**mixture** *recoverable in linear time.*

Proof: We present the code construction, and defer the presentation of the algorithm to recover the codewords to the full version of the paper. Let C_0 be a code of relative distance $1 - \frac{1}{4l}$ and rate $\Omega(1/l)$ that is linear-time encodable and linear-time decodable from a fraction 0.9 of erasures. An explicit family of such codes is now known [9]. The block length of C_0 is denoted by n, and its alphabet by $Q = [q]$ where $q = 2^{O(l)}$. In addition, we use the bipartite graph $G_2 = (V, E, E')$ as in Section 3 (recall that G_2 was the edge-vertex incidence graph of a Ramanujan graph $G = (V, E)$ of degree an absolute constant d_2). Our final code C^* is obtained by taking $C^* = G_2(C)$. It is clear that C^* can be encoded in linear time, since C can be, and G_2 is a constant degree graph. Also, the rate of C^* is a factor d_2 smaller than that of C_0, and is thus $\Omega(1/l)$. The alphabet size of C^* is q^2 and thus $2^{O(l)}$ as claimed. Finally, since C has relative distance greater than $1 - 1/l$, so does C^* and thus C^* is (combinatorially) l-**mixture** recoverable. It remains to give a linear time mixture recovering algorithm for C^*. The details of this are omitted in this extended abstract. □

Our mixture recoverable codes from Theorem 2 can be used to give (N, l)-**superimposed** codes over a universe of size $O(l^4 \log N)$ which are furthermore linear time "decodable". The details of this connection are deferred to the full version of this paper.

5 Linear-Time Binary List-Decodable Codes for Erasures

We now use the list-recoverable codes from Section 3 to construct binary codes that are encodable as well as list decodable from a fraction $(1 - \varepsilon)$ of erasures in linear time, and whose rate is very close to the $\Omega(\varepsilon^2)$ bound which is the best known for polynomial decoding complexity [8].

Theorem 3. *For every constant $\varepsilon > 0$, there is a polynomial time constructible family of* binary *codes of rate $\Omega(\varepsilon^2 \log^{-O(1)}(1/\varepsilon))$ such that every code in the family can be encoded as well as list decoded from a fraction $(1 - \varepsilon)$ of erasures using lists of size $O(\log(1/\varepsilon))$ in time linear in the block length.*

Proof: The basic idea in the construction claimed in the above theorem is to start with a (ρ, l, l)-**recoverable** code C_l for $\rho = \alpha_0$ and $l = \Theta(\log(1/\varepsilon))$. Theorem 1 guarantees such a code of rate $\log^{-O(1)}(1/\varepsilon)$ and linear time encoding and (ρ, l, l)-**recovering** algorithms; let n_0 denote the block length of C_l. We then take a degree $D = O(1/\varepsilon)$ Ramanujan expander R and construct the code $C' = R(C_l)$. Specifically, R will be an $n_0 \times n_0$ D-regular $(1 - \rho, 1 - \varepsilon/2)$-**expander**, or in other words a bipartite graph with the property that for every set T of $\varepsilon n_0/2$ nodes on the right, at least a ρ fraction of the nodes on the left side of R have at least one neighbor in T. It is well-known that an explicit such graph can be constructed with degree $D = O(1/\varepsilon)$.

Note that rate of C' is $\varepsilon/\log^{O(1)}(1/\varepsilon)$ and its alphabet size, say 2^Q, is a constant that depends only on ε (the exact dependence will be $Q = O(\varepsilon^{-1} \log^{O(1)}(1/\varepsilon))$). Our final binary code C^* will be obtained by concatenating C' with inner code an appropriate binary erasure list-decodable code, call it C_{in}, with the choice $k = Q$ and $\zeta = \varepsilon/2$ in the following lemma (which is easily proven via the probabilistic method).

Lemma 3 (Follows from Lemma 10 in [9]). *For every $\zeta > 0$ and integer k, there exists a binary code $C_k : \{0,1\}^k \to \{0,1\}^n$ with $n = O(k/\zeta)$ such that C_k is $(\zeta, 1, O(\log(1/\zeta)))$-**recoverable**, i. e., given a received word with at most $(1-\zeta)n$ erasures, the number of codewords of C_k that agree with the received word in the non-erased positions is at most $O(\log(1/\zeta))$.*

Note that the overall rate of C^* equals the product of rates of C' and C_{in} is thus $\Omega(\varepsilon^2 \log^{-O(1)}(1/\varepsilon))$, and its block length $N = n_0 n_1$. Details of the erasure decoding algorithm for C^* are omitted and will appear in the full version of the paper. □

Improving rate for larger alphabets. We can improve the above quadratic rate using a technique from [1,9] that involves taking several, say a, appropriately chosen binary concatenated codes of the same dimension and block length and *juxtaposing* them together, i.e., a message is encoded by the a codes independently, and the i**'th** bits of the encodings are put together to give an encoded string over alphabet size 2^a.

Theorem 4. *For every $\varepsilon > 0$ and every integer $a \geq 1$, there exists a polynomial time constructible code family over an alphabet of size 2^a with rate $\Omega(\varepsilon^{1+1/a} \log^{-O(1)}(1/\varepsilon))$ which is linear-time encodable and linear-time list decodable from a fraction $(1-\varepsilon)$ of erasures using lists of size $O(a \log(1/\varepsilon))$.*

References

1. Andres Albanese, Johannes Blomer, Jeff Edmonds, Michael Luby, and Madhu Sudan. Priority encoding transmission. *IEEE Transactions on Information Theory,* 42(6): 1737–1744, November 1996.
2. Michael Alekhnovich. Linear diophantine equations over polynomials and soft decoding of Reed-Solomon codes. *Proceedings of the Symposium on Foundations of Computer Science,* pages 439–448, 2002.
3. Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ronny Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory,* 38:509–516, 1992.
4. Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science,* pages 512–519, 1995.
5. Sigal Ar, Richard J. Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from mixed data. *SIAM Journal on Computing,* 28(2):487–510, 1999.
6. Peter Elias. Zero error capacity under list decoding. *Quarterly Progress Report ,Research Laboratory of Electronics, MIT,* 48:88–90, 1958.
7. G. L. Feng. Two fast algorithms in the Sudan decoding procedure. *Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing,* pages 545–554, 1999.
8. Venkatesan Guruswami. List decoding from erasures: Bounds and code constructions. *IEEE Transactions on Information Theory,* 49(11):2826–2833, 2003.
9. Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. *Proceedings of the Symposium on Theory of Computing,* pages 812–821, 2002.
10. Venkatesan Guruswami and Piotr Indyk. Linear-time encodable and list-decodable codes. *Proceedings of the Symposium on Theory of Computing,* pages 126–135, 2003.

11. Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory,* 45:1757–1767, 1999.
12. Alex Lubotzky, R. Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica,* 8(3):261–277, 1988.
13. Daniel Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory,* 42(6): 1723–1732, 1996.
14. Madhu Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity,* 13(1):180–193, 1997.
15. Gillés Zémor. On expander codes. *IEEE Transactions on Information Theory,* 47(2):835–837, 2001.

A Categorical Model for the Geometry of Interaction

Esfandiar Haghverdi[1] and Philip Scott[2*]

[1] School of Informatics & Department of Mathematics,
Indiana University, Bloomington, Indiana, USA
ehaghver@indiana.edu
[2] Department of Mathematics & Statistics, University of Ottawa,
Ottawa, Ontario, K1N 6N5, CANADA
phil@mathstat.uottawa.ca

Abstract. We consider the multiplicative and exponential fragment of linear logic (MELL) and give a Geometry of Interaction (GoI) semantics for it based on unique decomposition categories. We prove a Soundness and Finiteness Theorem for this interpretation. We show that Girard's original approach to GoI 1 via operator algebras is exactly captured in this categorical framework.

1 Introduction and Motivation

Girard introduced his Geometry of Interaction (GoI) program in the late 80's, through a penetrating series of papers [10,9,11].

The Geometry of Interaction was the first attempt to model, in a mathematically sophisticated way, the dynamics of cut-elimination. Traditional denotational semantics models normalization of proofs (or lambda terms) by static equalities: if Π, Π' are proofs of a sequent $\Gamma \vdash A$ and if we have a reduction $\Pi \succ \Pi'$ by cut-elimination, then their interpretations $[\![-]\!]$ in any model denote equal morphisms, i.e. $[\![\Pi]\!] = [\![\Pi']\!] : [\![\Gamma]\!] \to [\![A]\!]$. On the other hand *syntax* contains too much irrelevant information and does not yield an independent mathematical modelling of the dynamics of cut-elimination. Thus the goal of GoI is to provide precisely such a mathematical model.

The first implementation of this programme was given by Girard [10], based on the C^*-algebra of bounded linear operators on the space ℓ^2 of square summable sequences. For a much more elaborate account of the ideas above see [10,9,11].

The GoI interpretation was extended to untyped λ-calculus by Danos in [7]. Danos and Regnier further extended the GoI interpretation to define a *path-semantics* for proofs (=programs) and gave a detailed comparison with the λ-calculus notions of path. The idea is that a proof net is represented by a set of paths and the execution formula is an *invariant* of reduction (see [8]).

* Research supported by an operating grant from NSERC.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 708–720, 2004.

Abramsky and Jagadeesan gave the first categorical approach to GoI in [4]. Their formalisation is based on domain theory and arises from the construction of a categorical model of linear logic. The ideas and techniques used in [4] together with the development of traced monoidal categories, introduced by Joyal, Street and Verity [17], led to a more abstract formalisation of GoI via the notion of *GoI Situation* introduced by Abramsky in [2]. GoI Situations give a categorical embodiment of the essential ingredients of GoI, at least for the multiplicative and exponential fragment. Furthermore, in his Siena lecture [2] Abramsky introduced a general GoI construction. Abramsky's programme was sketched in [2] and completed in [12] and [3]. However, what was still missing was a tighter connection between the abstract GoI frameworks above and the original works of Girard et al. That is, we want our categorical models for GoI to be not only part of well-established categorical logic, but also we want our framework to explicitly connect with the details of the operator algebraic approach, e.g. the execution formula, orthogonality and the notion of type, all found in the original works but which could not be given in the generality of [3].

In this paper, we analyze how the first Girard paper GoI1 [10] fits into the general theory of GoI situations. The idea pursued here is to restrict the abstract traced monoidal categories in a GoI situation to a useful subclass: unique decomposition categories [12,13]. These are monoidal categories whose homsets are enriched with certain infinitary sums, thus allowing us to consider morphisms as matrices, the execution formula as an infinite sum, etc. Such categories are inspired from early categorical analyses of programming languages by Elgot, Arbib and Manes, et. al. (e.g. [18]).

The main contributions of this paper are the following:

1. We present a categorical model (implementation) for GoI and show that it captures the original Hilbert space model proposed by Girard in [10], including the notions of orthogonality and type.
2. We show that the execution formula at the heart of modeling computation as cut-elimination is perfectly captured by the categorical notion of trace.
3. We prove finiteness and soundness results for our model using the categorical properties of trace and GoI Situation.

We believe that our categorical interpretation views the original Girard GoI model in a new light. Not only do the original constructions appear less ad hoc, but this paper also opens the door towards accommodating other interesting models based on different categories and GoI Situations.

The rest of the paper is organized as follows: In Section 2 we recall the definitions of traced monoidal categories and GoI Situations, following [12,3]. In Section 3 we recall the definition of a unique decomposition category and give some examples. Sections 4 and 5 are the main sections of the paper where we discuss our categorical model for the GoI program and give the main theorems respectively. Section 6 discusses the original model introduced by Girard in [10]. Finally in section 7 we conclude by discussing related and future work.

2 Traced Monoidal Categories and GoI Situations

We recall the definitions of symmetric traced monoidal categories and GoI Situations. For more detailed expositions, see [12,3]. The categories introduced below admit a highly geometric presentation, but for lack of space, we omit drawing the pictures, and refer the reader to the above references.

Joyal, Street and Verity [17] introduced the notion of abstract trace on a balanced monoidal category (a monoidal category with braidings and twists.) This trace can be interpreted in various contexts where it could be called contraction, feedback, parametrized fixed-point, Markov trace or braid closure. The notion of trace can be used to analyse the cyclic structures encountered in mathematics and physics, most notably in knot theory. Since their introduction, traced monoidal categories have found applications in many different areas of computer science, for example the model theory of cyclic lambda calculi [14], categorical frameworks for the semantics of asynchronous communication networks [19], full completeness theorems for multiplicative linear logic via GoI models [12], analysis of finite state machines [16], relational dataflow [15], and independently arose in Stefanescu's work in network algebra [20].

Definition 1. A *traced symmetric monoidal category* is a symmetric monoidal category $(\mathbb{C}, \otimes, I, s)$ with a family of functions $Tr^U_{X,Y} : \mathbb{C}(X \otimes U, Y \otimes U) \to \mathbb{C}(X, Y)$ called a *trace,* subject to the following axioms:

- **Natural** in X, $Tr^U_{X,Y}(f)g = Tr^U_{X',Y}(f(g \otimes 1_U))$ where $f : X \otimes U \to Y \otimes U$, $g : X' \to X$,
- **Natural** in Y, $gTr^U_{X,Y}(f) = Tr^U_{X,Y'}((g \otimes 1_U)f)$ where $f : X \otimes U \to Y \otimes U$, $g : Y \to Y'$,
- **Dinatural** in U, $Tr^U_{X,Y}((1_Y \otimes g)f) = Tr^{U'}_{X,Y}(f(1_X \otimes g))$ where $f : X \otimes U \to Y \otimes U'$, $g : U' \to U$,
- **Vanishing (I,II)**, $Tr^I_{X,Y}(f) = f$ and $Tr^{U \otimes V}_{X,Y}(g) = Tr^U_{X,Y}(Tr^V_{X \otimes U, Y \otimes U}(g))$ for $f : X \otimes I \to Y \otimes I$ and $g : X \otimes U \otimes V \to Y \otimes U \otimes V$,
- **Superposing,** $Tr^U_{X,Y}(f) \otimes g = Tr^U_{X \otimes W, Y \otimes Z}((1_Y \otimes s_{U,Z})(f \otimes g)(1_X \otimes s_{W,U}))$ for $f : X \otimes U \to Y \otimes U$ and $g : W \to Z$,
- **Yanking,** $Tr^U_{U,U}(s_{U,U}) = 1_U$.

Joyal, Street, and Verity[17] also introduced the *Int* construction on traced symmetric monoidal categories $\mathbb{C}$; $Int(\mathbb{C})$ is a kind of "free compact closure" of the category $\mathbb{C}$. $Int(\mathbb{C})$ isolates the key properties of Girard's GoI for the multiplicative connectives, in that composition in $Int(\mathbb{C})$, which is defined via the trace, uses an abstract version of Girard's Execution Formula. Of course, one of our goals in this paper is to show that in our restricted models, this is exactly the original Girard formula.

The next problem was how to extend this to the exponential connectives. In the Abramsky program (see [3]) this is achieved by adding certain additional structure to a traced symmetric monoidal category. This structure involves a monoidal endofunctor, a reflexive object, and appropriate retractions, as introduced below. It was shown in [3] that this additional structure is sufficient to

generate certain *linear combinatory algebras* which capture the appropriate computational meaning of the exponentials.

Definition 2. A *GoI Situation* is a triple $(\mathbb{C}, T, U)$ where:

1. $\mathbb{C}$ is a traced symmetric monoidal category
2. $T : \mathbb{C} \to \mathbb{C}$ is a traced symmetric monoidal functor with the following retractions (note that the retraction pairs are monoidal natural transformations):
   a) $TT \lhd T$ (e, e') (Comultiplication)
   b) $Id \lhd T$ (d, d') (Dereliction)
   c) $T \otimes T \lhd T$ (c, c') (Contraction)
   d) $\mathcal{K}_I \lhd T$ (w, w') (Weakening). Here $\mathcal{K}_I$ is the constant I functor.
3. U is an object of $\mathbb{C}$, called a *reflexive object,* with retractions: (a) $U \otimes U \lhd U$ (j, k), (b) $I \lhd U$, and (c) $TU \lhd U$ (u, v).

For examples of GoI Situations see Section 6.

3 Unique Decomposition Categories

We consider monoidal categories whose homsets allow the formation of certain infinite sums. Technically, these are monoidal categories enriched in Σ-monoids (see below). In the case where the tensor is coproduct and Σ-monoids satisfy an additional condition, such categories were studied in computer science in the early categorical analyses of flow charts and programming languages by Bainbridge, Elgot, Arbib and Manes, et. al. (e.g. [18]). The general case, known as unique decomposition categories (UDC's), are particularly relevant for this paper, since they admit arbitray tensor product (not necessarily product or coproduct) and traced UDCs have a standard trace given as an infinite sum. For more facts and examples on UDCs see [12].

Definition 3. A *Σ-monoid* consists of a pair (M, Σ) where M is a nonempty set and Σ is a partial operation on the countable families in M (we say that $\{x_i\}_{i \in I}$ is *summable* if $\sum_{i \in I} x_i$ is defined), subject to the following axioms:

1. *Partition-Associativity Axiom.* If $\{x_i\}_{i \in I}$ is a countable family and if $\{I_j\}_{j \in J}$ is a (countable) partition of I , then $\{x_i\}_{i \in I}$ is summable if and only if $\{x_i\}_{i \in I_j}$ is summable for every $j \in J$ and $\sum_{i \in I_j} x_i$ is summable for $j \in J$. In that case, $\sum_{i \in I} x_i = \sum_{j \in J} (\sum_{i \in I_j} x_i)$
2. *Unary Sum Axiom.* Any family $\{x_i\}_{i \in I}$ in which I is a singleton is summable and $\sum_{i \in I} x_i = x_j$ if $I = \{j\}$.

Σ-monoids form a symmetric monoidal category (with product as tensor), called Σ**Mon**. A Σ**Mon-category** $\mathbb{C}$ is a category enriched in Σ**Mon**; i.e. the homsets are enriched with an additive structure such that composition distributes over addition from left and right. Note that such categories have nonempty homsets and automatically have zero morphisms, namely $0_{XY} : X \to Y = \sum_{i \in \emptyset} f_i$ for $f_i \in \mathbb{C}(X, Y)$. This does not imply the existence of a zero object.

Definition 4. A *unique decomposition category* (UDC) $\mathbb{C}$ is a symmetric monoidal Σ**Mon-category** which satisfies the following axiom:

(A) For all $j \in I$ there are morphisms called *quasi injections:* $\iota_j : X_j \to \otimes_I X_i$, and *quasi projections:* $\rho_j : \otimes_I X_i \to X_j$, such that 1. $\rho_k \iota_j = 1_{X_j}$ if $j = k$ and $0_{X_j X_k}$ otherwise. 2. $\sum_{i \in I} \iota_i \rho_i = 1_{\otimes_I X_i}$.

Proposition 1 (Matricial Representation). *Given* $f : \otimes_J X_j \to \otimes_I Y_i$ *in a UDC with* $|I| = m$ *and* $|J| = n$, *there exists a unique family* $\{f_{ij}\}_{i \in I, j \in J} : X_j \to Y_i$ *with* $f = \sum_{i \in I, j \in J} \iota_i f_{ij} \rho_j$, *namely,* $f_{ij} = \rho_i f \iota_j$.

Thus every $f : \otimes_J X_j \to \otimes_I Y_i$ in a UDC can be represented by its components. We will use the corresponding matrices to represent morphisms; for example f above (with $|I| = m$ and $|J| = n$) is represented by an $m \times n$ matrix $[f_{ij}]$. Composition of morphisms in a UDC then corresponds to matrix multiplication.

Remark. Although any $f : \otimes_J X_j \to \otimes_I Y_i$ can be represented by the unique family $\{f_{ij}\}$ of its components, the converse is not necessarily true; that is, given a family $\{f_{ij}\}$ with *I, J* finite there may not be a morphism $f : \otimes_J X_j \to \otimes_I Y_i$ satisfying $f = \sum_{ij} \iota_i f_{ij} \rho_j$. However, in case such an f exists it will be unique.

Proposition 2 (Execution/Trace Formula). *Let* $\mathbb{C}$ *be a unique decomposition category such that for every X, Y, U and* $f : X \otimes U \to Y \otimes U$, *the sum* $f_{11} + \sum_{n=0}^{\infty} f_{12} f_{22}^n f_{21}$ *exists, where* f_{ij} *are the components of* f. *Then,* $\mathbb{C}$ *is traced and* $Tr^U_{X,Y}(f) = f_{11} + \sum_{n=0}^{\infty} f_{12} f_{22}^n f_{21}$.

Example 1.

1. Consider the category **PInj** of sets and partial injective functions. Define $X \otimes Y = X \uplus Y$ (disjoint union); note that this does not give a coproduct, indeed **PInj** does not have coproducts. The UDC structure is given as follows: define $\iota_j : X_j \to \biguplus_{i \in I} X_i$ by $\iota_j(x) = (x, j)$, and define $\rho_j : \biguplus_{i \in I} X_i \to X_j$ by $\rho_j(x, j) = x$, and $\rho_j(x, i)$ is undefined for $i \neq j$.
2. This example will provide the connection to operator algebraic models. Given a set X let $\ell_2(X)$ be the set of all complex valued functions a on X for which the (unordered) sum $\sum_{x \in X} |a(x)|^2$ is finite. $\ell_2(X)$ is a Hilbert space and its norm is given by $||a|| = (\sum_{x \in X} |a(x)|^2)^{1/2}$ and its inner product is given by $< a, b >= \sum_{x \in X} a(x)\overline{b(x)}$ for $a, b \in \ell_2(X)$.

   Barr [6] observed that there is a contravariant faithful functor $\ell_2 : \mathbf{PInj}^{op} \to \mathbf{Hilb}$ where **Hilb** is the category of Hilbert spaces with morphisms the linear contractions (**norm** ≤ 1). For a set *X*, $\ell_2(X)$ is defined as above and given $f : X \to Y$ in **PInj,** $\ell_2(f) : \ell_2(Y) \to \ell_2(X)$ is defined by: $\ell_2(f)(b)(x) = b(f(x))$, if $x \in Dom(f)$ and 0, otherwise. This gives a correspondence between partial injective functions and partial isometries on Hilbert spaces ([11,1]).

   Let $\mathbf{Hilb}_2 = \ell_2[\mathbf{Pinj}]$; i.e. its objects are of the form $\ell_2(X)$ for a set X and its morphisms $u : \ell_2(X) \to \ell_2(Y)$ are of the form $\ell_2(f)$ for some partial injective function $f : Y \to X$. $\mathbf{Hilb}_2$ is a (nonfull) subcategory of **Hilb.**

For $\ell_2(X)$ and $\ell_2(Y)$ in **Hilb$_2$**, the Hilbert space tensor product $\ell_2(X) \otimes \ell_2(Y)$ and the direct sum $\ell_2(X) \oplus \ell_2(Y)$ yield tensor products in **Hilb$_2$**. **Hilb$_2$** is a traced UDC with respect to $\oplus$, where the UDC structure is induced from that of **PInj;** for more details see [12,3].

3. All partially additive categories [18,12] are examples of traced UDCs.

4 Interpretation of Proofs

In this section we define the GoI interpretation for proofs of MELL without the neutral elements. Let $\mathbb{C}$ be a traced UDC, T an additive endofunctor and U an object of $\mathbb{C}$, such that $(\mathbb{C}, T, U)$ is a GoI Situation. We interpret proofs in the homset $\mathbb{C}(U,U)$ of endomorphisms of U. Formulas (= types) will be interpreted in the next Section 5 as certain subsets of $\mathbb{C}(U,U)$; however, this introduces some novel ideas and is not needed to read the present section.

Convention: All identity morphisms are on tensor copies of U however we adopt the convention of writing 1_Γ instead of $1_{U^{\otimes n}}$ with $|\Gamma| = n$. U^n denotes the n-fold tensor product of U by itself. The retraction pairs are fixed once and for all.

Every MELL sequent will be of the form $\vdash [\Delta], \Gamma$ where Γ is a sequence of formulas and Δ is a sequence of cut formulas that have already been made in the proof of $\vdash \Gamma$ (e.g. $A, A^\perp, B, B^\perp$). This is used to keep track of the cuts that are already made in the proof of $\vdash \Gamma$. Suppose that Γ consists of n and Δ consists of $2m$ formulas. Then a proof Π of $\vdash [\Delta], \Gamma$ is represented by a morphism $[\![\Pi]\!] \in \mathbb{C}(U^{n+2m}, U^{n+2m})$. Recall that this corresponds to a morphism from U to itself, using the retraction morphisms $U \otimes U \lhd U\,(j,k)$. However, it is much more convenient to work in $\mathbb{C}(U^{n+2m}, U^{n+2m})$ (matrices on $\mathbb{C}(U,U)$). Define the morphism $\sigma : U^{2m} \to U^{2m}$, as $\sigma = s \otimes \cdots \otimes s$ (m-copies) where s is the symmetry morphism, the 2×2 antidiagonal matrix $[a_{ij}]$, where $a_{12} = a_{21} = 1; a_{11} = a_{22} = 0$. Here σ represents the cuts in the proof of $\vdash \Gamma$, i.e. it models Δ. If Δ is empty (that is for a cut-free proof), we define $\sigma : I \to I$ to be the zero morphism 0_{II}. Note that $U^0 = I$ where I is the unit of the tensor in the category $\mathbb{C}$.

Let Π be a proof of $\vdash [\Delta], \Gamma$. We define the GoI interpretation of Π, denoted by $[\![\Pi]\!]$, by induction on the length of the proof as follows.

1. Π is an *axiom* $\vdash A, A^\perp$, then $m = 0$, $n = 2$ and $[\![\Pi]\!] = s$.
2. Π is obtained using the *cut* rule on Π' and Π'' that is

$$\dfrac{\begin{array}{cc} \Pi' & \Pi'' \\ \vdots & \vdots \\ \vdash [\Delta'], \Gamma', A & \vdash [\Delta''], A^\perp, \Gamma'' \end{array}}{\vdash [\Delta', \Delta'', A, A^\perp], \Gamma', \Gamma''}\ (cut)$$

Define $[\![\Pi]\!]$ as follows: $[\![\Pi]\!] = \tau^{-1}([\![\Pi']\!] \otimes [\![\Pi'']\!])\tau$, where τ is a permutation.

3. Π is obtained using the *exchange* rule on the formulas A_i and A_{i+1} in Γ'. That is Π is of the form

$$\dfrac{\begin{array}{c}\Pi' \\ \vdots \\ \vdash [\Delta], \Gamma'\end{array}}{\vdash [\Delta], \Gamma} \ (exchange)$$

where in Γ' we have A_i, A_{i+1}. Then, $[\![\Pi]\!]$ is obtained from $[\![\Pi']\!]$ by interchanging the rows i and $i+1$. So suppose that $\Gamma' = \Gamma'_1, A_i, A_{i+1}, \Gamma'_2$, then $\Gamma = \Gamma'_1, A_{i+1}, A_i, \Gamma'_2$ and $[\![\Pi]\!] = \tau^{-1}[\![\Pi']\!]\tau$, where $\tau = 1_{\Gamma'_1} \otimes s \otimes 1_{\Gamma'_2 \otimes \Delta}$.

4. Π is obtained using an application of the *par* rule, that is Π is of the form:

$$\dfrac{\begin{array}{c}\Pi' \\ \vdots \\ \vdash [\Delta], \Gamma', A, B\end{array}}{\vdash [\Delta], \Gamma', A \parr B} \ (\parr)$$

Then $[\![\Pi]\!] = g[\![\Pi']\!]f$, where $f = 1_{\Gamma'} \otimes k \otimes 1_\Delta$ and $g = 1_{\Gamma'} \otimes j \otimes 1_\Delta$, recall that $U \otimes U \lhd U\,(j, k)$.

5. Π is obtained using an application of the *times* rule, that is Π has the form

$$\dfrac{\begin{array}{cc}\Pi' & \Pi'' \\ \vdots & \vdots \\ \vdash [\Delta'], \Gamma', A & \vdash [\Delta''], \Gamma'', B\end{array}}{\vdash [\Delta', \Delta''], \Gamma', \Gamma'', A \otimes B} \ (times)$$

Then $[\![\Pi]\!] = g\tau^{-1}([\![\Pi']\!] \otimes [\![\Pi'']\!])\tau f$, where τ is a permutation and $f = 1_{\Gamma' \otimes \Gamma''} \otimes k \otimes 1_{\Delta' \otimes \Delta''}$ and $g = 1_{\Gamma' \otimes \Gamma''} \otimes j \otimes 1_{\Delta' \otimes \Delta''}$.

6. Π is obtained from Π' by an *of course* rule, that is Π has the form :

$$\dfrac{\begin{array}{c}\Pi' \\ \vdots \\ \vdash [\Delta], ?\Gamma', A\end{array}}{\vdash [\Delta], ?\Gamma', !A} \ (ofcourse)$$

Then $[\![\Pi]\!] = ((ue_U)^{\otimes n} \otimes u \otimes u^{\otimes 2m})\varphi^{-1}T((v^{\otimes n} \otimes 1_A \otimes 1_\Delta)[\![\Pi']\!](u^{\otimes n} \otimes 1_A \otimes 1_\Delta))\varphi((e'_U v)^{\otimes n} \otimes v \otimes v^{\otimes 2m})$, where $TT \lhd T\,(e, e')$, $|\Gamma'| = n, |\Delta| = 2m$, and $\varphi : (T^2U)^{\otimes n} \otimes TU \otimes (TU)^{\otimes 2m} \to T((TU)^{\otimes n} \otimes U \otimes U^{\otimes 2m})$ is the canonical isomorphism.

7. Π is obtained from Π' by the *dereliction* rule, that is Π is of the form :

$$\dfrac{\begin{array}{c}\Pi' \\ \vdots \\ \vdash [\Delta], \Gamma', A\end{array}}{\vdash [\Delta], \Gamma', ?A} \ (dereliction)$$

Then $[\![\Pi]\!] = (1_{\Gamma'} \otimes ud_U \otimes 1_\Delta)[\![\Pi']\!](1_{\Gamma'} \otimes d'_U v \otimes 1_\Delta)$ where $Id \lhd T(d, d')$.

8. Π is obtained from Π' by the *weakening* rule, that is Π is of the form:

$$\dfrac{\begin{array}{c}\Pi' \\ \vdots \\ \vdash [\Delta], \Gamma'\end{array}}{\vdash [\Delta], \Gamma', ?A} \ (weakening)$$

Then $[\![\Pi]\!] = (1_{\Gamma'} \otimes uw_U \otimes 1_\Delta)[\![\Pi']\!](1_{\Gamma'} \otimes w'_U v \otimes 1_\Delta)$, where $\mathcal{K}_I \lhd T(w, w')$.

9. Π is obtained from Π' by the *contraction* rule, that is Π is of the form :

$$\dfrac{\begin{array}{c}\Pi' \\ \vdots \\ \vdash [\Delta], \Gamma', ?A, ?A\end{array}}{\vdash [\Delta], \Gamma', ?A} \ (contraction)$$

Then $[\![\Pi]\!] = (1_{\Gamma'} \otimes uc_U(v \otimes v) \otimes 1_\Delta)[\![\Pi']\!](1_{\Gamma'} \otimes (u \otimes u)c'_U v \otimes 1_\Delta)$, where $T \otimes T \lhd T(c, c')$.

Example 2. Let Π be the following proof:

$$\dfrac{\vdash A^\perp, A \quad \vdash A^\perp, A}{\vdash [A, A^\perp], A^\perp, A} \ (cut)$$

Then the GoI semantics of this proof is given by

$$[\![\Pi]\!] = \begin{bmatrix} 1&0&0&0 \\ 0&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \end{bmatrix} \begin{bmatrix} 0&1&0&0 \\ 1&0&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{bmatrix} \begin{bmatrix} 1&0&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 0&1&0&0 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & Id_2 \\ Id_2 & \mathbf{0} \end{bmatrix}$$

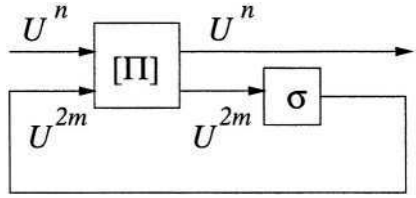
where Id_2 is the 2×2 identity matrix and $\mathbf{0}$ is the 2×2 zero matrix.

4.1 Dynamics

Dynamics is at the heart of the GoI interpretation as compared to denotational semantics and it is hidden in the cut-elimination process. The mathematical model of cut-elimination is given by the *execution formula* defined as follows:

$$EX([\![\Pi]\!], \sigma) = Tr^{U^{2m}}_{U^n, U^n}((1_{U^n} \otimes \sigma)[\![\Pi]\!]) \tag{1}$$

where Π is a proof of the sequent $\vdash [\Delta], \Gamma$. Pictorially this can be represented as follows:



Note that $EX(\llbracket \Pi \rrbracket, \sigma)$ is a morphism from $U^n \to U^n$ and it always makes sense since the trace of any morphism in $\mathbb{C}(U^{2m+n}, U^{2m+n})$ is defined. Since we are working with a traced UDC with the standard trace, by Proposition 2 we can rewrite the execution formula (1) in a more familiar form:

$$EX(\llbracket \Pi \rrbracket, \sigma) = \pi_{11} + \sum_{n \geq 0} \pi_{12}(\sigma\pi_{22})^n(\sigma\pi_{21})$$

where $\llbracket \Pi \rrbracket = \begin{bmatrix} \pi_{11} & \pi_{12} \\ \pi_{21} & \pi_{22} \end{bmatrix}$. Note that the execution formula defined in this categorical framework *always* makes sense, that is we do not need a convergence criterion (e.g. nilpotency or weak nilpotency). This is in contrast to Girard's case where the infinite sum must be made to make sense and this is achieved via proving a nilpotency result.

We later show that formula (1) is the same as Girard's execution formula. The intention here is to prove that the result of this formula is what corresponds to the cut-free proof obtained from Π using Gentzen's cut-elimination procedure. We will also show that for any proof Π of MELL the execution formula is a finite sum, which corresponds to termination of computation as opposed to divergence.

Example 3. Consider the proof Π in Example 2 above. Recall also that $\sigma = s$ in this case $(m = 1)$. Then

$$EX(\llbracket \Pi \rrbracket, \sigma) = Tr\left(\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{bmatrix}\begin{bmatrix} 0&0&1&0 \\ 0&0&0&1 \\ 1&0&0&0 \\ 0&1&0&0 \end{bmatrix}\right)$$
$$= \begin{bmatrix} 0&0 \\ 0&0 \end{bmatrix} + \sum_{n \geq 0} \begin{bmatrix} 1&0 \\ 0&1 \end{bmatrix}\begin{bmatrix} 0&0 \\ 0&0 \end{bmatrix}^n \begin{bmatrix} 0&1 \\ 1&0 \end{bmatrix} = \begin{bmatrix} 0&1 \\ 1&0 \end{bmatrix}.$$

Note that in this case we have obtained the GoI interpretation of the cut-free proof obtained by applying Gentzen's Hauptsatz to the proof Π.

5 Soundness of the Interpretation

In this section we shall prove the main result of this paper: the soundness of the GoI interpretation. In other words we have to show that if a proof Π is reduced (via cut-elimination) to its cut-free form Π', then $EX(\llbracket \Pi \rrbracket, \sigma)$ is a

finite sum and $EX(\llbracket \Pi \rrbracket, \sigma) = \llbracket \Pi' \rrbracket$. Intuitively this says that if one thinks of cut-elimination as computation then $\llbracket \Pi \rrbracket$ can be thought of as an algorithm. The computation takes place as follows: if we run $EX(\llbracket \Pi \rrbracket, \sigma)$, it terminates after finitely many steps (cf. finite sum) and yields a datum (cf. cut-free proof). This intuition will be made precise in this section through the definition of type and the main theorems (see Theorems 1,2).

Lemma 1 (Associativity of cut). *Let Π be a proof of $\vdash [\Gamma, \Delta], \Lambda$ and σ and τ be the morphisms representing the cut-formulas in Γ and Δ respectively. Then*

$$EX(\llbracket \Pi \rrbracket, \sigma \otimes \tau) = EX(EX(\llbracket \Pi \rrbracket, \tau), \sigma)$$

Proof. Follows from naturality and vanishing II properties of trace.

We proceed to defining types. This and similar definitions are directly inspired by the corresponding ones in [10], generalising them to our categorical framework.

Definition 5. Let f, g be morphisms in $\mathbb{C}(U, U)$. We say that f is *nilpotent* if $f^k = 0$ for some $k \geq 1$. We say that f is *orthogonal* to g, denoted $f \perp g$ if gf is nilpotent. Orthogonality is a symmetric relation and it makes sense because 0_{UU} exists. Also, $0 \perp f$ for all $f \in \mathbb{C}(U, U)$.

Given a subset X of $\mathbb{C}(U, U)$, we define

$$X^{\perp} = \{f \in \mathbb{C}(U, U) | \forall g (g \in X \Rightarrow f \perp g)\}$$

A *type* is any subset X of $\mathbb{C}(U, U)$ such that $X = X^{\perp\perp}$. Note that types are inhabited, since 0_{UU} belongs to every type.

Definition 6. Consider a GoI situation $(\mathbb{C}, T, U)$ as above with j_1, j_2, k_1, k_2 components of j and k respectively. Let A be an MELL formula. We define *the GoI interpretation of A,* denoted θA, inductively as follows:

1. If $A \equiv \alpha$ that is A is an atom, then $\theta A = X$ an arbitrary type.
2. If $A \equiv \alpha^{\perp}$, $\theta A = X^{\perp}$, where $\theta \alpha = X$ is given by assumption.
3. If $A \equiv B \otimes C$, $\theta A = Y^{\perp\perp}$, where $Y = \{j_1 a k_1 + j_2 b k_2 | a \in \theta B, b \in \theta C\}$.
4. If $A \equiv B \parr C$, $\theta A = Y^{\perp}$, where $Y = \{j_1 a k_1 + j_2 b k_2 | a \in (\theta B)^{\perp}, b \in (\theta C)^{\perp}\}$.
5. If $A \equiv !B$, $\theta A = Y^{\perp\perp}$, where $Y = \{uT(a)v | a \in \theta B\}$.
6. If $A \equiv ?B$, $\theta A = Y^{\perp}$, where $Y = \{uT(a)v | a \in (\theta B)^{\perp}\}$.

It is an easy consequence of the definition that $(\theta A)^{\perp} = \theta A^{\perp}$ for any formula A.

Definition 7. Let $\Gamma = A_1, \cdots, A_n$. A *datum* of type $\theta\Gamma$ is a morphism $M : U^n \to U^n$ such that for any $\beta_1 \in \theta(A_1^{\perp}), \cdots, \beta_n \in \theta(A_n^{\perp})$, $(\beta_1 \otimes \cdots \otimes \beta_n)M$ is nilpotent. An *algorithm* of type $\theta\Gamma$ is a morphism $M : U^{n+2m} \to U^{n+2m}$ for some integer m such that for $\sigma : U^{2m} \to U^{2m}$ defined in the usual way, $EX(M, \sigma) = Tr((1 \otimes \sigma)M)$ is a finite sum and a datum of type $\theta\Gamma$.

Lemma 2. *Let* $M : U^n \to U^n$ *and* $a : U \to U$. *Define* $CUT(a, M) = (a \otimes 1_{U^{n-1}})M : U^n \to U^n$. *Note that the matrix representation of* $CUT(a, M)$ *is the matrix obtained from* M *by multiplying its first row by* a. *Then* $M = [m_{ij}]$ *is a datum of type* $\theta(A, \Gamma)$ *iff for any* $a \in \theta A^{\perp}$, am_{11} *is nilpotent and the morphism* $ex(CUT(a, M)) = Tr^A(s_{\Gamma,A}^{-1} CUT(a, M) s_{\Gamma,A})$ *is in* $\theta(\Gamma)$. *Here* $s_{\Gamma,A}$ *is the symmetry morphism from* $\Gamma \otimes A$ *to* $A \otimes \Gamma$.

Theorem 1. *Let* Γ *be a sequent, and* Π *be a proof of* Γ. *Then* $[\![\Pi]\!]$ *is an algorithm of type* $\theta\Gamma$.

Theorem 2. *Let* Π *be a proof of a sequent* $\vdash [\Delta], \Gamma$ *in MELL. Then*

(i) $EX([\![\Pi]\!], \sigma)$ *is a finite sum.*

(ii) If Π *reduces to* Π' *by any sequence of cut-eliminations and "?" does not occur in* Γ, *then* $EX([\![\Pi]\!], \sigma) = EX([\![\Pi']\!], \tau)$. *So* $EX([\![\Pi]\!], \sigma)$ *is an invariant of reduction. In particular if* Π' *is any cut-free proof obtained from* Π *by cut-elimination, then* $EX([\![\Pi]\!], \sigma) = [\![\Pi']\!]$.

6 Girard's Operator Algebraic Model

In this section we observe that Girard's original C^*-algebra model (implementation) in GoI1 is captured in our categorical framework using the category $\mathbf{Hilb}_2$. First, recall [3] that $(\mathbf{PInj}, \mathbb{N} \times -, \mathbb{N})$ is a GoI situation.

Proposition 3. $(\mathbf{Hilb}_2, \ell^2 \otimes -, \ell^2)$ *is a GoI Situation which agrees with Girard's* C^**-algebraic model, where* $\ell^2 = \ell_2(\mathbb{N})$. *Its structure is induced via* ℓ_2 *from* **PInj**.

We next show that Girard's execution formula agrees with ours. Note that in Girard's execution formula $[\![\Pi]\!]$ and σ are both $n + 2m$ by $n + 2m$ matrices. Also below $\tilde{\sigma} = s \otimes \cdots \otimes s$ (m-times.)

Proposition 4. *Let* Π *be a proof of* $\vdash [\Delta], \Gamma$. *Then in Girard's model above,*

$$(1 - \sigma^2) \sum_{n=0}^{\infty} [\![\Pi]\!] (\sigma [\![\Pi]\!])^n (1 - \sigma^2) = Tr((1 \otimes \tilde{\sigma}) [\![\Pi]\!])$$

7 Conclusions and Further Work

In this paper we have given a categorical model for the GoI semantics of MELL and have proven the necessary theorems. We also showed that Girard's original operator algebra model fits this framework. We did not discuss the work by Abramsky and Jagadeesan [4] for the simple reason that it does not fit the unique decomposition category framework; that is, the category of domains does not form a UDC. This already suggests the necessity for a suitable generalization of the ideas presented in this paper. More precisely, we observe that the necessary

ingredients for a categorical interpretation (model) are provided in the definition of a GoI Situation. However one still needs to give general meaning to the notions of *orthogonality* and *type* as well as provide a notion of "nilpotency", "finite sum" or "convergence". Observe that these notions found natural meanings in UDCs but a general traced category does not always have corresponding notions.

We should note that there are many concrete GoI situations based on partially additive categories; thus there are many models of this paper ([13]). However, to obtain *exactly* Girard's GoI 1, we also used Barr's ℓ_2 representation of **PInj** in **Hilb.** We do not yet know of any operator-algebra representations for other models. That is an interesting open problem.

In [9], Girard addresses the issue of non-terminating algorithms and proves a convergence theorem for the execution formula (note that in this case nilpotency is out of the question). It would be interesting to see how this can be captured in our categorical framework where all existing infinite sums make sense. The challenge would be to have a means of distinguishing good and bad infinite sums, that is the ones corresponding to non-termination and to divergence.

Moreover in [11], Girard extended GoI to the full case, including the additives and constants. He also proved a nilpotency theorem for this semantics and its soundness (for a slightly modified sequent calculus) in the case of exponential-free conclusions. This too constitutes one of the main parts of our future work.

Last but certainly not least, we believe that GoI could be further used in its capacity as a new kind of semantics to analyze PCF and other fragments of functional and imperative languages and be compared to usual denotational and operational semantics through full abstraction theorems. The work on full completeness theorems for MLL via GoI in [12] is just a first step. Further related results, including those of Abramsky and Lenisa (e.g. [5]), should be examined.

References

1. Abramsky, S. (1996), Retracing Some Paths in Process Algebra. In *CONCUR 96, Springer LNCS* **1119**, 1-17.
2. Abramsky, S. (1997), *Interaction, Combinators and Complexity.* Lecture Notes, Siena, Italy.
3. Abramsky, S., Haghverdi, E. and Scott, P.J. (2002), Geometry of Interaction and Linear Combinatory Algebras. *MSCS,* vol. 12(5), 2002, 625-665, CUP.
4. Abramsky, S. and Jagadeesan, R. (1994), New Foundations for the Geometry of Interaction. *Information and Computation* **111** (1), 53-119.
5. Abramsky, S. and Lenisa, M. (2000), A Fully-complete PER Model for ML Polymorphic Types, *CSL'2000 Springer LNCS* **1862**, 140-155.
6. Barr, M. (1992), Algebraically Compact Functors. *JPAA,* vol. **82**, 211-231.
7. Danos, V. (1990), *La logique linéaire appliquée à l'étude de divers processus de normalisation et principalement du* λ*-calcul.* PhD thesis, Université Paris VII.
8. Danos, V. and Regnier, L. (1995), Proof-nets and the Hilbert Space. In: *Advances in Linear Logic,* London Math. Soc. Notes, **222**, CUP, 307–328.
9. Girard, J.-Y. (1988), Geometry of Interaction II: Deadlock-free Algorithms. In *Proc. of COLOG'88, LNCS* **417**, Springer, 76–93.

10. Girard, J.-Y. (1989a) Geometry of Interaction I: Interpretation of System F. In *Proc. Logic Colloquium 88,* North Holland, 221–260.
11. Girard, J.-Y. (1995), Geometry of Interaction III: Accommodating the Additives. In: *Advances in Linear Logic,* LNS **222**,CUP, 329–389,
12. Haghverdi, E. *A Categorical Approach to Linear Logic, Geometry of Proofs and Full Completeness,* PhD Thesis, University of Ottawa, Canada 2000.
13. Haghverdi, E. Unique Decomposition Categories, Geometry of Interaction and combinatory logic, *Math. Struct. in Comp. Science,* vol. **10**, 2000, 205-231.
14. Hasegawa, M. (1997), Recursion from Cyclic Sharing : Traced Monoidal Categories and Models of Cyclic Lambda Calculus, *Springer LNCS* **1210**, 196-213.
15. Hildebrandt, T. , Panangaden, P., Winskel, G, A Relational Model of Nondeterministic Dataflow, to appear in *Math. Struct. in Comp. Science,* 2004.
16. Hines, P. A categorical framework for finite state machines *Math. Struct. in Comp. Science,* vol. **13**, 2003, 451-480.
17. Joyal, A., Street, R. and Verity, D. (1996), Traced Monoidal Categories. *Math. Proc. Camb. Phil. Soc.* **119**, 447-468.
18. Manes, E.G. and Arbib, M.A. (1986), *Algebraic Approaches to Program Semantics.* Springer-Verlag.
19. Selinger, P. (1999), Categorical Structure of Asynchrony. *Electronic Notes in Theoretical Computer Science,* **20**. Elsevier Science B.V.
20. Stefanescu, G. *Network Algebra,* Springer-Verlag, 2000.

Testing Monotonicity over Graph Products

Shirley Halevy and Eyal Kushilevitz

Department of Computer Science, Technion, Haifa 3200, Israel.
{shirleyh, eyalk}@cs.technion.ac.il

Abstract. We consider the problem of monotonicity testing over graph products. Monotonicity testing is one of the central problems studied in the field of property testing. We present a testing approach that enables us to use known monotonicity testers for given graphs G_1, G_2, to test monotonicity over their product $G_1 \times G_2$. Such approach has been previously used in the special case of monotonicity testing over $[n]^d$ for a limited type of testers; however, we show that it can be applied to allow modular design of testers in many interesting cases: this approach works whenever the functions are boolean, and also in certain cases for functions with general range. We demonstrate the usefulness of our results by showing how a careful use of this approach improves the query complexity of known testers. Specifically, based on our results, we provide a new analysis for the known tester for $[n]^d$ which significantly improves its query complexity analysis in the low-dimensional case. For example, when $d = O(1)$, we reduce the best known query complexity from $O(\log^2 n/\epsilon)$ to $O(\log n/\epsilon)$.

1 Introduction

The classical notion of *decision problems* requires an algorithm to distinguish objects having some property $\mathcal{P}$ from those objects which do not have the property. *Property testing* is a relaxation of decision problems, where algorithms are only required to distinguish objects having the property $\mathcal{P}$ from those which are at least "ϵ-far" from every such object. The main goal of property testing is to avoid "reading" the whole object (which requires complexity at least linear in the size of its representation); i.e., to make the decision by reading a small (possibly, selected at random) fraction of the input (e.g., a fraction of size polynomial in $1/\epsilon$ and poly-logarithmic in the size of the representation) and still having a good (say, at least 2/3) probability of success. The notion of property testing was introduced by Rubinfeld and Sudan [24] and since then attracted a considerable amount of attention. Property testing algorithms (or *testers*) were introduced for problems in graph theory (e.g. [1,15,17,22]), monotonicity testing (e.g. [3,6,7,8,11,12,14]) and other properties (e.g. [2,4,10,19]; the reader is referred to surveys by Ron [23], Goldreich [13], and Fischer [9] for a presentation of some of this work, including some connections between property testing and other areas).

In this paper we focus on testing monotonicity of functions defined over graph products. *Monotonicity* has been one of the central problems studied in the field

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 721–732, 2004.

of property testing, e.g., [3,6,7,8,11,12,14]. A function $f : V \to (A, <_A)$ defined over some directed graph $G = (V, E)$ is *monotone* if for every $u, v \in V$ whenever $u \leq_G v$ (i.e., there is a directed path in G from u to v) then $f(u) \leq_A f(v)$. Monotonicity of general functions is a basic property and, as such, attracted much attention: efficient testers were presented for certain classes of graphs (e.g. [3,6,7,11, 12,14]), and hardness results show that monotonicity testing cannot be done for all graphs using poly-logarithmic number of queries (even for boolean functions) [11]. One family of graphs for which efficient monotonicity testers were presented is the d-dimensional hypercube, that is $[n]^d$. A partial order is defined on the domain in the natural way (for $\boldsymbol{y}, \boldsymbol{z} \in [n]^d$, we say that $\boldsymbol{y} \leq \boldsymbol{z}$ if each coordinate of $\boldsymbol{y}$ is bounded by the corresponding coordinate of $\boldsymbol{z}$). A function f over the domain $[n]^d$ is *monotone* if whenever $\boldsymbol{z} \geq \boldsymbol{y}$ then $f(\boldsymbol{z}) \geq f(\boldsymbol{y})$. Testing algorithms were developed to deal with both the low-dimensional and the high-dimensional cases. In what follows, we survey some known results relevant to our work.

In the low dimensional case, d is considered to be small compared to n (and, in fact, it is typically a constant); a successful algorithm for this case is typically one that is polynomial in $1/\epsilon$ and in $\log n$. The first paper to deal with this case is by Ergün et al. [7] which presented an $O(\frac{\log n}{\epsilon})$ algorithm for the line (i.e., the case $d = 1$), and showed that this query complexity cannot be achieved without using membership queries (that is, by querying the value of the function only at randomly drawn points of the domain); this algorithm was generalized for any fixed d in [3], with $O(\frac{(2 \log n)^d}{\epsilon})$ query complexity. For the case $d = 1$, there is a lower bound showing that monotonicity testing (for some constant ϵ) indeed requires $\Omega(\log n)$ queries [7,8]. In the high dimensional case, d is considered as the main parameter (and n might be as small as 2); a successful algorithm is typically one that is polynomial in $1/\epsilon$ and d. This case was first considered by Goldreich et al. [14], that showed an algorithm for testing monotonicity of boolean functions over the boolean $(n = 2)$ d-dimensional hyper-cube using $O(\frac{d}{\epsilon})$ queries. This was generalized in [6] to arbitrary values of n and general range; their analysis shows that $O(\frac{d \cdot \log^2 n}{\epsilon})$ queries suffice, which is the best known result so far. Lower bounds for monotonicity testing of functions $f : \{0,1\}^d \to \{0,1\}$ were shown in [11]: $\Omega(\sqrt{d})$ lower bound for non-adaptive, one-sided error algorithms, and an $\Omega(\log \log d)$ lower bound for two-sided error algorithms. Also, [11] considered graphs other than the hyper-cube, proving, for example, that testing monotonicity with a constant (depending on $\frac{1}{\epsilon}$ only) number of queries is possible for certain classes of graphs.

The focus of this paper is monotonicity testing of functions defined over graph products. A product of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a graph denoted by $G_1 \times G_2$ with vertex set $V_1 \times V_2$ and edge set E that consists of edges of the form $((u_1, v), (u_2, v))$ for $(u_1, u_2) \in E_1$ or $((u, v_1), (u, v_2))$ for $(v_1, v_2) \in E_2$. A very interesting question that arises when dealing with functions defined over graph products, is whether it is possible to use monotonicity testers known for the original graphs G_1 and G_2, to construct monotonicity testers for their products. This, if possible, will enable modular design of monotonicity testers for such graphs.

As mentioned, most previous work in the area of monotonicity testing, focused on functions defined over the d-dimensional hypercube, $[n]^d$. One feature of $[n]^d$ is that it can be viewed as a product of two lower dimensional hypercubes, or as the d^{th} power of the line graph $[n]$. Indeed, in [6, Lemma 6], dimension reduction is used for the case of boolean functions; they show a connection between the distance of a given boolean function over $[n]^d$ from being monotone to the average distance from being monotone of the one dimensional functions it induces. Based on this connection, they use previous results regarding monotonicity testing over the line to get a tester for $[n]^d$. Their approach applies only to specific kind of testers (see Section 4.2); nevertheless, it gives a first indication that graph products may be a useful tool for constructing monotonicity testers.

Our results: We study the use of graph products in monotonicity testing from several aspects. Our results go in two main directions: First, we show that the approach of testing monotonicity of functions over graph products using testers for the original graphs, can be used in a variety of cases, and not just for $[n]^d$ or for specific types of tests. In addition, we further study the dimension reduction for $[n]^d$ and show that a more careful use of this approach may yield better results than what was known prior to our work.

Let G_1 and G_2 be arbitrary graphs. Denote by ϵ the distance of a function defined over the graph product $G_1 \times G_2$ from being monotone, and by ϵ_1 (respectively, ϵ_2) the average distance of functions it induces on copies of G_1 (respectively, G_2) from being monotone. Our results establish certain relations between ϵ, ϵ_1 and ϵ_2, and show that the existence of these relations allow using any known testers for functions defined over the original graphs, G_1 and G_2, as black boxes to test monotonicity of functions that are defined over their product, $G_1 \times G_2$. At first glance, it may seem as if $\epsilon \leq \epsilon_1 + \epsilon_2$ always holds. We prove that this inequality does *not* always hold even for boolean functions; however, in many cases ϵ can indeed be bounded as a linear combination of ϵ_1 and ϵ_2. Specifically, we show that these relations between ϵ, ϵ_1 and ϵ_2 hold for boolean functions defined over graph products. That is, we show that $\epsilon \leq \epsilon_1 + \epsilon_2 + \min\{\epsilon_1, \epsilon_2\}$ for every boolean function f that is defined over $G_1 \times G_2$. For general range, we prove a linear bound for ϵ in terms of ϵ_1 and ϵ_2 in restricted types of products; specifically, whenever one of the graphs in question is the line, $\epsilon \leq 4(\epsilon_1 + \epsilon_2)$ for every (possibly non-boolean) function f. An important example for such a product is the d-dimensional hypercube $[n]^d$ (see above). Indeed, in the special case of $[n]^d$, based on the relations found between ϵ, ϵ_1 and ϵ_2 and on properties of the known tester for the line [6,7], we are able to provide a new analysis of the algorithm of [6]. Our analysis improves the best known upper bound on the query complexity for the low-dimensional case from $O(\frac{d \cdot \log^2 n}{\epsilon})$ [6] to $O(\frac{d \cdot 4^d \cdot \log n}{\epsilon})$ (this is an improvement for all $d \leq \frac{\log \log n}{2}$) and yields an $O(\frac{d^2 4^d \log^2 n}{\epsilon})$ bound on the running time.

Organization: In Section 2, we formally define graph products and state a general framework for the construction of testers for such graphs. In Section 3, we

focus on monotonicity testing of boolean functions over general graph products. In Section 4, we study monotonicity testing of general (non-boolean) functions defined over graph products of a line with any other graph, and use our results to give an improved analysis for the known monotonicity tester of [6]. For lack of space, many of the proofs are omitted.

2 Preliminaries and General Approach

Let A be some linear order. For a directed graph $G = (V, E)$ and functions $f, g : V \to A$, the *distance* between f and g is $dist(f,g) \stackrel{\text{def}}{=} |\{v \in V : f(v) \neq g(v)\}|/|V|$. Denote by $\mathcal{P}_{mono}(G, A)$ the class of functions $f : V \to A$ that are monotone with respect to G (i.e., for all $(u, v) \in E$ we have $f(u) \leq f(v)$). For every function $f : V \to A$ defined over G, denote by $\epsilon(f)$ the distance of f from monotone; i.e., $\epsilon(f) = \min_{g \in \mathcal{P}_{mono}(G,A)} dist(f, g)$; since G and A will be clear from the context, they do not appear in the notation $\epsilon(f)$.

Definition 1. *A* monotonicity tester *for $G = (V, E)$ and range A, is a probabilistic oracle machine M which, given a distance parameter $\epsilon > 0$ and an oracle access to an arbitrary function $f : V \to A$, satisfies the following two conditions:*
1. If f is monotone, then $\Pr\{M^f = Accept\} = 1$.
2. If f is ϵ-far from monotone, then $\Pr\{M^f = Accept\} \leq \frac{1}{3}$.

Definition 2. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The* product *of G_1 and G_2, denoted $G_1 \times G_2$, is the graph $G = (V, E)$ where:*
1. $V = V_1 \times V_2$.
2. $((v_1, u_1), (v_2, u_2)) \in E$ iff one of the following holds: (a) $(v_1, v_2) \in E_1$ and $u_1 = u_2$; or (b) $(u_1, u_2) \in E_2$ and $v_1 = v_2$.

Denote by $[n]$ the line graph; i.e., $G = (\{1, \ldots, n\}, \{(i, i+1) : 1 \leq i \leq n-1\})$. It is easy to see that the two dimensional mesh can be viewed as $[n] \times [n]$, and the d-dimensional hypercube can be viewed as the d^{th} power of $[n]$.

As mentioned, we are interested in using testers for G_1 and G_2 to construct a tester for $G_1 \times G_2$. Therefore, we are looking for connections between the distance from being monotone of a function defined over $G_1 \times G_2$, to the distance from being monotone of the functions it induces on copies of G_1 and G_2. To state our goal more formally, we introduce a few definitions.

Definition 3. *Given a graph product $G_1 \times G_2$ of graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, define for every vertex $v \in V_1$ the v-copy of G_2, denoted by $v \times G_2$, to be the subgraph of $G_1 \times G_2$ induced by $\{(v, u) : u \in V_2\}$ (observe that $v \times G_2$ is isomorphic to G_2). Given a function $f : V_1 \times V_2 \to A$, denote by f_v, for every $v \in V_1$, the function induced by f on $v \times G_2$. Similarly, for a vertex $v \in V_2$, define the graph $G_1 \times v$ and the induced function f_v.*

Definition 4. *Given a function $f : V_1 \times V_2 \to A$ defined over $G_1 \times G_2$, we say that f is* G_1-monotone *if, for every $v \in V_2$, the function f_v is monotone. In other words, all the functions induced by f on copies of G_1 are monotone. The notion of f being* G_2-monotone *is defined similarly.*

The next observation follows immediately from our definition of product.

Observation 1: A function defined over $G_1 \times G_2$ is monotone iff it is both G_1-monotone and G_2-monotone.

For every function $f : V_1 \times V_2 \to A$, defined over $G_1 \times G_2$, denote by $\epsilon_1(f)$ the expected distance of a function induced by f on a copy of G_1 from being monotone; $\epsilon_2(f)$ is defined similarly. That is, $\epsilon_1(f) = E_{v \in V_2}\epsilon(f_v)$ and $\epsilon_2(f) = E_{v \in V_1}\epsilon(f_v)$. Equivalently, $\epsilon_1(f)$ (respectively, $\epsilon_2(f)$) is the distance of the function f from the class of G_1-monotone (respectively, G_2-monotone) functions. This is because transforming f into a G_1-monotone function can be performed independently on every copy of G_1.

We are interested in bounding $\epsilon(f)$ as a function of $\epsilon_1(f)$ and $\epsilon_2(f)$. Specifically, a linear bound may be useful. Before presenting such bounds, we explain why this kind of (linear) bounds will enable us to use the monotonicity testers for G_1 and G_2 to construct a monotonicity tester for $G_1 \times G_2$. Assume that for some constant c, for every function f defined over $G_1 \times G_2$, indeed $\epsilon(f) \leq c(\epsilon_1(f) + \epsilon_2(f))$. (At first, it may seem as if always $\epsilon(f) \leq \epsilon_1(f) + \epsilon_2(f)$; however, in the next section we show that this is not the case even for boolean functions.)

We present a general testing scheme for $G_1 \times G_2$, using the testers for G_1 and G_2 as black boxes. Let T_1 be a monotonicity tester for G_1, and let $Q_1(\epsilon)$ be its query complexity. Similarly, let T_2 be a monotonicity tester for G_2, and let $Q_2(\epsilon)$ be its query complexity.

General_Tester(f, ϵ)
repeat $4c/\epsilon$ times:
  choose $i \in \{1, 2\}$.
  choose $v \in V_i$ uniformly.
  repeat twice - test, using T_i, that f_v is monotone with distance $\frac{\epsilon}{2c}$.
  if T_i rejects, **then return FAIL**.
return PASS

Theorem 1. *Let c be a constant. Assume that $\epsilon(f) \leq c(\epsilon_1(f) + \epsilon_2(f))$ for every function $f : V_1 \times V_2 \to A$ defined over $G_1 \times G_2$. Then, General_Tester(f, ϵ) is a monotonicity tester for functions defined over $G_1 \times G_2$ with query complexity $O(\frac{c}{\epsilon}(Q_1(\frac{\epsilon}{2c}) + Q_2(\frac{\epsilon}{2c})))$.*

The idea is that if f is indeed monotone then, by Observation 1, it passes the test with probability 1. On the other hand, if f is ϵ-far from monotone then, because $\epsilon(f) \leq c(\epsilon_1(f)+\epsilon_2(f))$, we deduce that the average distance of a function induced by f on either G_1 or G_2 is at least $\epsilon/2c$. This implies that at least $\epsilon/2c$ of these functions are at distance of at least $\epsilon/2c$ from monotone (notice that there are two possible extreme situations: one is that all the functions induced on copies of either G_1 or G_2 are $\frac{\epsilon}{2c}$-far from monotone, while the other is that only $\frac{\epsilon}{2c}$ of the functions are 1-far from monotone). The full detailed proof will appear in the full version of the paper.

Notice that, unlike the case in [6], our general tester assumes no knowledge of the testers for G_1 and G_2. However, it might be possible that such knowledge can be used to lower the query complexity, as shown in Section 4.

3 Testing Boolean Functions over Graph Products

In this section we deal with the case of *boolean* functions defined over graph products. We show that, given two graphs G_1 and G_2, for every boolean function f defined over $G_1 \times G_2$ it holds that $\epsilon(f) \leq \epsilon_1(f) + \epsilon_2(f) + \min\{\epsilon_1(f), \epsilon_2(f)\} \leq 2(\epsilon_1(f)+\epsilon_2(f))$. In addition, we give a counterexample that shows that the claim $\epsilon(f) \leq \epsilon_1(f) + \epsilon_2(f)$ does not hold for every boolean functions f; our counter example holds even in the simple case where the functions are defined over the two dimensional mesh (that is, $[n] \times [n]$). Specifically, we present a function f over the two dimensional mesh such that $\epsilon(f) \geq 1.1(\epsilon_1(f) + \epsilon_2(f))$.

Lemma 1. *Let $f : V_1 \times V_2 \to \{0,1\}$ be a function defined over $G_1 \times G_2$. Then, $\epsilon(f) \leq \epsilon_1(f) + \epsilon_2(f) + \min\{\epsilon_1(f), \epsilon_2(f)\} \leq 2(\epsilon_1(f) + \epsilon_2(f))$.*

The proof of Lemma 1 is based on the following lemma, that is stated without proof. This lemma shows that if a function f defined over $G_1 \times G_2$ is G_1-monotone, then the number of modifications required to transform f into a G_2-monotone function suffices also to transform it into a monotone function. Note that, in general, transforming f into a G_2-monotone function may damage its G_1-monotonicity. However, we show that such a transformation can be done while preserving the G_1-monotonicity.

Lemma 2. *Let $f : V_1 \times V_2 \to \{0,1\}$ be a G_1-monotone function. If the distance of f from being G_2-monotone is ϵ, then the distance of f from monotone is ϵ. In other words, $\epsilon(f) = \epsilon_2(f)$.*

Proof of Lemma 1: Let f be a boolean function defined over $G_1 \times G_2$. By the definition of $\epsilon_1(f)$, there exists a G_1-monotone boolean function f' defined over $G_1 \times G_2$ such that $dist(f, f') = \epsilon_1(f)$, and the distance of f' from being G_2-monotone is at most $\epsilon_1(f) + \epsilon_2(f)$. Thus, by Lemma 2, the distance of f' from being monotone is at most $\epsilon_1(f) + \epsilon_2(f)$. Implying that the distance of f from monotone is at most $\epsilon_1(f)+(\epsilon_1(f)+\epsilon_2(f))$. By symmetry, $\epsilon(f) \leq \epsilon_2(f)+(\epsilon_1(f)+\epsilon_2(f))$, implying that $\epsilon(f) \leq \epsilon_1(f) + \epsilon_2(f) + \min\{\epsilon_1(f), \epsilon_2(f)\}$ [1]. □

The reader may be tempted to conjecture that the proof unnecessarily pays $\epsilon_1(f)$ (or $\epsilon_2(f)$) twice, and in fact it is possible to show that $\epsilon(f) \leq \epsilon_1(f) + \epsilon_2(f)$. However, the next example shows that this is not the case. Consider the following boolean function f defined over $[n]^2$: $f(i,j) = 0$ iff $\frac{an+1}{(2a+b)} \leq i \leq \frac{(a+b)n}{2a+b}$ *or* $\frac{an+1}{(2a+b)} \leq j \leq \frac{(a+b)n}{2a+b}$; see Figure 1.

Clearly, $\epsilon_1(f)$ and $\epsilon_2(f)$ both equal $2 \cdot ab/(2a+b)^2$. On the other hand, we prove that $\epsilon(f) = (a^2 + 2ab)/(2a+b)^2$ for $2b \leq a < \frac{(2+\sqrt{8})}{2}b$ (details omitted). Thus, by setting $a = 2.41$, we have $\epsilon(f) \geq 1.1(\epsilon_1(f) + \epsilon_2(f))$.

[1] The lemma can also be proved using a generalization of the arguments used in [6, Lemma 5].

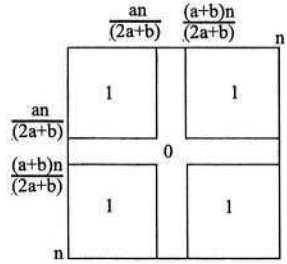


Fig. 1. The function f.

4 General Functions Defined over Products of the Line

This section deals with monotonicity testing of functions with arbitrary range, that are defined over a product of the line (that is, $[n]$) with another graph. An example for such a graph is $[n]^d$ which can be described as $[n] \times [n]^{d-1}$. Indeed, the bound presented in Lemma 3, will be used in the new analysis of the monotonicity tester for $[n]^d$. We show the following linear bound:

Lemma 3. *Let f be a function defined over $G_1 \times G_2$, where $G_1 = [n]$; then, $\epsilon(f) \leq 4\epsilon_1(f) + 3\epsilon_2(f) \leq 4(\epsilon_1(f) + \epsilon_2(f))$.*

Notice that, although this bound is not as good as the one shown in Lemma 1, it is no longer limited to a boolean range.

For simplicity, we deal with the case of $[n] \times [n]$ (Section 4.1). The generalization of our argument to a product of $[n]$ with an arbitrary graph G will appear in the full version of the paper. Then, in Section 4.2, we use Lemma 3 and specific knowledge of the monotonicity tester for the line, to improve the upper bound on the query complexity of the algorithm of [6] for general functions defined over $[n]^d$. In addition, we show that specific knowledge of the tester may be used in general to reduce the query complexity of the tester.

4.1 Monotonicity Testing for Functions Defined over $[n]^2$

In this section we prove Lemma 3 for $[n]^2$ (i.e., when $G_1 = G_2 = [n]$). In this context, we refer to G_2-monotone functions as *monotone in the first dimension* and similarly to G_1-monotone functions as *monotone in the second dimension.* Equivalently, view f as a two-dimensional array; if f is monotone in the first (second) dimension then each row (column) of the array is sorted. As before, notice that the fact that f is monotone in the first (or second) dimension does *not* imply that f is monotone; however, monotonicity is equivalent to monotonicity in both dimensions. For the proof, we need the following definition and simple lemma; proof omitted (a similar argument was used in [6]).

Definition 5. *Let $f : V \to A$ be a function defined over a graph $G = (V, E)$. A pair (u, v) is said to be an f-violation if $u <_G v$ (that is, there is a path in G from u to v) and $f(u) >_A f(v)$.*

Lemma 4. *Let $f : V \to A$ be a function defined over a graph $G = (V, E)$. Given $S \subseteq V$, if for every f-violation (i, j) either $i \in S$ or $j \in S$, then there exists a monotone function f' that differs from f only on points in S.*

Before proving the bound on the distance, we state without proof the following simple combinatorial lemma that is used in the proof.

Lemma 5. *Given $B \subseteq [n]$, define a set B' by the following process: first initialize $B' = B$; then, for every $i < j$, if at least half of the values between i and j are in B (i.e. $|\{k : k \in B \text{ and } i \leq k \leq j\}| \geq \frac{j-i+1}{2}$), set $B' = B' \cup \{k : i \leq k \leq j\}$. Then, $|B'| \leq 3 \cdot |B|$.*

The next lemma shows that if a function f is monotone in the first dimension, then if x value modifications are needed to transform f into a monotone function in the second dimension, then it is possible to transform f into a monotone function using $3x$ modifications. Lemma 3 for $[n]^2$ can then be proved in a similar way to the proof of Lemma 1, where Lemma 6 replaces Lemma 2.

Lemma 6. *Let $f : [n]^2 \to A$ be a function which is monotone in the first dimension, and ϵ-far from monotone in the second dimension (i.e., there exists a function g monotone in the second dimension s.t. $|\{(a, b) : f(a, b) \neq g(a, b)\}| \leq \epsilon \cdot n^2$). Then, the distance of f from being monotone is at most $3 \cdot \epsilon$.*

Proof. By Lemma 4, it is enough to show that there exists a set of points Y for which the following two conditions hold:
(Y_1) For every f-violation $((a_1, b_1), (a_2, b_2))$ at least one of (a_1, b_1) and (a_2, b_2) is in Y.
(Y_2) $|Y| \leq 3 \cdot \epsilon \cdot n^2$.

Since f is monotone in the first dimension, there are no f-violations of the form $((a, b_1), (a, b_2))$. Hence, there are two kinds of f-violations: pairs of the form $((a_1, b), (a_2, b))$ with $a_1 < a_2$, which we refer to as *vertical* f-violations, and pairs of the form $((a_1, b_1), (a_2, b_2))$ with $a_1 < a_2$ and $b_1 < b_2$, which we refer to as *diagonal* f-violations.

Since f is ϵ-far from monotone in the second dimension, there is a set of points $X \subseteq [n]^2$ of size at most $\epsilon \cdot n^2$ and a function g monotone in the second dimension such that, for every point (a, b), if $g(a, b) \neq f(a, b)$ then $(a, b) \in X$. Also, for every vertical f-violation $((a_1, b), (a_2, b))$, either (a_1, b) or (a_2, b) is in X. Before constructing the set Y, based on X and the set of f-violations, we need the following observation: let $((a_1, b_1), (a_2, b_2))$ be a diagonal f-violation, hence $f(a_1, b_1) >_A f(a_2, b_2)$ and let $b_1 \leq b \leq b_2$. Since f is monotone in the first dimension we have $f(a_1, b_1) \leq_A f(a_1, b)$ and $f(a_2, b) \leq_A f(a_2, b_2)$. Therefore, $f(a_2, b) <_A f(a_1, b)$, implying that $((a_1, b), (a_2, b))$ is a vertical f-violation; i.e., either (a_1, b) or (a_2, b) is in X. The construction of Y is as follows:

Step 1. Initialize $Y = X$.

Step 2. For every diagonal f-violation $((a_1, b_1), (a_2, b_2))$, consider the set $\{b_1 \leq b \leq b_2 : (a_1, b) \in X\}$. If the size of this set is more than $\frac{b_2-b_1+1}{2}$ (i.e., for more than half of the values b between b_1 and b_2 the point (a_1, b) is in X) then set $Y = Y \cup \{(a_1, b) : b_1 \leq b \leq b_2\}$. Otherwise, set $Y = Y \cup \{(a_2, b) : b_1 \leq b \leq b_2\}$.

We prove that Y satisfies the two conditions (Y_1) and (Y_2):
(Y_1): We need to show that every f-violation has (at least) one end-point in Y. As for the vertical violations – since $X \subseteq Y$ and since every vertical f-violation has at least one end-point in X, we are done. Therefore, let $((a_1, b_1), (a_2, b_2))$ be a diagonal f-violation. By step 2 in Y's construction, either (a_1, b_1) or (a_2, b_2) is in Y.
(Y_2): We need to prove that $|Y| \leq 3 \cdot |X|$. Applying Lemma 5 to each row, we conclude that the number of Y-points in each row is at most three times the number of X-points in that row. □

4.2 Testing Monotonicity over $[n]^d$

In this section we use the result shown in the previous section to provide a new analysis of the monotonicity tester presented by [6], based on dimension reduction, and by this improve the known upper bound on the query complexity of the algorithm. The main difference between the dimension reduction introduced here and the one introduced in [6] is that our reduction deals directly with functions with general range, while the approach in [6] deals with the boolean case first, and then generalizes to an arbitrary range using a general transformation presented in that work.

First, we consider the tester that follows directly from the general framework presented in Section 2. Then, using specific knowledge of the monotonicity tester for the line, we significantly improve the query complexity of the tester. By successive applications of our general scheme (presented in Section 2), one can obtain a monotonicity tester for $[n]^d$ with query complexity exponential in $\frac{1}{\epsilon}$ (which is undesirable). A different possible approach may be to successively use Lemma 3 to reduce the testing problem to the one-dimensional case, as was done in [6]; it is possible to show that if a function f defined over $[n]^d$ is ϵ-far from monotone, then the expected distance of a one-dimensional function f' induced by f (by fixing $d-1$ coordinates and allowing the remaining coordinate to range from 1 to n) from being monotone is at least $\epsilon' = \frac{\epsilon}{d \cdot 4^{d-1}}$. Based on this, a possible testing strategy is the following: randomly choose a one-dimensional function f' induced by f (i.e., randomly set $d-1$ coordinates), and test the function f' using the monotonicity tester that was presented in [7] (with f', ϵ'). It follows from the expected distance of such a one-dimensional function f' that at least $\frac{\epsilon}{d \cdot 4^{d-1}}$ of the one-dimensional functions induced by f are $\frac{\epsilon}{d \cdot 4^{d-1}}$ -far from monotone; therefore, randomly choosing $O(\frac{d \cdot 4^d}{\epsilon})$ such lines and testing monotonicity on each of them, yields a monotonicity tester for $[n]^d$.

However, the query complexity of this tester is $O(\frac{d^2 \cdot 16^d \cdot \log n}{\epsilon^2})$ (in particular it has a quadratic dependence on $\frac{1}{\epsilon}$). Our goal is to get linear dependence on $\frac{1}{\epsilon}$ (notice that for the one-dimensional case the algorithm presented in [7] already achieves this goal). As stated before, the general approach assumes no knowledge of the testers, and uses them as black boxes. However, in this case we use specific properties of the tester to get a significant improvement.

The one-dimensional monotonicity tester of [7] can be viewed as based on picking pairs of points (according to some distribution, denoted by P) and looking for a violation of monotonicity. This observation leads to the following different approach: in each phase of the algorithm pick a line, but rather than applying to it the full one-dimensional tester, choose only one pair of points on this line (according to the distribution P), It turns out that this approach enables us to use less queries; specifically, $O(\frac{d \cdot 4^d \cdot \log n}{\epsilon})$ queries suffice. We show that this can be used to reduce the query complexity in many other cases as well.

Below we focus on a special kind of testers: "edge tests" with query complexity linear in $\frac{1}{\epsilon}$. We first define the notion of a *linear edge test* more formally.

Definition 6. *A monotonicity tester T, for $G = (V, E)$ and range A, is said to be an* edge test *if T works by repeatedly picking an edge $e \in E$ according to some distribution P, and looking for a violation of monotonicity between the two endpoints of e. We say that T is* linear *if whenever f is ϵ-far from monotone, then the probability of picking an f-violation of monotonicity using P is a linear function of ϵ. We refer to this probability as the* error probability *of T and denote it by $P_T(\epsilon)$.*

A linear monotonicity edge test for $[n]$ can be found at [6] (it can also be obtained from the monotonicity tester of [7]).

We show that, for the special case of linear edge tests, lower query complexity can be achieved. Then, we show that for the special case of powers of graphs, this scheme can be further improved. The result for $[n]^d$ will be obtained as a special case of graph powers. The correctness proofs for both cases will appear in the full version of the paper.

General testing scheme for linear edge tests: Given $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, let T_1 be a linear monotonicity edge test for G_1 with error probability $P_{T_1}(\epsilon)$ and query complexity $Q_1(\epsilon)$, and let P_1 be the distribution on E_1 according to which T_1 picks the edges. Similarly, let T_2 be a linear monotonicity edge test for G_2 with error probability $P_{T_2}(\epsilon)$ and query complexity $Q_2(\epsilon)$ that picks the edges in E_2 according to P_2. The testing scheme appears in Figure 2.

Linear_Tester(f, ϵ)
repeat $2c \max\{Q_1(\epsilon), Q_2(\epsilon)\}$ times:
  choose $i \in \{1, 2\}$.
  choose $v \in V_i$ uniformly.
  choose $e \in E_i$ according to P_i.
  check both endpoints of the edge corresponding to e in f_v.
  if a violation is found **then return FAIL**.
return PASS

Fig. 2. Testing scheme for linear edge tests.

Successive application of this testing scheme yields a monotonicity tester with query complexity of $O(\frac{8^d \log n}{\epsilon})$. This query complexity is indeed linear in $\frac{1}{\epsilon}$; however, as stated above, in the case of graph powers, G^d, a better bound (as a function of d) on the query complexity can be achieved.

Linear edge tests for graph powers: Let $G = (V, E)$ be a graph. We wish to reduce the testing problem of functions defined over G^d to the problem of monotonicity testing over G. We first define the functions induced by f on copies of G.

Definition 7. *Given a function $f : V^d \to A$, define the set of one-dimensional functions induced by f. For every coordinate $i \in [d]$ and $d-1$ vertices $v_1, \ldots, v_{d-1} \in V$, the function $f^i_{v_1,\ldots,v_{d-1}} : V \to A$ is defined in the following manner: given a point $u \in V$, set $f^i_{v_1,\ldots,v_{d-1}}(u) = f(v_1, \ldots, v_{i-1}, u, v_i, \ldots, v_{d-1})$.*

Let T be a monotonicity linear edge test for G, and let P be the distribution according to which T picks the edges in E. Denote by $Q_T(\epsilon)$ the query complexity of T and by $P_T(\epsilon)$ its error probability. The tester is as follows:

Power(f, ϵ)
repeat $c^{d-1} d Q_T(\epsilon)$ times:
  choose $v_1, \ldots, v_{d-1} \in_R V$, $i \in_R [d]$
  choose $e = (u, u') \in E$ according to P.
  if $f^i_{v_1,\ldots,v_{d-1}}(u) >_A f^i_{v_1,\ldots,v_{d-1}}(u')$ **then return FAIL**
return PASS

The proof of the above tester is based on the fact that if f is indeed ϵ-far from monotone, then the functions induced by f on copies of G are not likely to be too close to monotone. This scheme for $[n]^d$ improves the upper bound on the query complexity of the monotonicity tester presented in [6] to $O(\frac{d \cdot 4^d \cdot \log n}{\epsilon})$ (this is an improvement for all $d \leq \frac{\log \log n}{2}$).

References

1. N. Alon, E. Fischer, M. Krivelevich, and M. szegedy, Efficient *testing of large graphs, FOCS* 1999, pp. 656–666.
2. N. Alon, M. Krivelevich, I. Newman, and M. Szegedy, *Regular languages are testable with a constant number of queries, SIAM Journal on Computing* 30:1842–1862, 2001 (also appeared in Proceedings of FOCS 1999, pages 645–655).
3. T. Batu, R. Rubinfeld, and P. White, Fast *approximation PCPs for multidimensional bin-packing problems, RANDOM-APPROX* 1999, 246–256,.
4. M. Blum, M. Luby, and R. Rubinfeld, *Self testing/correcting with applications to numerical problems, Journal of Computer and System Science* 47:549–595, 1993.
5. A. Bogdanov, K. Obata, and L. Trevisan, *A lower bound for testing 3-colorability in bounded-degree graphs, FOCS,* 2002, pp. 93-102.
6. Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky, *Improved testing algorithms for monotonicity, RANDOM-APPROX* 1999, pp. 97–108.

7. E. Ergün, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan, *Spot-checkers*, *Journal of Computing and System Science,* 60:717–751, 2000 (a preliminary version appeared in STOC 1998).
8. E. Fischer, *On the strength of comparisons in property testing, manuscript (available at ECCC 8(8): (2001)).*
9. E. Fischer, *The art of uninformed decisions: A primer to property testing, The Computational Complexity Column of The bulletin of the European Association for Theoretical Computer Science,* 75:97–126, 2001.
10. E. Fischer, G. Kindler, D. Ron, S. Safra, and A. Samorodnitsky, *Testing Juntas, FOCS* 2002*,* pages 103–112.
11. E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld and, A. Samorodnitsky, *Monotonicity testing over general poset domains, STOC* 2002, pp. 474–483.
12. E. Fischer and I. Newman, *Testing of matrix properties*, *STOC 2001,* pp. 286–295.
13. O. Goldreich, *Combinatorial property testing – a survey, In: Randomized Methods in Algorithms Design,* AMS-DIMACS pages 45–61, 1998 .
14. O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky, *Testing Monotonicity*, *Combinatorica,* 20(3):301–337, 2000 (a preliminary version appeared in FOCS 1998).
15. O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation, Journal of the ACM,* 45(4):653–750, 1998 (a preliminary version appeared in FOCS 1996).
16. O. Goldreich and D. Ron, *On testing expansion in bounded-degree graphs. In Electronic Colloquium on Computational Complexity* 7(20), 2000.
17. O. Goldreich and D. Ron, *Property testing in bounded degree graphs, STOC* 1997, pp. 406–415.
18. O. Goldreich and L. Trevisan, *Three theorems regarding testing graph properties, FOCS* 2001, pp. 302–317.
19. S. Halevy and E. Kushilevitz, *Distribution-free property testing. In RANDOM-APPROX* 2003, pp. 341–353.
20. T. Kaufman, M. Krivelevich, and D. Ron, *Tight bounds for testing bipartiteness in general graphs, RANDOM-APPROX* 2003, pp. 341–353.
21. Y. Kohayakawa, B. Nagle, and V. Rodl, *Efficient testing of hypergraphs, ICALP* 2002, pp. 1017–1028.
22. M. Parnas, and D. Ron, *Testing the diameter of graphs, RANDOM-APPROX* 1999, pp. 85–96.
23. D. Ron, *Property testing (a tutorial), In: Handbook of Randomized Computing* (S.Rajasekaran, P. M. Pardalos, J. H. Reif and J. D. P. Rolin eds), Kluwer Press (2001).
24. R. Rubinfeld and M. Sudan, *Robust characterization of polynomials with applications to program testing, SIAM Journal of Computing,* 25(2):252–271, 1996. (first appeared as a technical report, Cornell University, 1993).

The Minimum-Entropy Set Cover Problem

Eran Halperin[1]* and Richard M. Karp[2]

[1] CS department, Princeton University, Princeton, NJ 08544,
heran@cs.princeton.edu.
[2] International Computer Science Institute, 1947 Center St., Berkeley, CA 94704.
karp@icsi.berkeley.edu

Abstract. We consider the minimum entropy principle for learning data generated by a random source and observed with random noise.
In our setting we have a sequence of observations of objects drawn uniformly at random from a population. Each object in the population belongs to one class. We perform an observation for each object which determines that it belongs to one of a given set of classes. Given these observations, we are interested in assigning the most likely class to each of the objects.
This scenario is a very natural one that appears in many real life situations. We show that under reasonable assumptions finding the most likely assignment is equivalent to the following variant of the set cover problem. Given a universe U and a collection $\mathcal{S} = (S_1, \ldots, S_m)$ of subsets of U, we wish to find an assignment $f : U \to \mathcal{S}$ such that $u \in f(u)$ and the entropy of the distribution defined by the values $|f^{-1}(S_i)|$ is minimized.
We show that this problem is NP-hard and that the greedy algorithm for set cover finds a cover with an *additive* constant error with respect to the optimal cover. This sheds a new light on the behavior of the greedy set cover algorithm. We further enhance the greedy algorithm and show that the problem admits a polynomial time approximation scheme *(PTAS)*.
Finally, we demonstrate how this model and the greedy algorithm can be useful in real life scenarios, and in particular, in problems arising naturally in computational biology.

1 Introduction

The Shannon entropy function is a measure of the concentration of a distribution which plays an important role in various fields of computer science, such as coding theory, compression, learning, speech recognition and others. In many applications one is given a data set that has been corrupted by noise and wishes to extract the true data. In this paper we use a minimum entropy principle to attack such problems.

Data classification is an important problem in learning theory. Given a data set generated by a random source, one would like to learn the distribution of

* Some of this work was done while the author was in UC Berkeley and ICSI, Berkeley, CA. The research was partly supported by NSF ITR Grant CCR-0121555.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 733–744, 2004.

the source. Often, the data is generated by the source and then passes through a noisy channel which adds ambiguity to the data. In such cases, one would like to learn both the distribution of the source and the origin of each of the data points, thus removing the noise effects.

We consider the following scenario for noisy data generated by a random source. We are given a sequence of observations of a set of objects drawn uniformly at random from a population. Each member of the population has a type. For each object drawn from the population, we perform an observation which determines that the object's type is one of a given set of types. Given these observations, we are interested in assigning the most likely type to each of the objects.

These types might be code words in an erasure code, phonemes, letters of an alphabet, words in a limited lexicon, insurance risk categories, genomic haplotypes, alleles of a gene, different types of a disease such as leukemia, or any phenotype or trait, as long as each object has only one type. In the case of code words for example, the observation we perform on each object might be the output of an erasure channel.

We show that under some reasonable assumptions the most likely assignment is the one that minimizes the entropy of the distribution of the types. The problem of finding the most likely assignment via minimum entropy is of great practical importance. A number of approaches to this and related problems have been suggested, including the EM algorithm, Markov Chain Monte Carlo and convex optimization (see e.g. [4,8,7,10]), but we are not aware of any prior work on the computational complexity of solving the problem exactly or approximately.

The problem of finding the assignment which minimizes the entropy of the distribution of the types can be formulated as the following variant of the well-known minimum-cardinality set cover problem. We are given a universe U and a collection $\mathcal{S} = (S_1, S_2, \ldots, S_t)$ of subsets of U. A cover of U is a function $f : U \to \mathcal{S}$ such that $u \in f(u)$. The objective of the problem is to find a cover f which minimizes the entropy of the distribution $(\frac{|f^{-1}(S_1)|}{|U|}, \frac{|f^{-1}(S_2)|}{|U|}, \ldots, \frac{|f^{-1}(S_t)|}{|U|})$.

The minimum-cardinality set cover problem is well studied, and it is well known that the greedy algorithm achieves a $\ln n$ approximation [1] and that this is best possible unless $NP \subseteq ZTIME[n^{polylog(n)}]$ [6,2,5]. Although the greedy algorithm's worst-case performance for the minimum-cardinality set cover problem is far from optimal, when one looks closely at its behavior, it does not seem to give a totally unreasonable solution in the sense that most of the universe U is usually covered by relatively large sets. In fact, it has been shown that, for any t, the number of elements covered by the t largest sets in the greedy set cover is at least $1 - (\frac{t-1}{t})^t$ of the number of elements covered by the t largest sets in any set cover. In this paper we explore the greedy algorithm further, and show that it approximates the minimum entropy cover within a small *additive* constant. Thus, in this sense, the greedy algorithm actually finds a cover which explains the data nearly as well as the optimal distribution.

We further show that one can actually enhance the greedy algorithm to a polynomial time approximation scheme (PTAS) for the minimum entropy cover problem. Finally, we show how we can use the PTAS and the greedy algorithm in various scenarios arising in computational biology, and we explore the theoretical and empirical behavior of the greedy algorithm in these special cases.

2 The Minimum Entropy Cover Problem

The problem we consider in this paper is a variant of the minimum-cardinality set cover problem. We begin by formally defining the problem. In the next section we give the main motivation for the problem.

We first need some notations and definitions. Throughout the paper, all logarithms are taken to base 2. The concentration of a multiset $n_1, n_2, \ldots, n_k$ of natural numbers is defined as $\sum_{i=1}^{k} n_i \log n_i$. If $N = n_1 + \ldots + n_k$, then the entropy of $\{n_i\}$ is $\sum_{i=1}^{k} \frac{n_i}{N} \log \frac{N}{n_i}$, which is simply the entropy of the distribution $(p_1, \ldots, p_k)$ where $p_i = n_i/N$.

A set system is a universe U and a collection $\mathcal{S} = (S_1, \ldots, S_t)$ of subsets of U. A *cover* is a function $f : U \to \mathcal{S}$ such that, for all $u \in U$, $u \in f(u)$. The entropy of the cover f, denoted by $ENT(f)$, is the entropy of the sequence of numbers $\{|f^{-1}(S_i)|\}$. Similarly, the concentration of the cover f, denoted by $CON(f)$ is the concentration of $\{|f^{-1}(S_i)|\}$.

We are now ready to define the Minimum Entropy Cover Problem.

Definition 1. The Minimum Entropy Cover Problem (MIN-ENT)
INPUT: *A set system* $(U, \mathcal{S})$.
OUTPUT: *A cover* $f : U \to \mathcal{S}$.
GOAL: *Minimize* $ENT(f)$.

Informally, in the Minimum Entropy Cover Problem we are interested in finding a cover such that the distribution of the cover is as concentrated as possible. Thus, a related problem is the Maximum Concentration Problem, which is formally defined as follows.

Definition 2. The Maximum Concentration Cover Problem
INPUT: *A set system* $(U, \mathcal{S})$.
OUTPUT: *A cover* $f : U \to \mathcal{S}$.
GOAL: *Maximize* $CON(f)$.

Clearly, a cover of maximum concentration is also a cover of minimum entropy and vice versa, since there is an affine relation between the entropy and the concentration.

3 A Random Generative Model

In this section we introduce a probabilistic model for classification or identification problems with noisy data, and show that these problems can be formulated

as instances of the Maximum Concentration Problem. The setting for this model is as follows. We are given a set of *objects* drawn uniformly at random from a population. Each member of the population has a *type.* We are not told the types of the given objects, but we perform an *observation* on each object which determines that its type lies within some set of types. Given these observations we would like to find the most likely assignment of types to the objects.

Let T be the set of types and A the set of possible observations. Let $P(a|i)$ be the conditional probability of observation a, given that the object observed is of type i. Our key assumption is that for each a there is a positive real number $q(a)$ such that, for every i, $P(a|i) \in \{0, q(a)\}$. Let $COMPAT(a) = \{i \mid P(a|i) = q(a)\}$. If $i \in COMPAT(a)$ then type i is said to be *compatible* with observation a. Thus, we assume that, for all types compatible with observation a, the conditional probability of observation a is the same. We also assume that these conditional probabilities are fixed (but not necessarily known). In the important case where each type is specified by a vector of attributes and a randomly chosen subset of the attributes get observed, our assumption holds provided that the random choice of attributes to be observed is independent of the type of the object.

Suppose N objects are drawn from the population and a_j is the observation of object j. An *assignment* is a function f which assigns to each object j a type compatible with its observation. Let p_i be the (unknown) frequency of type i in the population. Then the joint probability of the observations $(a_1, a_2, \cdots, a_N)$ and the event that each object j is of type $f(j)$ is given by $\Pi_{j=1}^{N} q(a_j)p(f(j))$. We call this quantity the *joint likelihood* of the assignment of types and the observations of the objects. Note that $\Pi_{j=1}^{N} q(a_j)$ is fixed, by the assumption that the sets $COMPAT(a)$ are part of the specification of the model, and that the probabilities $q(a)$ are fixed. Thus the joint likelihood is maximized by maximizing the product $\Pi_{j=1}^{N} p(f(j))$. For each type i, let $n_i = |f^{-1}(i)|$ Then we wish to maximize $\Pi_i p_i^{n_i}$. Using simple calculus, one can verify that this quantity is maximized by choosing $p_i = \frac{n_i}{N}$. With this choice the function to be maximized becomes $\Pi_i \left(\frac{n_i}{N}\right)^{n_i}$. Taking logarithms and using the fact that the n_i sum to N, this is equivalent to maximizing the concentration $\sum_i n_i \log n_i$. Thus the problem of maximizing the joint likelihood is an instance of the Maximum Concentration Problem where, for each i, $S_i = \{j \mid i \in COMPAT(a_j)\}$.

4 The Complexity of MIN-ENT

As noted above, a maximum concentration cover is also a minimum entropy cover, and thus, if one of these problems is solvable in polynomial time then so is the other. Unfortunately, the problems are NP-hard. In fact, we prove the following stronger theorem:

Theorem 1. *Maximum concentration cover is APX-hard.*

Proof. The proof is omitted. To appear in the full version of the paper.

Note that the fact that approximating the concentration within an arbitrary constant is hard does not imply that approximating MIN-ENT within an arbitrary constant is hard! It simply implies that MIN-ENT is NP-hard. In fact, we will actually show that MIN-ENT admits a PTAS.

4.1 The Greedy Algorithm

Although it is hard to approximate the maximum concentration cover within an arbitrarily small constant factor, we shall prove a surprising property: the greedy algorithm provides an approximation with a small *additive* error.

The greedy algorithm constructs a cover f_G in the following way. We iteratively add a set $S_i \in \mathcal{S}$ which covers the maximum number of elements of U. We remove all its elements from U and from the other sets of $\mathcal{S}$, and recurse on the resulting set system. Thus, if $S_{i_1}, S_{i_2}, \ldots$, are the sets chosen by the greedy algorithm, then $f_G^{-1}(S_{i_1}) = S_{i_1}, f_G^{-1}(S_{i_2}) = S_{i_2} \setminus S_{i_1}$, and in general, $f_G^{-1}(S_{i_k}) = S_{i_k} \setminus (S_{i_1} \cup \ldots \cup S_{i_{k-1}})$.

Let $N = |U|$. We now prove the following theorem.

Theorem 2. *Let f_{OPT} be a cover of maximum concentration. Let f_G be the cover produced by the greedy algorithm. Then $ENT(f_G) \leq ENT(f_{OPT}) + 3$. Equivalently, $CON(f_G) \geq COV(f_{OPT}) - 3N$.*

Theorem 2 may not seem intuitive at first sight in view of the $\log n$ approximation factor for the performance of the greedy algorithm on the minimum-cardinality set cover problem. The theorem gives a new interpretation for the greedy algorithm: it finds a cover with an almost minimum entropy. In many real life situations, a minimum-entropy cover seems more 'natural' than a minimum-cardinality cover.

Before proving Theorem 2 we need to introduce some more notations and definitions. For two non-increasing sequences $\{n_i\}$ and $\{m_j\}$ of nonnegative real numbers, we say that $\{n_i\}$ majorizes $\{m_j\}$ if for every $k \geq 1$, their partial sums satisfy $n_1 + \ldots + n_k \geq m_1 + \ldots + m_k$. The following is a standard fact about convex functions, and it will be repeatedly used in our proof (see e.g. [3]):

Lemma 1. *Let F be a nondecreasing convex function such that $F(0) = 0$, and let $\{n_i\}$ and $\{m_j\}$ be two non-increasing sequences of nonnegative real numbers such that $\{n_i\}$ majorizes $\{m_j\}$. Then $\sum_i F(n_i) \geq \sum_i F(m_j)$, where each sum is taken over all the elements of the sequence.*

Let $S_{i_1}, S_{i_2}, \ldots$, be the sets chosen by the greedy algorithm. Furthermore, let $g_j = |f_G^{-1}(S_{i_j})|$ be the size of the j-th set covered by the greedy algorithm. By definition of the greedy algorithm, $g_1 \geq g_2 \geq \ldots$. Let $B_1, B_2, \ldots$, be the sets chosen by an optimal cover f_{OPT}, that is, for each j, there exists some i such that $B_j = f_{OPT}^{-1}(S_i)$. Finally, let $n_j = |B_j|$ and assume without loss of generality that $n_1 \geq n_2 \geq \ldots$. Theorem 2 states that $\sum g_i \log g_i \geq \sum n_i \log n_i - 3N$. In order to prove the theorem, we show that $\{g_i\}$ majorizes a certain multiset which is directly defined by $\{n_i\}$, and we then bound the concentration of that multiset.

Lemma 2. *For all i, $g_{i+1} \geq \lceil \max_k [\frac{\sum_{j=1}^{k} n_j - \sum_{j=1}^{i} g_j}{k}] \rceil$*

Proof. For every k, the number of elements covered by the largest k sets of f_{OPT} is $n_1 + \ldots + n_k$. On the other hand, the number of elements covered by the first i sets of the greedy algorithm is $g_1 + \ldots + g_i$. Therefore, before the $i+1$-th iteration of greedy, there are at least $\sum_{j=1}^{k} n_j - \sum_{j=1}^{i} g_j$ uncovered elements in $B_1 \cup \ldots \cup B_k$. By averaging, there is at least one set B_l for some $l \in \{1, \ldots, k\}$ such that the number of uncovered elements in B_l is at least $\frac{\sum_{j=1}^{k} n_j - \sum_{j=1}^{i} g_j}{k}$, and thus $g_{i+1} \geq \frac{\sum_{j=1}^{k} n_j - \sum_{j=1}^{i} g_j}{k}$.

Motivated by Lemma 2, we define a multiset $\{m_i\}$ in the following way. Let $m_1 = n_1$, and for $i \geq 2$ let

$$m_{i+1} = \lceil \max_k [\frac{\sum_{j=1}^{k} n_j - \sum_{j=1}^{i} m_j}{k}] \rceil.$$

We call this multiset the *extremal greedy multiset.*

Lemma 3. *The concentration of the greedy cover is at least the concentration of the extremal greedy multiset.*

Proof. We prove by induction on i that $\sum_{j=1}^{i} m_j \leq \sum_{j=1}^{i} g_j$. By Lemma 2, we get that $m_1 \leq g_1$. Assume for induction that $\sum_{j=1}^{i} m_j \leq \sum_{j=1}^{i} g_j$. Let k be such that $m_{i+1} = \lceil \frac{\sum_{j=1}^{k} n_j - \sum_{j=1}^{i} m_j}{k} \rceil$. Then, by Lemma 2,

$$\begin{aligned} m_{i+1} &= \lceil \frac{\sum_{j=1}^{k} n_j - \sum_{j=1}^{i} m_j}{k} \rceil \\ &\leq \lceil \frac{\sum_{j=1}^{k} n_j - \sum_{j=1}^{i} g_j}{k} \rceil + \lceil \frac{\sum_{j=1}^{i} g_j - \sum_{j=1}^{i} m_j}{k} \rceil \\ &\leq g_{i+1} + \lceil \frac{\sum_{j=1}^{i} g_j - \sum_{j=1}^{i} m_j}{k} \rceil, \end{aligned}$$

and so,

$$\begin{aligned} \sum_{j=1}^{i+1} m_j &\leq \sum_{j=1}^{i} m_j + g_{i+1} + \lceil \frac{\sum_{j=1}^{i} g_j - \sum_{j=1}^{i} m_j}{k} \rceil \\ &= \sum_{j=1}^{i+1} g_j + (\sum_{j=1}^{i} m_j - \sum_{j=1}^{i} g_j) + \lceil \frac{\sum_{j=1}^{i} g_j - \sum_{j=1}^{i} m_j}{k} \rceil \leq \sum_{j=1}^{i+1} g_j, \end{aligned}$$

where the last inequality follows from the induction hypothesis and the fact that g_j and m_j are integers. Since $\sum_{j=1}^{i} m_j \leq \sum_{j=1}^{i} g_j$, then by Lemma 1, $\sum g_j \log g_j \geq \sum m_j \log m_j$, that is, the concentration of greedy is greater than the concentration of the extremal greedy multiset.

We now describe another intermediate multiset $\{r_h\}$ whose concentration is at most that of the extremal greedy multiset. We then proceed to show that the concentration of $\{n_j\}$ exceeds that of $\{r_h\}$ by at most N. For each h, r_h will be equal to $\lceil \frac{\sum_{j=1}^{k_h} n_j - \sum_{j=1}^{h-1} r_j}{k_h} \rceil$, where the choice of the index k_h is as follows. Let $J_l = \{j \mid 2^{l-1} < \frac{N}{n_j} \leq 2^l\}$, let $W_l = \sum_{j \in J_l} n_j$ and let $t_l = \max_{j \in J_l} j$. Then, we set $k_h = \min\{t_l \mid W_1 + W_2 + \cdots W_l > r_1 + r_2 + \cdots + r_{h-1}\}$.

Lemma 4. *The concentration of the extremal greedy multiset is greater than or equal to* $\sum_h r_h \ln(r_h) - N$.

Proof. For every $l \geq 1$, let $R_l = \{h \mid k_h = t_l\}$. Let $r_{l1} \geq r_{l2}, \ldots$ be the set of $r_h \in R_l$. Then, $r_{l,i+1} = \lceil \frac{W_l - (r_{l1} + \ldots + r_{li})}{t_l} \rceil$.

We consider another intermediate multiset $\{a_i\}$ which is defined by applying the following modifications to the multiset $\{m_i\}$. We define a breakpoint at m_i if for some l, $m_1 + \ldots + m_{i-1} < W_1 + \ldots + W_l \leq m_1 + \ldots + m_i$. We replace the element m_i by a new element m_i' such that $m_1 + \ldots + m_i' = W_1 + \ldots + W_l$. We then replace m_{i+1} by $m_{i+1} + m_i - m_i'$. It is easy to see that the resulting multiset $\{a_i\}$ satisfies that $\sum m_i \log m_i \geq \sum a_i \log a_i - N \log 2 = \sum a_i \log a_i - N$ and that in every interval W_l, if $a_{l1} \geq a_{l2} \geq \ldots$ are the elements of $\{a_i\}$ in that interval, then $a_{l,i+1} \geq \lceil \frac{W_l - (a_{l1} + \ldots + a_{li})}{t_l} \rceil$.

Since for every $l \geq 1$, $\{a_{li}\}$ majorizes $\{r_{li}\}$ then by Lemma 1, $\sum a_i \log a_i \geq \sum r_i \log r_i$, and thus the lemma follows.

Since the multiset $\{r_h\}$ is explicitly given, we can lower bound its concentration by a simple calculation.

Lemma 5. *For every* $l \geq 1$, $\sum_{h \in R_l} r_h \log r_h \geq W_l \log W_l - W_l \log t_l - W_l$.

Proof.

$$\begin{aligned}
\sum_{h \in R_l} r_h \lg r_h &\geq \sum_{i=0}^{\infty} \frac{W_l}{t_l} \left(1 - \frac{1}{t_l}\right)^i \log \left(\frac{W_l}{t_l} \left(1 - \frac{1}{t_l}\right)^i\right) \\
&= W_l \log(\frac{W_l}{t_l}) + \frac{W_l}{t_l} \log \left(1 - \frac{1}{t_l}\right) \sum_{i=0}^{\infty} i \left(1 - \frac{1}{t_l}\right)^i \\
&= W_l \log(\frac{W_l}{t_l}) + W_l(t_l - 1) \log \left(1 - \frac{1}{t_l}\right) \\
&\geq W_l \log(\frac{W_l}{t_l}) - W_l.
\end{aligned}$$

The proof of the following claim is straightforward by the definition of W_l and t_l.

Claim. $(t_l - t_{l-1}) \frac{N}{2^l} < W_l \leq (t_l - t_{l-1}) \frac{N}{2^{l-1}}$.

We now upper bound the concentration of f_{OPT} in each of the intervals W_l.

Lemma 6. $\sum_{j \in J_l} n_j \log n_j \leq W_l \log(\frac{W_l}{t_l - t_{l-1}})$

Proof. For a set of $t = t_l - t_{l-1}$ numbers $a_1, \ldots, a_t$ such that $a_1 + \ldots + a_t = W_l$, it is easy to see that $\sum a_i \log a_i$ is maximized when for every i, $a_i = \frac{W_l}{t}$, and in that case, $\sum a_i \log a_i = W_l \log \frac{W_l}{t}$. Therefore, the lemma follows.

Lemmas 6 and 5 allow us to bound the difference between the concentration of $\{r_h\}$ and that of $\{n_j\}$.

Lemma 7. $\sum_j n_j \log n_j - \sum_h r_h \ln r_h \leq 2N$

Proof. By the lemmas above,

$$\begin{aligned}\sum_j n_j \log n_j - \sum_h r_h \ln r_h &\leq \sum_l W_l (\log\left(\frac{t_l}{t_l - t_{l-1}}\right) + 1) \\ &= N + \sum_l W_l \log(\frac{t_l}{t_l - t_{l-1}}) \\ &\leq N + \sum_l W_l \frac{t_{l-1}}{t_l - t_{l-1}} \leq N + N \sum_l \frac{t_{l-1}}{2^{l-1}},\end{aligned}$$

where the last inequality follows from Claim 4.1. But note that $\sum_l \frac{t_l - t_{l-1}}{2^{l-1}} \leq \sum_j \frac{n_j}{N} = 1$, and thus, $\sum_l \frac{t_l}{2^l} \leq 1$.

We can now prove Theorem 2:

Proof. By Lemmas 3 and 4, $CON(f_G) \geq \sum r_h \log r_h - N$. On the other hand, by Lemma 7, $CON(f_{OPT}) - 2N \leq \sum r_h \log r_h$. Thus, $CON(f_G) \geq CON(f_{OPT}) - 3N$.

Theorem 2 shows that the greedy algorithm comes within an additive constant of the optimal entropy. In order to implement the greedy algorithm, one has to solve the subroutine that finds a set $S \in \mathcal{S}$ which covers the maximum number of elements of U. If the collection $\mathcal{S}$ is given explicitly, then this subroutine can be done by enumerating over all possible sets. But in some scenarios, the sets are given implicitly, and then finding the set which covers the maximum number of uncovered elements may be NP-hard. If this subroutine admits an α-approximation algorithm for some $\alpha < 1$, then by tracing the proof of Theorem 2, one can verify that $CON(f_G) \geq CON(f_{OPT}) - (3 + \log(\frac{1}{\alpha}))N$. Examples where this result is applicable include covering the edges of a graph by cut-sets, covering the vertices of a graph by dominating sets, and covering a finite set of points in R^n by balls of a given radius.

Note that a constant-factor approximation for the maximum concentration problem does not immediately follow from Theorem 2, but the greedy algorithm does in fact achieve such an approximation. We omit the proof from this version of the paper.

4.2 A PTAS for MIN-ENT

The greedy algorithm finds a cover with relatively small entropy, but there is a family of instances (given in the full version of this paper) in which the ratio between the optimal entropy and the entropy of the greedy cover is bounded above by a constant smaller than one. In this section we show how can one enhance the greedy algorithm and find a polynomial time approximation scheme for MIN-ENT, that is, we show that for every constant $\epsilon > 0$ one can approximate MIN-ENT within a factor of $1 + \epsilon$.

We keep the notations from the previous section. We let $OPT = ENT(f_{OPT})$, and $a = \frac{3}{\epsilon}$. We say that f is a *large partial* cover of U, if the following three properties hold:

- The domain of f (denoted D_f) is a subset of U (that is, the cover does not have to cover all of U).
- For every $S \in \mathcal{S}$, either $f^{-1}(S)$ is empty or $|f^{-1}(S)| \geq \frac{N}{2^a}$.
- If $f^{-1}(S)$ is not empty, then $S \subseteq D_f$.

The support of a large partial cover f is $\mathcal{X}_f = \{S \in \mathcal{S} \mid f^{-1}(S) \neq \emptyset\}$. Note that if the support of f is $\mathcal{X}_f$, then $\cup_{S \in \mathcal{X}_f} S = D_f$. A cover g of U is an extension of f if for every $i \in D_f$, $g(i) = f(i)$. The algorithm is the following:

1. Apply the greedy algorithm. Let the concentration of the resulting cover be CON_0.
2. For every large partial cover f, find an extension g of f by applying the greedy algorithm to all the sets that are not entirely covered by f.
3. Output the cover with maximum concentration among CON_0 and all the covers found in step 2

We first prove that the algorithm indeed gives a $1 + \epsilon$ approximation. First note that if $OPT > \frac{3}{\epsilon}$, then by Theorem 2, the greedy algorithm finds a cover f such that $ENT(f) \leq OPT + 3 < (1 + \epsilon)OPT$. We thus assume that $OPT \leq \frac{3}{\epsilon}$.

Let $k = \max_{n_j > \frac{N}{2^a}} j$, that is k is the maximal index such that $n_j > \frac{N}{2^a}$. Let $X = \sum_{j \geq k+1} n_j$. Then,

$$N \log N - N \cdot OPT = CON(f_{OPT}) \leq X(\log N - a) + (N - X) \log N,$$

and thus, $X \leq \frac{N \cdot OPT}{a}$.

It is easy to see that if B_j is the set corresponding to n_j in the optimal solution, then the projection of the optimal cover to $B_1 \cup \ldots \cup B_k$ is a large partial cover. Therefore, in step 2 of the algorithm, one possible large partial cover is the one defined by the multiset $n_1, n_2, \ldots, n_k$. For this specific partial cover, the algorithm extends it to a cover g such that its concentration satisfies

$$CON(g) \geq \sum_{j=1}^{k} n_j \log n_j + \sum_{j \geq k+1} n_j \log n_j - 3 \sum_{j \geq k+1} n_j \geq CON(f_{OPT}) - 3\frac{N \cdot OPT}{a}.$$

Thus,

$$ENT(g) = \log N - \frac{CON(g)}{N} \leq \log N - \frac{CON(f_{OPT})}{N} + 3\frac{OPT}{a}$$

$$= OPT(1 + \tfrac{3}{a}) = OPT(1 + \epsilon).$$

Finally, it remains to show that the algorithm can be implemented in polynomial time. Clearly, the greedy algorithm can be implemented in polynomial time. Thus, it suffices to show that one can enumerate over all large partial covers in polynomial time. A simple enumeration will be too costly, since the number of large partial covers may be exponential in N. We therefore introduce a polynomial-size subset of the set of all large partial covers, and we show that it is enough to enumerate over this subset.

Let f be a large partial cover, and let $\mathcal{X}_f = \{S'_1, \ldots, S'_l\}$ be its support and D_f its domain. f is called a maximal partial cover if for every $x, y \in D_f$ such that $f(x) \neq f(y)$ there is $i \leq l$, such that $x \in S'_i, y \notin S'_i$ or $x \notin S'_i, y \in S'_i$.

It is easy to see that if f is not maximal, then f cannot be extended to a maximum concentration cover. Therefore, it is enough to enumerate over all maximal partial covers. Note that the support of a large partial cover contains at most 2^a sets. Hence, we can enumerate over all possible supports of these covers since there are at most $t^{2^a} = t^{2^{3/\epsilon}}$ such supports. Let $\mathcal{X} = \{S'_1, \ldots, S'_l\}$, where $l \leq 2^a$. We bound the number of maximal partial covers with support $\mathcal{X}$ and domain $D = S'_1 \cup \ldots \cup S'_l$. Let $\mathcal{A} = \{A_1, A_2, \ldots, A_{2^l}\}$ be the subsets of D defined by the possible intersections of sub-collections of $\mathcal{X}$. It is easy to see that by enumerating over all partitions of D by sets of $\mathcal{A}$, we enumerate over all maximal partial covers with support $\mathcal{X}$. There are at most $2^{2^l} \leq 2^{2^{2^a}}$ such partitions. We thus get the following theorem:

Theorem 3. *For every $\epsilon > 0$, there is a $(1 + \epsilon)$-approximation algorithm for MIN-ENT which runs in time $O(2^{2^{2^{3/\epsilon}}} \cdot t^{2^{3/\epsilon}} \cdot (Nt)^{O(1)})$.*

5 Applications

In this section we introduce a scenario where the random generative model is helpful.

We introduce an application which naturally arises in computational biology, but can also be viewed as a more general string-oriented problem.

A partial haplotype is a string over $\{0, 1, *\}^k$. A complete haplotype is simply a binary string of size k. A complete haplotype h is compatible with a partial haplotype h' if and only if for each i, if $h'(i) \neq *$ then $h(i) = h'(i)$.

In the haplotype resolution problem, we are given a set $U = \{h_1, h_2, \ldots, h_m\}$ of partial haplotypes of length k. For each complete haplotype $h \in \{0, 1\}^k$, let $S_h = \{h_i \in U \mid h \text{ is compatible with } h_i\}$. The set U together with its collection of subsets $\mathcal{S} = \{S_h \mid h \in \{0, 1\}^k\}$ forms a set system. We wish to find a minimum-entropy cover for this system.

The problem arises in the following biological context. A geneticist conducts an experiment, in which one of the steps is to sequence the DNA of a sample of

individuals from the population. Unfortunately, current sequencing technology often gives the DNA sequence with some missing nucleotide bases at some positions. Our goal is to complete these missing bases. In terms of the notations above, each partial haplotype $h_i \in U$ corresponds to the DNA sequence of one individual, and the * values correspond to missing bases. Clearly, the data observed by the geneticist follows the random generative model described in Section 3, where the types are the complete haplotypes, the observations are the partial haplotypes in U, and for each $h_i \in U$, $COMPAT(h_i) = \{h \in \{0,1\}^k \mid h_i \in S_h\}$. Thus, by the analysis given in Section 3, the most likely completion of the partial haplotypes is the one defined by the minimum entropy cover.

Since the haplotype resolution cover is a special case of MIN-ENT, there is hope to find a polynomial-time algorithm for it. We now show that this is not possible in the general case.

Theorem 4. *The haplotype resolution problem is APX-hard.*

Proof. The proof is omitted. To appear in the full version of the paper.

In the context of haplotype resolution, the greedy algorithm iteratively finds the complete haplotype which covers the maximum number of partial haplotypes in the data set. It then completes these partial haplotypes to that haplotype, and removes them from the data set. When $k = O(\log m)$, finding the complete haplotype can be done in polynomial time, simply by enumerating over all possible complete haplotypes. For an arbitrary k, this is NP-hard [9]. For practical data sets, the length of the DNA sequences is quite short (around 10) due to some practical considerations[1]. Therefore, for such regions, one can efficiently apply the greedy algorithm. In the full version of the paper we report some successful results over real biological data.

References

1. V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research,* 4:233–235, 1979.
2. U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM,* 45, 1998.
3. G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities.* Cambridge University Press, Cambridge, England, 1934.
4. E.H. Herskovits and G.F. Cooper. Kutato: an entropy-driven system for construction of probabilistic expert systems from database. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence,* pages 54–62, 1990.
5. C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proceedings of the 25rd Annual ACM Symposium on Theory of Computing, San Diego, California,* pages 286–293, 1993.

[1] The number of sequenced individuals is usually not very large, and a long sequence would mean that each DNA sequence appeared only once in the data set (and thus, there is no information). Another reason to use short regions is that there are strong correlations among different positions in the DNA that are physically close to each other.

6. Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, Texas,* pages 475–484, 1997.
7. S. Roberts, R. Everson, and I. Rezek. Minimum entropy data partitioning. In *Proc. of 9th International Conference on Articial Neural Networks,* pages 844–849, 1999.
8. Stephen J. Roberts, Christopher Holmes, and Dave Denison. Minimum-entropy data partitioning using reversible jump markov chain monte carlo. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 23(8):909–914, 2001.
9. R. Sharan. Personal communication. 2003.
10. Y. Xiang, S. K. Michael Wong, and Nick Cercone. A "microscopic" study of minimum entropy search in learning decomposable markov networks. *Machine Learning,* 26(1):65–92, 1997.

Communication Versus Computation

Prahladh Harsha[1]*, Yuval Ishai[2], Joe Kilian[3],
Kobbi Nissim[4]**, and S. Venkatesh[5]***

[1] Computer Science and Artificial Intelligence Laboratory, MIT,
Cambridge, MA 02139, USA, prahladh@mit.edu
[2] Computer Science Department, Technion, Haifa 32000, Israel,
yuvali@cs.technion.ac.il
[3] NEC Laboratories America Inc, Princeton, NJ 08540, USA,
joe@nec-labs.com
[4] Microsoft Research, SVC. 1065 La Avenida, Mountain View, CA 94043, USA,
kobbi@microsoft.com
[5] Computer Science Department, University of Victoria,
Victoria, BC, Canada V8W 3P6, venkat@cs.uvic.ca

Abstract. We initiate a study of tradeoffs between communication and computation in well-known communication models and in other related models. The fundamental question we investigate is the following: Is there a computational task that exhibits a strong tradeoff behavior between the amount of communication and the amount of time needed for local computation?

Under various standard assumptions, we exhibit boolean functions that show strong tradeoffs in the following computation models: (1) two-party randomized communication complexity; (2) query complexity; (3) property testing. For the model of deterministic communication complexity, we show a similar result relative to a random oracle.

Finally, we study a time-degree tradeoff problem that arises in arithmetization of boolean functions, and relate it to time-communication tradeoff questions in multi-party communication complexity and in cryptography.

1 Introduction

A Motivating Riddle. Consider the following multi-party communication game. Fix a finite field F and let M be a $n \times k$ matrix over F. The columns of F are assigned to k players so that each player j knows all columns of M *except* the jth. (This is known as the "input on the forehead" model [CFL83].) The players' goal is to compute the product of the n row sums, namely the function

$$\mathsf{PS}(M) = \prod_{i=1}^{n} \sum_{j=1}^{k} M_{i,j},$$

* Research done while the author was at NEC Laboratories America.
** Research done while the author was at NEC Laboratories America.
*** Research done while the author was at MPI for Informatik, Germany.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 745–756, 2004.

by means of simultaneously sending messages to an external referee. This can be easily done by having the entire matrix M sent to the referee (e.g., letting P_1 send the second column and P_2 the remaining columns). The goal is to minimize the *communication complexity,* measured as the length of the longest message sent. A closely related problem was studied in [BGKL03]. When $k > n$ (say, $k = n+1$) our problem admits the following simple solution, implicit in [BGKL03]. Write $\mathsf{PS}(M)$ as the sum of k^n terms, where each term is a product involving a single entry from each row of M. Since there are more players than rows, for each such term there is a player holding all of its values. Hence, one can assign each term to some player who knows its value, and have each player send the sum of all terms assigned to it. The referee can then recover $\mathsf{PS}(M)$ by simply adding up the k field elements it received. While this protocol is very efficient in communication, the combined *computation* of the players is exponential in n. Note that if one uses the natural greedy strategy of assigning each term to the *first* player to which it can be assigned, then player $n+1$ will need to compute the *permanent* of an $n \times n$ sub-matrix of M, a #*P*-hard problem.[1] Thus, a natural question is the following:

> Does the function $\mathsf{PS}(M)$ admit a protocol in which (1) each player only sends a single element of F; and (2) the local computation of each player is polynomial in n?

A negative answer seems likely in light of the failure of the natural term assignment strategy. It also seems reasonable that for *any* valid way of assigning the k^n terms to the players, some player will be forced to compute a hard function. Thus, this problem looks like a good candidate for a *time-communication tradeoff*: it requires little time to compute when there is no limit on the communication complexity, requires little communication when there is no limit on the time complexity, but seems to defy solutions that are simultaneously efficient with respect to both complexity measures.

Quite surprisingly, it turns out that the answer to the above question is "yes". (The impatient reader can skip to Section 5 for a solution to the riddle.) Thus, this particular problem does not exhibit the time-communication tradeoff that was initially suspected. However, this question served as the original motivation for this work, which explores the existence of similar kinds of tradeoffs in related contexts.

1.1 Problem Description

Let $f : X \times Y \to Z$ be an arbitrary function of two inputs. In the two-party communication model of Yao [Yao79], there are two players A and B. A is given $x \in X$, B is given $y \in Y$ and they need to compute $z = f(x, y)$ by communicating with each other. In any communication protocol designed for f, there are three useful measures of complexity:

[1] Even if F has characteristic 2, in which case the permanent can be efficiently computed, it is not clear that the computation of (say) the middle player can be made efficient.

- **Communication complexity:** The total number of bits exchanged between A and B;
- **Time complexity:** The amount of time needed by A and B for local computation;
- **Round complexity:** The number of messages exchanged by A and B.

Given any two of these three complexity measures, it is natural to ask if there are tasks which exhibit a tradeoff between them. The question of rounds vs. computation does not arise in the two-party model, as the simple protocol in which A send his entire input over to B is optimal with respect to both measures.[2] Tradeoffs between round complexity and communication complexity have been well studied (see below). In this paper, we initiate the study of the remaining question: proving tradeoffs between communication and local computation. Specifically, our main goal is to find functions f such that: (1) f can be efficiently computed given both its inputs, i.e., given no restriction on the communication; (2) f has a protocol with low communication complexity given no restriction on the computation; and (3) there is no protocol for f which simultaneously has low communication and efficient computation.

1.2 Related Work

Papadimitriou and Sipser [PS84] first discussed the problem of showing tradeoffs between rounds of communication and communication complexity. For any fixed k, they proposed a boolean function p_k called the *pointer chasing problem* that has a k-round protocol with $O(\log n)$ bits of communication. They conjectured that its communication complexity is at least linear if only $k-1$ rounds are allowed. In other words, p_k shows a strong tradeoff behavior between rounds and communication complexity. This conjecture was proved in a series of papers [PS84,DGS87,NW93].

Additional complexity measures which are not considered in this work are *space* complexity and *randomness* complexity. Tradeoffs between space and communication were considered by Beame et al. [BTY94]. Tradeoffs between randomness and communication were studied by Canetti and Goldreich [CG93].

1.3 Our Results

Our first result is a strong time-communication tradeoff for a boolean function in the two-party randomized communication model.

Randomized communication model. Suppose that there is a UP relation R such that the search problem corresponding to R is not in $\mathrm{BPTIME}[2^{O(T(n))}]$. (This would follow from the existence of a one-way permutation secure against

[2] However, this question does make sense in a cryptographic setting when players need to compute a function of their inputs without revealing their inputs to each other. Such a tradeoff question is addressed in Section 5.3.

a $2^{O(T(n))}$ bounded adversary.) Then, there is an efficiently computable boolean function f_R with the following properties. If Alice and Bob are computationally unbounded, then there is an $O(\log n)$-bit 1-round randomized protocol that computes f_R. But if Alice and Bob are computationally bounded, then any randomized protocol for f_R, even with multiple rounds, will require $\Omega(T(n))$ bits of communication (see Section 3).

As a corollary we get the following strong separation result. Let F_c denote the class of functions $f(x,y) \in \text{PTIME}$ such that the randomized communication complexity of f is bounded by c. Similarly, let F_c^{poly} be the functions $f(x,y) \in \text{PTIME}$ such that $f(x,y)$ is computable by polynomial-time parties with communication c. Then there is an explicit boolean function f in $F_{\log n} \setminus F_{T(n)}^{\text{poly}}$ for $T(n)$ as above.

Deterministic communication model. Obtaining similar tradeoff results for the deterministic two-party model appears to be much harder. We show a strong tradeoff result relative to a random oracle. Specifically, let L be a random sparse language. Then, with probability 1 over choice of L, there is a boolean function f_L (efficiently computable relative to L) with the following properties. There is a *deterministic* communication protocol for f_L with, say, $O(\log^2 n)$ bits of communication if both Alice and Bob are computationally unbounded with oracle access to L. However, any protocol in which Alice and Bob are computationally bounded will require $\Omega(n)$ bits of communication, even with oracle access to L. We defer the proof of this tradeoff to the full version of the paper [HIKNV].

Query complexity and property testing. Our next results prove tradeoffs in related models like the query complexity model and the property testing model. In these models, information is stored in the form of a table and the queries are answered by bit-probes to this table. We view the probes as communication between the stored table and the query scheme (or the tester), and the computation of the query scheme (or the tester) as the local computation. We show that: (a) Under a cryptographic assumption, there exists a language L such that, on inputs of length n, a query scheme with unlimited computation makes $O(\log n)$ queries while a query scheme with efficient local computation requires $\Omega(n^\varepsilon)$ queries for some fixed $\varepsilon < 1$; (b) assuming $\text{NP} \not\subseteq \text{BPP}$, given any $\varepsilon > 0$, there exists a property P such that, on inputs of length n, a computationally unbounded tester will require only n^ε bits to check if the input satisfies the property or is far from satisfying it. On the other hand, a computationally bounded tester will require $n^{1-\varepsilon}$ bits. We only provide the proof of the tradeoff in the query complexity model (see Section 4) and defer the proof of the tradeoff in the property testing model to the full version of the paper [HIKNV].

Natural tradeoff questions. In addition to proving the *existence* of tradeoffs in various contexts, we also put forward several concrete *natural* tradeoff questions and relate them to each other. We propose three different tradeoff questions arising in different contexts: arithmetization of boolean functions, multi-party

communication, and cryptography. We relate them by showing that a "positive" resolution of the first would imply a solution to the second, which in turn would imply a solution to the third. Hence, the cryptographic application may serve as an additional motivation for studying the other two. For want of space, we defer the entire discussion on these natural tradeoff questions to the full version of the paper [HIKNV].

2 Preliminaries

In this section, we describe the communication complexity model, a formal definition of the problem we consider and the notion of UP relations.

2.1 The Communication Complexity Model [Yao86]

Let X, Y and Z be arbitrary finite sets and $f : X \times Y \to Z$ be an arbitrary function. There are two players, Alice and Bob who wish to evaluate $f(x, y)$ for $x \in X$ and $y \in Y$. However, Alice only knows x and Bob only knows y. To evaluate the function, they communicate with each other according to some fixed protocol P in which they send messages to each other.

The cost of a protocol P on an input (x, y) is the number of bits exchanged by Alice and Bob when Alice is given x and Bob is given y. The cost of a protocol P is the worst case cost of P over all inputs (x, y). The (deterministic) communication complexity of f is the minimum cost of a protocol that computes f.

If Alice and Bob are allowed access to random coin tosses and their messages depend also on the result of the coin tosses besides their input and the communication so far, we say that the protocol P is randomized. The randomized communication complexity of a function f is the minimum cost of a randomized protocol that computes f with error at most $\frac{1}{4}$ on any input (x, y). The error is over the internal coin tosses of the protocol.

2.2 Tradeoffs

We now describe formally our tradeoff problem in the two-party communication complexity model. Similar definitions can be given for other models we consider. Our goal is to find a boolean function $f : X \times Y \to \{0, 1\}$ with the following properties:

- $f(x, y)$ can be computed efficiently, that is in polynomial time, if both the inputs $x \in X$ and $y \in Y$ are given.
- f has very efficient communication protocols, that is, protocols with communication complexity $(\log n)^c$ for some c.
- There is no protocol for f which is simultaneously communication and computation efficient. In other words, any protocol in which Alice and Bob use only polynomial time for local computation requires almost linear number of bits of communication in the worst case.

2.3 UP Relations

Definition 2.1. *A relation $R \subseteq \Sigma^* \times \Sigma^*$ is said to be a* UP relation *(with witness size n^k) if*

1. *there exists a deterministic Turing machine that decides the language $\{(x,w)|(x,w) \in R\}$ in polynomial time.*
2. *for every x, there exists at most one w such that $(x,w) \in R$ and furthermore, this w satisfies $|w| = |x|^k$. We denote this w, if it exists, by $w(x)$.*

The search problem *corresponding to R is the problem of finding w such that $R(x,w)$ holds, given x.*

We will assume the existence of UP relations for which the corresponding search problem is very hard. Such an assumption is standard in cryptography since the existence of strong one-way permutations implies the existence of such hard UP relations. More formally,

Definition 2.2. *We say is that a UP relation R is $T(n)$-hard if no probabilistic algorithm running in time $2^{O(T(n))}$ solves the search problem corresponding to R.*

3 Tradeoffs in the Two-Party Communication Complexity Model

We start with the definition of the boolean function we consider.

Definition 3.1. *Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a UP relation with witness size n^k. Consider the 2-player (Alice and Bob) boolean function $f_R : \{0,1\}^{n+n^k} \times \{0,1\}^{n^k} \to \{0,1\}$.*

$$\textit{Alice's input: } (x,z) \in \{0,1\}^n \times \{0,1\}^{n^k}; \qquad \textit{Bob's input: } w \in \{0,1\}^{n^k}$$

$$f_R((x,z),w) = \begin{cases} \langle z,w \rangle & \text{if } R(x,w) \text{ holds} \\ 0 & \text{otherwise} \end{cases}$$

where $\langle a,b \rangle$ denotes the inner product of a,b modulo 2.

Theorem 3.2. *Let R be a $T(n)$-hard UP reaction. Then, thepredicate f_R has the following properties.*

1. *f_R is computable in polynomial time.*
2. *There exists a randomized protocol that computes f_R with $O(\log n)$-bit communication.*
3. *If Alice and Bob are computationally bounded, then any randomized protocol for f_R, even with multiple rounds, will require $\Omega(T(n))$ bits of communication.*

Proof. Observe that f_R can be computed efficiently given both its inputs. We just need to check that $R(x,w)$ holds and if so, output $\langle z,w \rangle$.

Lemma 3.3. *If Alice is computationally unbounded, then there exists a randomized protocol that computes* f_R *with* $O(\log n)$*-bit communication.*

Proof. Alice computes the unique w such that $R(x, w)$ holds. Alice and Bob then engage in an "equality" protocol[3] to check that Bob's input equals w. If so, she computes and sends Bob the answer $\langle z, w \rangle$. □

The following lemma demonstrates that such a communication-efficient protocol is unlikely when Alice and Bob are computationally bounded. In fact, it is sufficient for the proof that only Alice is computationally bounded. Bob is allowed to be computationally unbounded.

Lemma 3.4. *Suppose there exists a* $b(n)$*-bit communication randomized multiround protocol* Π *that computes* f_R *involving Alice whose running time is at most* $T_A(n)$*, then there exists a randomized algorithm that solves the search problem corresponding to R in time* $\text{poly}(n, 2^{b(n)}) \cdot T_A(n)$.

Proof. For the rest of the argument, we assume that for any x, w is the unique w such that $R(x, w)$ holds, denoted by $w(x)$. Hence, for our purposes, $f_R((x, z), w) = \langle z, w \rangle$.

Our goal is to relate the search problem of computing w given x to the problem of computing $\langle z, w \rangle$ with a low communication protocol. Our approach is to convert a low communication protocol into an efficient oracle that computes $\langle z, w \rangle$ with some advantage over random guessing. Given such an oracle, we can then use the Goldreich-Levin reconstruction algorithm to compute a small number of candidates for w. More precisely, we create a "small" set of oracles, one of the oracles computes $\langle z, w \rangle$ with some nontrivial advantage. We try each oracle by exhaustive search, and use the fact that we can recognize the correct w.

Converting Protocols into Oracles

Let $\mathcal{T}$ be a transcript. For simplicity, we assume Alice outputs $f_R((x, z), w)$ as its final bit; this convention increases the size of the transcript by at most 2 bits. Thus, $\mathcal{T}$ includes a "guess" as to $\langle z, w \rangle$. We define the probabilistic oracle $A_{\mathcal{T}}(x, z)$ for computing $\langle z, w \rangle$, as follows.

Algorithm $A_{\mathcal{T}}$ (Input: $(x, z) \in \{0,1\}^n \times \{0,1\}^{n^k}$).
Simulate the protocol Π from Alice's end. Whenever a message from Bob is required, use the transcript $\mathcal{T}$ to obtain the corresponding message. If at any point the message generated by Alice according to the protocol Π disagrees with the contents of the transcript $\mathcal{T}$, abandon the protocol and output a random bit b. Otherwise, follow the protocol to the end and output the bit b generated by the protocol Π.

First we define our notation for the advantage of Π and $A_{\mathcal{T}}$ in guessing $\langle z, w \rangle$.

Definition 3.5. *Let* $x \in \{0,1\}^n$, $w = w(x)$ *and* z *be distributed uniformly. We define* $\text{ADV}(\Pi, x)$ *by*

$$\text{ADV}(\Pi, x) = \Pr[\textit{Alice outputs } \langle z, w \rangle] - \Pr[\textit{Alice doesn't output } \langle z, w \rangle],$$

[3] Recall that the randomized communication complexity of equality is $O(\log n)$.

where Alice and Bob run Π with respective inputs (x, z) and w, and the probability is taken over the choice of z and over the coin tosses of Alice and Bob. We define $\text{ADV}(A_{\mathcal{T}}, x)$ *analogously. Fixing x and a transcript $\mathcal{T}$, we define* $\text{ADV}(\Pi, x, \mathcal{T})$ *by*

$$\text{ADV}(\Pi, x, \mathcal{T}) = \Pr[\mathcal{T} \textit{ occurs and Alice outputs } \langle z, w\rangle] - \Pr[\mathcal{T} \textit{ occurs and Alice doesn't output } \langle z, w\rangle].$$

Note that the only contribution to $A_{\mathcal{T}}$'s advantage is by events in which $\mathcal{T}$ occurs, hence we do not bother to define $\text{ADV}(A_{\mathcal{T}}, x, \mathcal{T})$. It follows from the definitions that,

$$\text{ADV}(\Pi, x) = \sum_{\mathcal{T}} \text{ADV}(\Pi, x, \mathcal{T}). \tag{1}$$

Since the protocol Π computes f_R correctly, it holds that $\text{ADV}(\Pi, x) \geq \frac{1}{2}$ for every x. Since there are at most $2^{2b(n)}$ possible transcripts $\mathcal{T}$, it follows from Equation (1) that for every $x \in \{0,1\}^n$, there exists a transcript $\mathcal{T}^*$,

$$\text{ADV}(\Pi, x, \mathcal{T}^*) \geq \frac{1}{2^{2b(n)+1}} \tag{2}$$

Let $\rho_w(\mathcal{T})$ be the probability that Bob's coins are consistent with $\mathcal{T}$. Note that $\rho_w(\mathcal{T})$ is independent of z. It can easily be verified from the definitions that

$$\text{ADV}(\Pi, x, \mathcal{T}) = \text{ADV}(A_{\mathcal{T}}, x)\rho_w(\mathcal{T}). \tag{3}$$

Since $0 \leq \rho_w(\mathcal{T}) \leq 1$, it follows from Equation (2) that

$$\text{ADV}(A_{\mathcal{T}^*}, x) \geq \frac{1}{2^{2b(n)+1}}. \tag{4}$$

Set $\varepsilon = \frac{1}{2^{2b(n)+1}}$. Now we run the Goldreich-Levin algorithm GL (See Theorem 3.6) with parameters n, ε, oracle access to $A_{\mathcal{T}^*}(x, .)$ and predicate $R(x, .)$.

Theorem 3.6 (Goldreich-Levin [GL89]). *There exists a randomized algorithm GL with oracle access to a function and a predicate satisfying the following: Fix $u \in \{0,1\}^n$. Let $h : \{0,1\}^n \to \{0,1\}$ be a randomized algorithm such that $h(v) = \langle u, v\rangle$ with probability at least $\frac{1}{2} + \varepsilon$ where the probability is over choice of v, picked uniformly at random, and the internal coin tosses of h. Let $P : \{0,1\}^n \to \{0,1\}$ be a polynomial time computable predicate such that $P(v) = 1$ iff $u = v$. Then, the randomized algorithm GL with oracle access to h and P satisfies*

$$\Pr[GL^{h,P}(n, \varepsilon) = u] \geq \frac{3}{4}$$

Moreover, the running time of GL is at most $\text{poly}(n, \frac{1}{\varepsilon})$.

Theorem 3.6 guarantees that the algorithm GL computes w in time $\text{poly}(n, 1/\varepsilon)$ with constant probability. However, we do not have the transcript $\mathcal{T}^*$. (Recall that we only know that there exists a transcript $\mathcal{T}^*$ that satisfies Equation (4), we do not how to obtain one.) For this purpose, we run the Goldreich-Levin algorithm *GL* for every possible transcript $\mathcal{T}$ with parameters n and ε. One of these must succeed. Moreover, we can check which one succeeds by verifying that $R(x, w)$ holds. The total time taken by this algorithm is at most $2^{2b} \cdot \text{poly}(n, 2^{2b+1}) \cdot T_A(n) = \text{poly}(n, 2^b) \cdot T_A(n)$. This proves Lemma 3.4. □

To conclude the proof of the tradeoff result, we now use the assumption that the search problem corresponding to UP relation R does not have randomized algorithm that run in time $2^{o(T(n))}$ on inputs of length n. Therefore, $\text{poly}(n, 2^b) \cdot T_A(n) \geq 2^{\Omega(T(n))}$ and hence $b(n) = \Omega(T(n))$ since $T_A(n)$ is polynomially bounded in n. □

Remarks:

1. If we make the assumption that there is a search problem in UP that does not have sub-exponential time randomized algorithms, we get a very strong tradeoff. Such an assumption is used in cryptography.
2. We can prove the same result under a weaker assumption that the class FewP has a relation whose search problem is hard. In this case, we could use the set membership function instead of equality.
3. If the search problem corresponding to the relation R had average-case complexity at least $2^{\Omega(T(n))}$ when x is chosen from the distribution $\mathcal{D}$ (instead of worst case complexity), then the same proof as above demonstrates that f_R has average-case communication complexity at least $\Omega(T(n))$ for polynomially bounded Alice and Bob when x is chosen from the distribution $\mathcal{D}$, z uniformly and $w = w(x)$.

4 Communication Versus Computation in the Query Complexity Model

We consider the query complexity model in which a decision procedure D probes its input x choosing to look at some bits, but not others. The query complexity of a predicate P on n-bit inputs is given by $\min_D \max_x$ (# probes D makes on x). Here, D ranges over all decision procedures for P and x ranges over all inputs of length n.

We can consider the computationally bounded analog of this measure, where D is restricted to run in probabilistic polynomial time. Some subtleties arise in such a definition. For example, D must be quantified before n, since polynomial time is an asymptotic notion, but under this quantification there may be no "best" D for all inputs. Also, we may wish to augment our definitions to allow for an error probability.

Fortunately, Theorem 4.2 establishes a tradeoff that is clearly resilient to these technical issues.

Definition 4.1. *We say that a one-way permutation p is $\ell(n)$-lsb hard if no probabilistic polynomial-time procedure, on input x, can compute (simultaneously) the $l(n)$ least significant bits of $p^{-1}(x)$ with probability non-negligibly greater than $2^{-\ell(n)}$, where x is chosen uniformly from $\{0,1\}^n$.*

We note that such permutations exist based on the hardness of computing discrete logarithms over composite integers [SS90,HSS93].

Theorem 4.2. *Let p be $\ell(n)$-lsb hard. Then there exists a predicate $C_p : (\{0,1\}^n)^{2^{\ell(n)}+1} \longrightarrow \{0,1\}$ with the following properties:*

1. *C_p is computable in polynomial time.*
2. *The query complexity of C_p is at most $2n$.*
3. *No polynomial-time bounded decision procedure Q can compute C_p querying only $2^{\alpha\ell(n)}$ bits, where $\alpha < 1$ is any constant. In particular, there is a distribution on the inputs so that if Q computes C_p with advantage ε, then one can compute $lsb_{\ell(n)}(x)$ from $p(x)$ with probability $\Omega(\varepsilon 2^{-\alpha\ell(n)})$.*

Proof. (Sketch) For notational simplicity, we write ℓ instead of $\ell(n)$. We define $C_p(y, x_1, \ldots, x_{2^\ell})$ to be 1 iff there exists some i, $1 \le i \le 2^\ell$, such that $p(x_i) = y$ and $lsb_\ell(x_i) = i$ (treating i as an ℓ-bit string).

The predicate C_p is computable in polynomial time, since we can run over all the (polynomially-many) possible values of i. To see that C_p has query complexity at most $2n$, consider the following (computationally unbounded decision procedure):

1. Query y (which is n bits long)
2. Compute $x = p^{-1}(y)$ and $i = lsb_\ell(x)$.
3. Query x_i (which is n bits long), and accept iff $x_i = x$.

Our proof that no polynomial-time bounded decision procedure exists is by contradiction. Given Q, as above, we construct a polynomial-time algorithm G for guessing $lsb_\ell(x)$ from $p(x)$, as follows:

1. Given $p(x)$, compute $y = p(x)$ and choose $x_1, \ldots, x_{2^\ell}$ uniformly at random from $\{0,1\}^n$.
2. Run Q on input $(y, x_1, \ldots, x_{2^\ell})$. Define I by

$$I = \{i : Q \text{ queries at least one bit of } x_i\}.$$

3. Choose a random index i from I and output i (as an ℓ-bit quantity).

We relate the success probability of G to Q's advantage, ε at computing $C_p(y, x_1, \ldots, x_{2^\ell})$ under the distribution of inputs obtained as follows:

1. Choose x uniformly from $\{0,1\}^n$, and let $y = p(x)$ and $i = lsb_\ell(x)$.
2. For $j \neq i$, choose x_j uniformly from $\{0,1\}^n$.
3. With probability 1/2, choose $x_i = x$ (the predicate is true). Else, choose x_i uniformly from $\{0,1\}^n - x$ (the predicate is false).

Clearly, if on a particular run, Q never queries any bit in x_i, it has no advantage in guessing the value of the predicate. It follows that with probability $\Omega(\varepsilon)$, $i \in I$, where I is defined as above. In this case, choosing from I uniformly will yield i with probability $1/|I|$. Since $I \leq 2^{\alpha\ell(n)}$, the theorem follows. □

Our construction only assumes that $p()$ is strong against polynomial adversaries, resulting in any polynomial tradeoff. With stronger assumptions on the simultaneous hardness of bits in $p()$, we can prove any sub-exponential tradeoff.

5 Solution to the Riddle

We now present the solution to the riddle introduced in the introduction. Let s_i denote the sum of the entries in the ith row of M. We show how $k = n + 1$ players can communicate $\mathsf{PS}(M) = \prod_{i=1}^{n} s_i$ to the referee by each sending a single, *efficiently computable* element of F. (The same solution will work for any larger number of players.) The high-level idea is to first convert the "additive" representation of s_i to a degree-1 polynomial representation over a sufficiently large extension field, then make each player locally multiply its values of the n polynomials (one for each s_i), and finally project down to the original field. The protocol's outline is described below.

1. Each entry of M is lifted to an extension field F' of F such that $|F'| \geq k+1$. (This is only a conceptual step and requires no action, since F is a subfield of F'.) Let $\alpha_1, \ldots, \alpha_k$ be distinct nonzero elements of F'.
2. The players locally process their entries of M, and each outputs a single element of F' for each row. Let $P_{i,j}$ denote the output of player j corresponding to the ith row. The values $P_{i,j}$ should satisfy the following requirement: for each i, the k points $(\alpha_j, P_{i,j})$ lie on a degree-1 polynomial over F' whose free coefficient is s_i. The implementation of this stage will be described below.
3. Each player j multiplies its n local outputs $P_{i,j}$ from the previous state, resulting in a single element $q_j \in F'$. Note that the k points (α_j, q_j) now lie on a degree-n polynomial whose free coefficient is precisely $\prod_{i=1}^{n} s_i = \mathsf{PS}(M)$. Since $k > n$, this polynomial can be uniquely determined by interpolation and its free coefficient can be written as $\sum_{j=1}^{k} \lambda_j q_j$ for some fixed coefficients $\lambda_j \in F'$. Each player j projects $\lambda_j q_j$ down to the original field using a field homomorphism $h : F' \to F$, and sends the result to the referee.
4. The referee outputs the sum of the k field elements it received.

It remains to describe the implementation of Step 2. Define a $k \times k$ matrix L over F' such that $L_{\ell,m} = 1 - \frac{\alpha_\ell}{\alpha_m}$. For each i, we let $P_{i,j} = \sum_{m=1}^{k} L_{j,m} M_{i,m}$. Note that since $L_{j,j} = 0$, player j can compute this sum based on his local input. It remains to argue that the above local computations indeed produce the required degree-1 representation of s_i. This follows by noting that for any column m of L, the values $(\alpha_\ell, L_{\ell,m})$ lie on a degree-1 polynomial whose free coefficient is 1. By linearity, the values $(\alpha_j, P_{i,j})$ lie on a degree-1 polynomial whose free coefficient is $\sum_{j=1}^{k} 1 \cdot M_{i,j} = s_i$. Thus, we have shown:

Theorem 5.1. *The function* $\mathsf{PS}(M) = \prod_{i=1}^{n} \sum_{j=1}^{k} M_{ij}$, *where* $k > n$, *admits a* computationally efficient *simultaneous messages protocol in which each player holds all but one column of M and sends a single field element to the referee.*

References

[BGKL03] BABAI, L., GÁL, A., KIMMEL, P. G., AND LOKAM, S. V. Communication complexity of simultaneous messages. *SIAM Journal of Computing 33,* 1 (2003), 137–166. (Preliminary Version in *12th STACS,* 1995).

[BTY94] BEAME, P., TOMPA, M., AND YAN, P. Communication-space tradeoffs for unrestricted protocols. *SIAM Journal of Computing 23,* 3 (June 1994), 652–661. (Preliminary Version in *31st FOCS,* 1990).

[CG93] CANETTI, R., AND GOLDREICH, O. Bounds on tradeoffs between randomness and communication complexity. *Computational Complexity 3* (1993), 141–167. (Preliminary Version in *31st FOCS,* 1990).

[CFL83] CHANDRA, A. K., FURST, M. L., AND LIPTON, R. J. Multi-party protocols. In *Proc.* 15*th ACM Symp. on Theory of Computing* (Boston, Massachusetts, 25–27 Apr. 1983), pp. 94–99.

[DGS87] DURIS, P., GALIL, Z., AND SCHNITGER, G. Lower bounds on communication complexity. *Information and Computation 73,* 1 (Apr. 1987), 1–22.

[GL89] GOLDREICH, O., AND LEVIN, L. A. A hard-core predicate for all one-way functions. In *Proc.* 21*st ACM Symp. on Theory of Computing* (Seattle, Washington, 15–17 May 1989), pp. 25–32.

[HIKNV] HARSHA, P., ISHAI, Y., KILIAN, J., NISSIM, K., AND VENKATESH, S. Communication vs. computation. Technical Report (to be posted in ECCC). Available at `http://theory.csail.mit.edu/~prahladh/papers/`

[HSS93] HÅSTAD, J., SCHRIFT, A. W., AND SHAMIR, A. The discrete logarithm modulo a composite hides $O(n)$ bits. *Journal of Computer and System Sciences 47*, 3 (Dec. 1993), 376–404. (Preliminary Version in *22nd STOC,* 1990).

[NW93] NISAN, N., AND WIGDERSON, A. Rounds in communication complexity revisited. *SIAM Journal of Computing 22,* 1 (Feb. 1993), 211–219. (Preliminary Version in *23rd STOC,* 1991).

[PS84] PAPADIMITRIOU, C. H., AND SIPSER, M. Communication complexity. *Journal of Computer and System Sciences 28,* 2 (Apr. 1984), 260–269. (Preliminary Version in 14*th STOC,* 1982).

[SS90] SCHRIFT, A. W., AND SHAMIR, A. The discrete log is very discreet. In *Proc. 22nd ACM Symp. on Theory of Computing* (Baltimore, Maryland, 14–16 May 1990), pp. 405–415.

[Yao79] YAO, A. C.-C. Some complexity questions related to distributive computing (preliminary report). In *Proc.* 11*th ACM Symp. on Theory of Computing* (Atlanta, Georgia, 30 Apr.–2 May 1979), pp. 209–213.

[Yao86] YAO, A. C.-C. How to generate and exchange secrets? (extended abstract). In *Proc.* 27*th IEEE Symp. on Foundations of Comp. Science* (Toronto, Ontario, Canada, 27–29 Oct. 1986), pp. 162–167.

Optimal Website Design with the Constrained Subtree Selection Problem*

Brent Heeringa[1,2] and Micah Adler[1]

[1] Department of Computer Science, University of Massachusetts, Amherst
140 Governors Drive Amherst, MA 01003
[2] Department of Computer Science, Williams College, Williamstown, MA, 01267
{heeringa,micah}@cs.umass.edu

Abstract. We introduce the Constrained Subtree Selection (CSS) problem as a model for the optimal design of websites. Given a hierarchy of topics represented as a DAG G and a probability distribution over the topics, we select a subtree of the transitive closure of G which minimizes the expected path cost. We define path cost as the sum of the page costs along a path from the root to a leaf. Page cost, γ, is a function of the number of links on a page. We give a sufficient condition for γ which makes CSS NP-Complete. This result holds even for the uniform probability distribution. We give a polynomial time algorithm for instances of CSS where G does not constrain the choice of subtrees and γ favors pages with at most k links. We show that CSS remains NP-Hard for constant degree DAGs, but also provide an $O(\log(k)\gamma(d+1))$ approximation for any G with maximum degree d, provided that γ favors pages with at most k links. We also give a complete characterization of the optimal trees for two special cases: (1) linear degree cost in unconstrained graphs and uniform probability distributions, and (2) logarithmic degree cost in arbitrary DAGs and uniform probability distributions.

1 The Constrained Subtree Selection Problem

In this paper, we study the optimal design of websites given a set of page topics, weights for the topics, and a hierarchical arrangement of the topics. Automatic website design provides a principled choice for information organization, facilitates individualized and user-centric site layout, and decreases the average time spent searching for relevant information.

As an example, imagine that A Different Drummer's Kitchen is creating a new website for their catalog of kitchenware. They want a website where their customers can quickly find information on specific products by descending a hierarchy of general to specific categories, much like the *Yahoo!* portal. They want to minimize the number of intermediate pages it takes to find pepper mills

* This research partially funded under NSF Research Infrastructure Award EIA-0080119, NSF Faculty Early Career Development Award CCR-0133664, and NSF ITR Grant ITR-0325726

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 757–769, 2004.

but not at the expense of filling a page with links to marginally related products like tea kettles, cookie cutters and aprons.

Constrained Subtree Selection (CSS) models these website design problems. We suppose that prior to site development, topics are hierarchically arranged by a designer to represent their natural organization. We represent this initial hierarchy as a rooted, directed acyclic graph, called the *constraint graph* where the nodes are categories, the leaves are topics and the edges are topical constraints. A path through the constraint graph follows a general to specific trajectory through the categories. For example, in the kitchenware hierarchy cutlery leads to knives leads to paring knives. Note that a particular paring knife may belong to other categories (like the knife manufacturer), and thus the constraint graph may be a DAG that is not a directed tree.

A website should preserve this logical relationship in its own topology. We represent websites as directed trees, where pages are represented by nodes and links are represented by directed edges. We require that the directed tree satisfy two conditions. First, there must be a one-to-one mapping $\mathcal{M}$ of nodes in the website to nodes in the constraint graph. This is a constraint since adding new nodes would infer structure that is not represented in the constraint graph. Second, if categories in the constraint graph are not included in the website, a user should still be able to descend naturally toward the desired topic. This means that if page A descends directly from page B in the website then $\mathcal{M}(A)$ must be reachable from $\mathcal{M}(B)$ in the constraint graph. A necessary and sufficient condition for both of these conditions to be satisfied is that the website be a directed subtree of the transitive closure of the constraint graph. In this way, the initial hierarchy offers a set of constraints on topic layout but frees the web site developer to move specific pages to more general categories. Finally, we stipulate that the subtree include the root and leaves of the constraint graph since they represent the entry and endpoints of any natural descent in the website.

Our objective is to find the website which minimizes the expected time searching for a topic. We say the cost of a search is the sum of the cost of the pages along the search path. We represent page cost as a function of the number of links on a page, so we call it the *degree cost.* Adding more links decreases the height of the tree, but increases the time spent searching a page; minimizing the number of links on a page makes finding the right link easy, but adds height to the website. For this reason, we can also think of the degree cost as capturing the inherent tension between breadth and depth. Different scenarios demand different tradeoffs between these competing factors. For example, if network latency is a problem when loading web pages then favoring flatter trees with many links per page decreases idle waiting. In contrast, web browsers on handheld devices have little screen area, so to reduce unnecessary scrolling it's better to decrease the number of links in favor of a deeper tree. In the spirit of generality, we attempt to keep our results degree-cost independent. At times however, we examine particular degree costs such as logarithmic and linear.

Naturally, some pages are more popular than others. We capture this aspect with a probability distribution over the topics, or equivalently by topic weights.

Given a path, we say the weighted path cost is the sum of the page costs along the path (i.e. the unweighted path cost) multiplied by the topic weight. Since we want a website that minimizes the average search time for a topic, we take the cost of a tree as the expected path cost for a topic chosen from the probability distribution over the topics. An optimal tree is any minimal cost subtree of the transitive closure of the constraint graph that includes the leaves and root.

We're now in a position to define our model more formally. Let T be a directed tree (a branching) with n leaves where leaf u_i has weight w_i. Let $u_i = (u_{i_1}, \ldots, u_{i_m})$ be a path from the root of T to the i^{th} leaf of T. If $\delta(v)$ is the out-degree of node v and γ is a function from the positive integers to the reals, then the cost of u_i is:

$$c(u_i) = \sum_{j=1}^{m-1} \gamma(\delta(u_{i_j}))$$

and the weighted cost is $w_i \cdot c(u_i)$. The cost of T is the sum of the n weighted paths: $c(T) = \sum_{i=1}^{n} w_i \cdot c(u_i)$.

An instance of the *Constrained Subtree Selection* problem is a triple $I = (G, \gamma, (w_i))$ where G is a rooted, directed, acyclic *constraint* graph with n leaves, γ is a function from the positive integers to the non-negative reals, and $(w_i) = (w_1 \ldots w_n)$ are non-negative, real-valued leaf weights summing to one. A solution to I is a directed subtree T (hereafter a tree) of the transitive closure of G that includes the leaves and root of G. An optimal solution is one that minimizes the cost function under γ. Sometimes we consider instances of CSS with fixed components. For example, we might study the problem when the degree cost is always linear, or leaf weights form a uniform probability distribution. We refer to these cases as *CSS with* γ or *CSS with equal leaf weights* so that it is clear that γ and (w_i) are not part of the input.

Websites are not the only realization of this model. For example, consider creating and maintaining user-specific directory structures on a file system. One can imagine that the location of `/etc/httpd` may be promoted to the root directory for a system administrator whereas a developer might find `~/projects/source` directly linked in their home directory. Similarly, users may have individualized views of network filesystems targeted to their own computing habits. In this scenario a canonical version of the network structure is maintained, but the CSS problem is tailored to the individual. In general, any hierarchical environment where individuals actively use the hierarchy to find information invites modeling with CSS.

1.1 Results

In this paper, we give results on the complexity of CSS, polynomial time algorithms and characterizations of the optimal solution for certain restricted instances of CSS, and a polynomial time constant approximation algorithm for fixed-degree constraint graphs in a broad class of degree costs.

First, we show a sufficient condition on the degree cost which makes Constrained Subtree Selection NP-Complete in the strong sense for arbitrary input

DAGs. Many natural degree costs (e.g., linear, exponential, ceiling of the logarithm) meet this condition. Furthermore, this result holds even for the case of uniform leaf weights.

Because of this negative result, we turn our attention to restricted scenarios and approximation algorithms. We first consider the case of inputs where the topological constraints of the graph are removed (i.e., where the constraint graph allows any website tree to be constructed). Within this scenario, we consider a general class of degree functions, called *k-favorable* degree costs, where the optimal solution favors trees such that all the nodes have out-degree k or less. We give an $O(n^{k+\gamma(k)})$ time algorithm for finding an optimal tree when the topological constraints of the graph are removed and when γ is non-decreasing, restricted to functions with integer co-domains, and k-favorable. This result holds for arbitrary leaf weights, and demonstrates that the computational hardness of the CSS problem is a result of the conditions imposed by the constraint graph. We also provide an exact characterization of the optimal solution for the linear cost function (which is 3-favorable) in the case of a uniform probability distribution and no topological constraints.

We next consider the case of bounded out-degree constraint graphs. We demonstrate that when γ favors complete k-ary trees, CSS remains NP-Hard for graphs with degree at most $k+5$ and uniform leaf weights. However, we also give a polynomial time constant factor approximation algorithm for constraint graphs with degree no greater than d and arbitrary leaf weights, provided that γ is k-favorable for some k. The approximation ratio depends on both d and γ. Additionally, we show the linear degree cost favors complete k-ary trees.

Finally, for arbitrary constraint graphs, $\gamma(x) = \lceil \log_2(x) \rceil$, and uniform leaf weights, we demonstrate that even though this case is NP-Complete, the depth-one tree approximates the optimal solution within an additive constant of 1. Due to space constraints, most of the proofs of our results appear in [9].

1.2 Related Work

Constrained Subtree Selection is related to three distinct bodies of work. The first is work in the AI community by Perkowitz and Etzioni [1]. While the authors are concerned with many issues related to building intelligent websites, they concentrate on the *index page synthesis problem* which seeks to "automatically generate index pages to facilitate efficient navigation of a site or to offer a novel view of the site" using new clustering and concept learning algorithms which harness the access logs of the website. Here efficient means making sure visitors find their topic of interest (recall) and minimizing the amount of time spent finding that topic (effort). The time spent finding a topic is measured by the time it takes to scan successive pages for the right link and the overall number of links taken. Notice their definition of effort strongly resembles our notion of cost. In this light, our work may be viewed as supplying a model for the index page synthesis problem as it relates to minimizing the average effort in finding the topic of interest.

The Hotlink Assignment (HA) problem introduced by Bose et. al ([2], [3]) also relates to our problem. Here, a website *is* represented by a DAG with a probability distribution over the leaves. A constant number of arcs, called hotlinks, are added to the DAG to minimize the expected distance from the root to leaves. Since multiple paths from the root to a leaf may exist, the expected distance is computed using the shortest path. The problem is NP-Hard for arbitrary graphs, but tractable for binary trees with arbitrary probability distributions over the leaves. Recently, the problem was revised so that nodes have a fixed page cost proportional to the size of the web page they represent [4]. In this formulation, the cost of a path is not its length, but instead the sum of the page costs on the path. The problem seeks to assign at most k hotlinks per node to minimize the expected page cost.

Hotlink Assignment (HA) is different from CSS for a number of reasons. The first is how we model page cost. In HA, page cost does not change with the addition of hotlinks. In CSS, the cost of a page is a function of the number of links it contains. This means we can think of CSS as minimizing the expected amount of choice a user faces when traversing a website as opposed to HA which essentially minimizes the expected amount of time waiting for pages to load. Note that the generality of our degree function means we can also include a network latency term in to our degree cost. Another difference is how we view the initial topologies. With HA, the DAG represents a website that needs improving. In CSS, we take the DAG as a set of constraints for building a website. This difference is both conceptual and technical. While the *shortest path* tree can be extracted from the Hotlink DAG after the links are assigned, a tree with longer paths cannot be considered. We consider all paths in our subtree selection since longer paths are viewed in terms of constraints and not cost. Finally, HA assigns a constant number of hotlinks where CSS has no restriction. The constant number is important to HA because without this restriction, the optimal website would always have hotlinks from the root to all the leaves. In CSS this corresponds to a constant degree function where the optimal tree is always the depth-one tree.

Certain relaxed versions of the Constrained Subtree Selection problem bear resemblance to the Optimal Prefix-free Coding (OPC) problem: The general problem asks for a minimal prefix code for n weighted words using at most r symbols where symbol i has cost c_i ([5], [6]). This problem is equivalent to finding a tree with n leaves where all internal nodes having degree at most r, the length of the i^{th} edge of a node is c_i, and the external weighted path length is minimized. There is no known polynomial time solution for the general problem, but it is not known to be NP-Hard. When the costs are restricted to fixed integers, there is an $O(n^{C+2})$ time dynamic programming algorithm where C is the maximum integer cost [7].

On the surface, our problems appear similar because they both ask to minimize external weighted path cost—the sum of weighted path costs from the root to each of the leaves. However the cost in OPC is edge-based, where the cost of CSS is node-based. More appropriately, the node cost in CSS is dynamic; adding an additional edge means the cost of the node changes. If we view the

node costs as edge costs, than adding an edge potentially changes the edge costs of all its siblings. This difference, along with the lack of prior constraints on the tree structure in prefix-free codes, distinguish the problems enough that it seems difficult to transform one to the other. Still, by relaxing the graph constraints, and restricting the degree cost, we can show that some instances of CSS are exactly instances of OPC for a binary alphabet with equal character costs, and that in more general cases, we can adapt portions of the dynamic programming algorithm for finding optimal prefix-free codes to our find optimal trees in the CSS problem.

2 Complexity

In this section we show that even when the leaf weights are equal, the CSS problem is NP-Complete in the strong sense for a large class of degree functions. The reduction is from Exact Cover by 3-Sets (XC3) [8] which, when given a set X of $3k = n$ items and a set C of three item subsets of X, asks whether a subset of C exists that exactly covers X. The related decision problem for CSS asks whether a subtree of G exists with cost at most D.

Definition 1. *Let γ be a non-decreasing function. If for all integers $k \geq 1$, there exists some $c > 0$ and some function $s(k) \in O(k^c)$ such that*

$$\gamma(s(k) + k + 1) > \gamma(s(k) + k) + \gamma(3)\frac{3k}{s(k) + 3k}$$

then γ is degree-3-increasing

Many degree costs are degree-3-increasing. For example, the linear degree cost, $\gamma(x) = x$, (choose $s(k) = 7k$), exponential degree cost $\gamma(x) = \exp(x)$ (again, $s(k) = 7k$ will work) and ceiling of the logarithm degree cost $\gamma(x) = \lceil \log_2(x) \rceil$ (choose $s(k) = 3k$) all meet the definition. The following theorem tells us that when γ is degree-3-increasing and in NP, that CSS with γ is NP-complete for any DAG and any probability distribution.

Theorem 1. *For any degree-3-increasing degree cost γ where γ is in NP, CSS with γ is NP-Complete.*

Because CSS is not a number problem when the leaf weights are equal (i.e. we can ignore them when computing cost), we can show that it is NP-Complete in the strong sense for a broad class of degree costs.

Theorem 2. *For any degree-3-increasing degree cost γ, γ in NP, if there exists $c > 0$ such that $\gamma(s(n/3) + n/3) = O(n^c)$ then CSS with γ is NP-Complete in the strong sense.*

3 Subtree Selection Without Constraints

Imagine we are building a website without any prior knowledge of the organization of the topics. The most natural solution is to build a website that minimizes the expected search time for the topics, but has no constraints on the topology. This design problem is an instance of CSS where any website is a subtree of the transitive closure of the constraint graph. In this section we'll show that these instances are solvable in polynomial time for a broad class of degree functions. This is interesting because it means the NP-Hardness of our problem comes from the graphical constraints rather than the degree cost and leaf weights.

We begin with some definitions. A tree is *full* when every interior node has at least two children. A constraint graph G with n leaves is called *constraint-free* when every full tree with n leaves is a subtree of the transitive closure of G. This means that G does not constrain the optimal subtree. A tree is *monotone* when the leaf weights cannot be permuted (among the leaves) to yield a tree of lower cost. Hence, if we listed the leaves in increasing order by path cost, the weights of the leaves would be in decreasing order. Prom these definitions it's easy to see that every instance of CSS has at least one optimal solution that is full and that all solutions to CSS are monotone when the the graph is constraint-free.

A degree cost γ is *k-favorable* if and only if there exists $k > 0$ such that any instance of CSS where G is constraint-free has an optimal solution under γ where the out-degree of every node is at most k. This definition is useful because it gives us a bound on the out-degree of any node in an optimal solution to the CSS problem where the graph is constraint-free. Proving that a particular γ exhibits k-favorability for some k typically means showing that any node with out-degree at least $(k+1)$ can be split into nodes of smaller degree with no increase to the overall cost of the tree. Many degree costs are k-favorable. For example the linear degree cost $\gamma(x) = x$ is 3-favorable, but not 2-favorable [9]. In section 5 we characterize the optimal tree for the linear degree cost when the graph is constraint-free and the weights are equal. It is worth noting that any instance of CSS where G is constraint-free and γ is 2-favorable reduces to the optimal prefix code problem for a binary alphabet with equal letter costs. In other words, Huffman's greedy algorithm ([10]) solves these problems. Examples of degree costs that favor binary trees are $\gamma(x) = \lceil \log(x) \rceil$ and $\gamma(x) = e^x$.

But what happens when γ is k-favorable but not $k-1$-favorable and $k > 2$? More generally, is there a polynomial time algorithm that solves $(G, \gamma, (w_i))$ when G is constraint-free and γ is k-favorable? In this section we give a dynamic programming algorithm which leads to the following result.

Theorem 3. *There is a $O(n^{\gamma(k)+k})$ time algorithm which finds an optimal solution to any instance of CSS where G is constraint-free, γ is k-favorable for some integer k, non-decreasing and maps the positive integers to the positive integers.*

We adapt the dynamic programming algorithm for finding optimal prefix-free codes (OPC) given by Golin and Rote ([7]) to the CSS problem. We highlight some of the similarities and differences between the two algorithms here but give a complete description of our algorithm and a proof of Theorem 3 in [9].

The solution to the optimal prefix-free coding problem with integer costs relies on a lopsided representation of a tree. A lopsided tree equates a node's level to its path cost from the root. In other words, if u is a node in T, and the path cost to u is C, then we say u is at level C. Restricting the cost to integers means the levels are also integers. Golin and Rote associate a signature with each tree level so that if a tree has h levels, then it has h signatures. Signatures are always taken with respect to the truncation of a tree at a certain level. If T is a tree with n leaves, then the *level-i-truncation* of T, denoted $\mathrm{Trunc}_i(T)$, prunes away all nodes of T with *parents* at levels deeper than i. The *level-i-signature* of T is the $(C+1)$ vector: $sig_i(T) = (m, l_1, \ldots, l_C)$ where m is the number of leaf nodes at levels 0 through i, l_j is the number of nodes at level $i+j$ in $\mathrm{Trunc}_i(T)$, and C is the largest symbol (edge) cost. If $v_1, \ldots, v_n$ are the leaves of T given in increasing order by level and $w_1 \ldots, w_n$ are the leaf weights given in decreasing order then the *level-i-cost* of T is $c_i(T) = \sum_{j=1}^{m} \mathrm{level}(v_j) w_j + \sum_{s=m+1}^{n} i \cdot w_s$ where m is the number of leaf nodes in $\mathrm{Trunc}_i(T)$.

The level-i-signature of a tree $(m, l_1, \ldots, l_C)$ equates to an entry in the dynamic programming table $\mathrm{MIN}[m, l_1, \ldots, l_C]$. This entry gives the minimum level-i-cost of all trees with signature $(m, l_1, \ldots, l_C)$. There are $O(n^{C+1})$ table entries since the number of nodes at the fringe of the tree never exceeds n. Note that the signature does not indicate level, so the value of an entry may correspond to the level-i-cost of trees at a variety of levels. Given a tree's signature at level $i+1$, it's possible to enumerate what level-i-signatures lead to it. Similarly, the level-$(i+1)$-cost of a tree can be written in terms of the level-i-cost of the tree associated with the signature that precedes it which gives a natural method for filling in the dynamic programming table.

When considering how level-$(i+1)$-signatures relate to level-i-signatures, we must consider structural changes to the tree. In the OPC domain, adding an edge does not change the lopsided structure of the rest of the tree. In our domain when an edge is added, the lopsided structure of the tree does change because the node degree changes. As a result, we cannot apply Golin and Rote's algorithm verbatim; we can use the subproblem representation (i.e. the signatures) by letting $C = \gamma(k)$ but filling in the table requires a different approach.

We must examine the way two trees with the same signature at level i can differ in their level-$(i+1)$-signature. Given a level-i-signature we must first choose how many level $(i+1)$ nodes will be internal, and them among those, which will have degree 2, degree 3, and so on. We denote these choices with a $(k+1)$-vector $a = (a_0, \ldots, a_k)$ called a child vector where a_0 is the number of nodes at level-$(i+1)$ that are internal to T and each a_j is the number among those a_0 having degree j. Note that $a_0 \leq l_1$ and that $a_1 = 0$ since there is always an optimal tree with no nodes having out-degree 1. Also, since $\sum_{j=2}^{k} a_j = a_0$ we know there are $O(n^{k-1})$ choices for a. In other words, given a level-i-signature, it is the possible parent of $O(n^{k-1})$ level-$(i+1)$-signatures. The following Lemma tells us exactly which signatures are children of a level-i-signature parent.

Lemma 1. *Let T be a tree with $sig_i(T) = (m, l_1, \ldots, l_{\gamma(k)})$. If $a = (a_0, a_1, \ldots, a_k)$ is the level-i-child vector of T yielding T', then $sig_{i+1}(T') = (m', l'_1, \ldots, l'_{\gamma(k)})$ where*

$$(m', l'_1, \ldots, l'_{\gamma(k)}) = (m + l_1, l_2, \ldots, l_{\gamma(k)}, 0) + b$$

with $b = (b_0, \ldots, b_{\gamma(k)})$ *where* $b_0 = -a_0$ *and* $b_{\gamma(i)} = i \cdot a_i$ *for* $2 \leq i \leq k$.

While Lemma 1 tells us how level-i-signatures relate to level-$(i+1)$-signatures it does not tell us how the costs relate. The second part of Lemma 5 from [7] tells us that if T is a tree with $\mathrm{sig}_i(T) = (m, l_1, \ldots, l_{\gamma(k)})$ then $c_{i+1}(T) = c_i(T) + \sum_{j=m+1}^{n} w_j$. Fortunately, this result holds for all monotone, lopsided trees with level-i-costs defined as above so even though our problem has a different dependency structure in the table, it does not require a new way of computing cost in terms of cost to subproblems. Golin and Rote give a linear ordering of the table entries that respects their dependency structure. This ordering works for us too, although their proof of this fact no longer applies because our table entries have a different dependency structure. We describe the ordering in [9] and show that it works for our problem too. What's most important is that viewing table entries as nodes and dependencies as edges still leaves us with a DAG, so any topological sort yields an appropriate order for filling in the table.

Here is a description of our algorithm. We repeatedly process table entries in an order that respects the dependency structure, beginning with the entry corresponding to the level-0-truncation of a single node with two children (MIN[0, 0, 2, ... , 0]) and ending with the entry corresponding to a tree with n leaves ($\mathrm{MIN}[n, 0, \ldots, 0]$). Given an entry we consider all its children (via Lemma 1) and then update the cost of the children (by Lemma 5 in [7]) if there is an improvement. After completing the table, the entry $\mathrm{MIN}[n, 0, \ldots, 0]$ contains the cost of the minimum tree. We can keep an additional table relaying points to the entries which yield the optimal cost to easily reconstruct the optimal tree. The $O(n^{\gamma(k)+k})$ running time of the algorithm follows because the table has $O(n^{\gamma(k)+1})$ entries of which each has at most $O(n^{k-1})$ dependencies to check.

4 Approximations

Many hierarchies have the property that no category has more than a constant number of subcategories. This means the out-degree of every node in the constraint graph is bounded above by a constant. In this section we give two theorems dealing with such cases. The first theorem says that even if we restrict the problem to DAGs of constant maximum degree, CSS remains NP-Hard for certain degree costs. The second theorem gives an $O(\log(k)\gamma(d+1))$ approximation algorithm for all instances of CSS where the maximum degree of the constraint graph is bounded above by some constant d, and γ is k-favorable and has a lower bound of 1.

Let a cost function be *k-tree optimal* if, for all instances of CSS with constraint-free graphs and equal leaf weights, the unique optimal website tree with k^c leaves, for any positive integer c, is a complete k-ary tree of depth c. For example, in [9] we show that the linear degree cost is 3-tree optimal.

Theorem 4. *For any cost function that is k-tree optimal, for any $k \geq 3$, the CSS problem is NP-Hard even when restricted to the uniform probability distribution and DAGs with degree at most $k + 5$.*

Consider the *Partitioned Exact Cover by 3 Sets* (PX3S) problem, which we define here. The input is a set S of $3q$ elements, where q is an integer, a collection C of subsets of S of size 3, and a partition P of the collection C into exactly q cells. We ask whether there is an exact cover of S that uses exactly one subset from each cell of P. The proof of Theorem 4 appears in [9], but we provide a high level overview here. The proof is in two parts. We first show that the PX3S problem is reducible to the CSS problem with a k-tree optimal cost function, restricted to DAGs of degree at most $k+r-1$, where r is the maximum number of subsets in any cell of the partition P. We then show that the PX3S problem is NP-Complete even when we restrict r to six.

Theorem 5. *For any constraint graph G with m nodes where every node has out-degree at most d and for every k-favorable degree cost γ where γ is bounded below by 1, CSS with G and γ has an $O(m)$ time $O(\log(k)\gamma(d+1))$-approximation to the optimal solution.*

Proof. We begin by giving a lower bound on any instance of CSS where the degree cost is k-favorable and bounded below by 1. Take W as the probability distribution over leaf weights, $W(x)$ as the total weight of the leaves in the subtree rooted at x and H as the entropy function.

Lemma 2. *For any k-favorable degree cost γ with γ bounded below by 1, $\frac{H(W)}{\log(k)}$ is a lower bound on the cost of an optimal solution to CSS with γ.*

The proof of the lemma appears in [9] but the main idea is that the cost of any optimal tree to the CSS problem is bounded below by the cost of the optimal prefix-free code over a k-ary alphabet with character costs 1 which is bounded below by $\frac{H(W)}{\log(k)}$ by Shannon's theorem.

Our approximation algorithm also requires the following result which is easy to prove (although we provide a proof in [9]).

Claim. For any tree with with weights on its m nodes, there exists at least one node, which, when removed, divides the tree into subtrees where every subtree has at most half the weight of original tree. Furthermore we can find such a node in $O(m)$ time.

Let $I = (G, \gamma, (w_i))$ be an instance of CSS where where every node in G has out-degree at most d and γ is k-favorable. Extract any spanning tree T from G. Using Claim 4 we can identity a node in T called the *splitter* which, when removed, divides T into subtrees where each subtree has at most half the probability mass of T. In our algorithm, we don't remove the splitter from the tree but rather, remove the edge(s) connecting it to its parent(s). We reconnect the splitter to the root of T. Recursively apply this procedure on the subtrees rooted by the children of the root of T and call the final tree T'. Note that T'

is still a subtree of the transitive closure of G since the splitter node is always descendent of the root of the tree under consideration. If G has m nodes then extracting a spanning tree from G takes $O(m)$ time since each node has constant degree. The complete procedure takes $O(m)$ time since applying Claim 4 to all m nodes can be accomplished in $O(m)$ time with some bookeeping.

Claim. If r and s are nodes in T' where s is the grandchild of r, then $W(r) \geq 2 \cdot W(s)$

This claim follows immediately from the construction of T' with respect to Claim 4. Since any two hops in T' divides the probability mass of the subtree in half, we know the depth of leaf i is bounded above $-2\log_2(w_i)$. Since each node in T' has degree at most $d+1$, the cost of T' is at most $2 \cdot \gamma(d+1) \sum_{i=1}^{n} w_i(-\log_2(w_i)) = 2 \cdot \gamma(d+1) \cdot H(W)$

Since $O(\gamma(d+1)H(W))$ approximates the lower bound of $H(W)/\log(k)$ by a multiplicative factor of $O(\log(k)\gamma(d+1))$ we have the desired result.

5 Leaves of Equal Weight

It is easy to imagine fledgling companies building websites without any prior popularity statistics on their products. To gather such statistics, they may want a website which puts all their products on an equal footing. Finding the optimal website for equally-weighted topics corresponds to instances of CSS with a uniform probability distribution over the leaves. We characterize optimal trees for these instances of CSS for the linear degree cost when the graph is constraint-free, and for the logarithmic degree cost for any DAG.

5.1 Linear Degree Cost

Theorem 6 gives the cost of an optimal tree for the linear degree function when the graph is constraint-free and $\gamma(x) = x$. We arrive at this cost by showing how to construct an optimal tree. Proof of the the construction's optimality is involved, but the tree is simple to describe: An optimal tree with n leaves begins with a complete tertiary tree with $\lfloor \log_3(n) \rfloor$ leaves. Additional leaves are added in pairs by splitting the leaves of the complete tertiary tree into binary nodes. Finally, if we still require more leaves, we add an additional edge to each binary node. In some sense, an optimal tree for n leaves is one that is always trying to be the most complete tertiary tree with n leaves.

Theorem 6. *If $(G, \gamma, (w_i))$ is an instance of CSS where G is constraint-free, $\gamma(x) = x$, and the n leaf weights are equal, then if $n \leq 2 \cdot 3^k$, where $k = \lfloor \log_3(n) \rfloor$, an optimal tree has cost $3nk + 4(n - 3^k)$ otherwise it has cost $3(k+1)(3^{k+1}) - ((3^{k+1} - n)(3(k+1)+2))$.*

5.2 Logarithmic Degree Costs

Another natural choice of degree cost is $\gamma(x) = \lg(x)$ (where $\lg = \log_2$) because it gives the number of bits needed to encode the out-degree of the node. In this section we show the depth-one tree (where the root has n edges directly to its n leaves) is an optimal solution to any instance of CSS where the n leaf weights are equal and $\gamma(x) = \lg(x)$. This result holds for arbitrary constraint graphs because the depth-one tree is always a subtree of the transitive closure. Proof of Theorem 7 is given in [9].

Theorem 7. *Let $I = (G, \gamma, (w_i))$ be an instance of CSS where $\gamma(x) = \log(x)$ and the n leaf weights are equal. An optimal tree for I is the depth-one tree.*

Finally, we noted in Sec. 2 that CSS with degree cost $\gamma(x) = \lceil \log_2(x) \rceil$ is NP-Hard even with equal leaf weights. This is somewhat surprising given the depth-one tree is optimal for $\gamma(x) = \log(x)$ with equal leaf weights. The result holds because the ceiling provides a place where the cost jumps enough so that any non-optimal tree suffers the impact of this slight increase. A corollary to Theorem 7 is that the depth-one tree approximates the optimal solution when $\gamma(x) = \lceil \log_2(x) \rceil$ within an additive constant of 1.

Corollary 1. *If $(G, \gamma, (w_i))$ is an instance of CSS with $\gamma(x) = \lceil \log_2(x) \rceil$ and n leaf weights are equal, then the depth-one tree approximates the optimal cost tree within an additive constant of 1.*

6 Final Thoughts

While we have positive results for CSS when the initial hierarchy is constraint-free, and negative results when it is a DAG, we have yet to characterize the problem for directed trees. We have looked at specific tree topologies, like binary trees and complete r-ary trees, but even in these cases, have not characterized the optimal solutions for the linear degree cost. Additionally, we have not explored probability distributions other than arbitrary and uniform. For example, what happens with a geometric or Zipfian distribution? Finally, we are interested in CSS in dynamic environments. For example, on a website, page statistics are constantly changing. Is there a way to dynamically update the optimal tree in time proportional to the height of the tree?

References

1. Perkowitz, M., Etzioni, O.: Towards adaptive web sites: Conceptual framework and case study. Artificial Intelligence **118** (2000) 245–275
2. Bose, P., Czyzowicz, J., Gasienicz, L., Kranakis, E., Krizanc, D., Pelc, A., Martin, M.V.: Strategies for hotlink assignments. In Lee, D.T., Teng, S.H., eds.: Algorithms and Computation, 11th International Conference. Volume 1969 of Lecture Notes in Computer Science., Springer (2000) 23–34

3. Czyzowicz, J., Kranakis, E., Krizanc, D., Pelc, A., Martin, M.V.: Evaluation of hotlink assignment heuristics for improving web access. In: Second International Conference on Internet Computing, CSREA Press (2001) 793–799
4. Czyzowicz, J., Kranakis, E., Krizanc, D., Pelc, A., Martin, M.V.: Enhancing hyperlink structure for improving web performance. Journal of Web Engineering **1** (2003) 93–127
5. Karp, R.: Minimum-redundancy coding for the discrete noiseless channel. IRE Transactions on Information Theory **IT** (1961) 27–29
6. Colin, M.J., Kenyon, C., Young, N.E.: Huffman coding with unequal letter costs. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM Press (2002) 785–791
7. Golin, M.J., Rote, G.: A dynamic programming algorithm for constructing optimal prefix-free codes with unequal letter costs. IEEE Transactions on Information Theory **44** (1998) 1770–1781
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York, New York (1979)
9. Heeringa, B., Adler, M.: Optimal website design with the constrained subtree selection problem. Technical Report 04-09, University of Massachusetts Amherst (2004)
10. Gormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. 2 edn. The MIT Press/McGraw-Hill Book Company (2001)

Simple Permutations Mix Well

Shlomo Hoory, Avner Magen, Steven Myers, and Charles Rackoff

Department of Computer Science
University of Toronto
{shlomoh, avner, myers, rackoff}@cs.toronto.edu

Abstract. We study the random composition of a small family of $O(n^3)$ simple permutations on $\{0,1\}^n$. Specifically we ask what is the number of compositions needed to achieve a permutation that is close to k-wise independent. We improve on a result of Gowers [8] and show that up to a polylogarithmic factor, n^3k^3 compositions of random permutations from this family suffice. We further show that the result applies to the stronger notion of k-wise independence against adaptive adversaries. This question is essentially about the rapid mixing of the random walk on a certain graph, and we approach it using a new technique to construct canonical paths. We also show that if we are willing to use a much larger family of simple permutations then we can guaranty closeness to k-wise independence with fewer compositions and fewer random bits.

1 Introduction

A question that occurs naturally in cryptography is how well the composition of permutations drawn from a simple distribution resembles a random permutation. Although this type of construction is a common source of security for cryptographic primitives such as DES and its successors, the mathematical justification for it is troubling, and is one of the motivations of this work.

A source or a distribution is *pseudo-random* if it is random in the computational sense, namely no computationally bounded machine can distinguish it from a truly random one. Another natural and well studied measure for randomness, although lacking an obvious linkage to computational considerations, is the notion of almost k-*wise independence*. When the distribution is over permutations, which is always the case in this paper, this means that the distribution induced by applying a random permutation in the family to any k distinct elements is almost the same as the distribution when applying a truly random permutation to this set, i.e. the uniform distribution over the sets of k distinct elements. We can now form the following question: consider a small set of simple permutations over $\{0,1\}^n$, that we call *basic permutations,* and compose T random elements from this set to get a permutation f. Is the distribution over f pseudo-random? How close is this distribution to k-wise independent? The second question is the focus of this paper; specifically we bound from above the number of times T we need to compose the basic permutations in order to generate a family of permutations that is a good approximation to a k-wise independent family of permutations.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 770–781, 2004.

In [8] T. Gowers studied this question. The basic permutations he considered were the ones that fix all but three coordinates of the n-bit strings. This set is of size $\Theta(n^3)$ which is a tiny fraction of the $(2^n)!$ possible permutations. [1]. Gowers shows that by composing [2] $\tilde{O}(n^3k(n^2+k)(n^3+k))$ randomly chosen basic permutations, one constructs a distribution over permutations that is close to k-wise independent, provided a certain divisibility condition regarding n and k applies. In this work we show that by using this set of permutations it is sufficient to compose $\tilde{O}(n^3k^3)$ basic permutations to get the above guarantee, and there is no need for n and k to satisfy divisibility conditions. Further, we demonstrate that a more restricted set than Gowers' (although still of order n^3) suffices for this result.

Our question concerning the minimal number of compositions of basic permutations, T, needed to achieve a distribution that is close to k-wise independent can be restated in terms of random walks. Namely, we are interested in the mixing time of a random walk on the graph whose vertices are k-tuples of distinct n-bit strings, and whose edges are induced by the obvious operation of basic permutations on the vertices. The mixing time of this graph is exactly that minimal number of compositions T that we seek. We bound the mixing time by means of the *canonical path method.* In the course of our proof, we improve upon Gowers' upper bound of the diameter of this graph from $O(kn^2)$ to the tight bound of $\tilde{O}(kn)$. In order to estimate the conductance of our graph we present a new and general way to construct the *canonical paths* in a wide class of graphs (Cayley and Schreier graphs) that provides an "algorithmic" method to demonstrate mixing. We believe that this technique (essentially Lemma 1) can be useful in showing rapid mixing for other Markov chains.

We also consider the notion of *strong closeness to k-wise independence* which is a strengthening of the standard closeness to k-wise independence: given a permutation f drawn from a particular distribution, how well can a computationally unbounded machine that is allowed to query f adaptively k times, distinguish it from a truly random permutation[3]. We show in Proposition 1 a connection between being strongly ϵ-close to k-wise independent and mixing using relative point-wise distance (as opposed to the standard total variation distance).

To define our graph we need to define our basic permutations. We look at permutations that change just one bit of their input, by XORing it with a function on few other bits. Formally, for $0 < w < n$ we define the set of permutations $\mathcal{F}_w = \{f_{i,J,h}\}$ where $i \in [n]$, $J = \{j_1, \ldots, j_w\}$ is a size w index set disjoint from i, and h is a boolean function on $\{0,1\}^w$. The permutation $f_{i,J,h}$ maps $(x_1, \ldots, x_n) \in \{0,1\}^n$ to $(x_1, \ldots, x_{i-1}, x_i \oplus h(x_{j_1}, \ldots, x_{j_w}), x_{i+1}, \ldots, x_n)$. Clearly $\mathcal{F}_2$ is a subset of Gowers' set of basic permutations. Also note that $|\mathcal{F}_w| = n \cdot \binom{n-1}{w} \cdot 2^{2^w}$. We now state our main results.

---

[1] Observe that there are $n(n-1)(n-2)$ choices for the three distinct coordinates, and 8! permutations of $\{0,1\}^3$.

[2] The tilde in the notation $\tilde{O}$ suppresses polylogarithmic factors in n and k.

[3] For perfect k-wise independent permutation distributions, the notions are equivalent, but there are simple examples that separate the notions when we consider distributions that are ϵ-close to k-wise independent.

Theorem 1. *Let $k = O(2^{n/4})$, and let T be the minimal number of random compositions of independent and uniformly distributed permutations from $\mathcal{F}_2$ needed to generate a permutation which is ϵ-close to k-wise independent. Then $T = \tilde{O}(n^2k^2 \cdot (\log(1/\epsilon) + nk))$.*

If, instead of striving to achieve the minimal set of basic permutations, we want to use as few random bits as possible to get k-wise independence, then it is interesting to check other candidate sets of basic-permutations. Note, the number of random bits used is simply the $\log_2$ of the number of basic permutations times the number of times we compose them. Therefore, Theorem 1 tells us $\tilde{O}(n^3k^3)$ random bits suffice to get the desired property. It follows from the next theorem that one can use as little as $\tilde{O}(n^2k^2)$ such bits, when instead of $\mathcal{F}_2$ we take $\mathcal{F}_w$ where $w = 2\log k + \log n + O(\log\log n)$.

Theorem 2. *Let T be the minimal number of random compositions of independent and uniformly distributed permutations from $\mathcal{F}_w$ for $w \geq 2\log k + \log n + \log\log n + 8$, needed to generate a permutation which is ϵ-close to k-wise independent. Then $T = O(\log(1/\epsilon) \cdot n \cdot \log n \cdot (\log k + \log n))$.*

The proof of Theorem 2 is omitted from this extended abstract. As will be shown in Section 3, Theorems 1 and 2 apply for strong ϵ-closeness to k-wise independence. Also, it is interesting to note that [6] implies that both Gowers' and our sets of basic permutations generate all even permutations of $\{0,1\}^n$.

2 Preliminaries

Let f be a random permutation on some base set X. Denote by $X^{(k)}$ the set of all k-tuples of distinct elements from X. We say that f is ϵ-close to k-wise independent if for every $(x_1, \ldots, x_k) \in X^{(k)}$ the distribution of $(f(x_1), \ldots, f(x_k))$ is ϵ-close to the uniform distribution on $X^{(k)}$. We measure the distance between two probability distributions p, q by the total variation distance, defined by

$$d(p,q) = \frac{1}{2}||p-q||_1 = \frac{1}{2}\sum_{\omega}|p(\omega) - q(\omega)| = \max_A \sum_{\omega \in A} p(\omega) - q(\omega).$$

We sometimes replace p or q by a random variable having this distribution.

Assume a group H is acting on a set X and let S be a subset of H closed under inversion. Then the *Schreier graph* $G = \mathrm{sc}(S, X)$ is defined by $V(G) = X$ and $E(G) = \{(x, xs) : x \in X, s \in S\}$. Also, for a sequence $\omega = (s_1, \ldots, s_l) \in S^l$ we denote $x\omega = xs_1 \cdots s_l$. We will sometimes refer by xw also to the walk $x, (xs_1), \ldots, (xs_1 \cdots s_l)$.

The *random walk* $(X_0, X_1, \ldots)$ associated with a d-regular graph G is defined by the transition matrix $P_{vu} = \Pr(X_{i+1} = u | X_i = v)$ which is $1/d$ if $(v,u) \in E(G)$ and zero otherwise. The uniform distribution π is stationary for this Markov process. If G is connected and not bipartite, we know that given any initial distribution of X_0, the distribution of X_t tends to the uniform distribution. We define the mixing time of G as $\tau(\epsilon) = \max_{v \in V(G)} \min\{t : d(P^{(t)}(v, \cdot), \pi) < \epsilon\}$,

where $P^{(t)}(v,.)$ is the probability distribution of X_t given that $X_0 = v$. It is not hard to prove (see for example Lemma 20 in [1]) that

$$\tau(2^{-l-1}) \leq l \cdot \tau(1/4). \tag{1}$$

3 Strong Closeness to k-Wise Independence

Let $\mathcal{F}$ be a distribution of permutations $f : \Omega \to \Omega$, where $\Omega = \{0,1\}^n$. We can think of k-wise independence in the following terms: a computationally unbounded adversary chooses a tuple $\boldsymbol{x} \in \Omega^{(k)}$; it is then given either a random permutation p from the set $\mathcal{P}$ of all permutations $\Omega \to \Omega$, or a random permutation $f \in \mathcal{F}$; and is asked to distinguish the two distributions. To say that a distribution is k-wise independent (resp. ϵ-close to k-wise independent) is to say that the distinguishing probability is 0 (resp. less than ϵ). One can strengthen the notion of adversary to permit it to adaptively choose its queries. Such an adversary is a tuple $A = (\alpha_1, ..., \alpha_k, \overline{A})$, where $\alpha_i : \Omega^{(i-1)} \to \Omega$ and $\overline{A} : \Omega^{(k)} \to \{0,1\}$. The adversary iterates through k steps, where in the ith step it requests $q_i = \alpha_i(r_1, .., r_{i-1})$ and gets response $r_i = f(q_i)$. After the kth step it outputs $\overline{A}(r_1, ..., r_k)$. We denote by A^f the output of A after it has interacted with f.

In the case of (strict) k-wise independence it can be shown that such a strengthening cannot help the adversary distinguish the distributions, and this is not the case for ϵ-close to k-wise independence[4]. This motivates the following definition: a distribution $\mathcal{F}$ is said to be *strongly* ϵ-close to k-wise independent if it is ϵ-close to k-wise independent against *adaptive* adversaries. This definition has previously been considered in the context of cryptography on several occasions [12,10].

We state (without proof) a proposition that shows that any distribution of functions that is ϵ-close to k-wise independent using the relative pointwise distance measure is also *strongly* ϵ-close to k-wise independent using the total variation distance measure. The *relative pointwise distance,* or d_{rp}, between probability distributions p and q over Ω is: $d_{rp}(p,q) = \max_{\omega \in \Omega} |p(\omega) - q(\omega)|/p(\omega)$.

Proposition 1. *For $\boldsymbol{y} \in \Omega^{(k)}$ let $P(\boldsymbol{y})$ and $F(\boldsymbol{y})$ be the distributions induced by $p(\boldsymbol{y})$ and $f(\boldsymbol{y})$ respectively, for randomly chosen $p \in \mathcal{P}$ and $f \in \mathcal{F}$. Let $d_{rp}(\mathcal{P}, \mathcal{F}) = \max_{\boldsymbol{y} \in \Omega^{(k)}} d_{rp}(P(\boldsymbol{y}), F(\boldsymbol{y}))$. Then, for every adaptive adversary A:*

$$\Pr_{p \in \mathcal{P}}[A^p = 1] - \Pr_{f \in \mathcal{F}}[A^f = 1] \leq d_{rp}(\mathcal{P}, \mathcal{F}).$$

Since $\epsilon \cdot 2^{-nk}$-closeness to k-wise independence in terms of the total variation distance implies ϵ-closeness in terms of the relative pointwise distance, it follows from Proposition 1 that if ϵ is not extremely small then Theorem 1 also applies in the case of strong ϵ-closeness to k-wise independence. A recent result by Maurer and Pietrzak [10] shows that if we double the number of compositions

[4] Consider the uniform distribution over the set $\mathcal{F}$ of permutations $f : \Omega \to \Omega$ where $f = f^{-1}$, and the case $k = 2$.

needed to get a distribution that is ϵ-close to k-wise independent, then we get a distribution that is strong $2\epsilon(1 + \log(1/\epsilon))$-close to k-wise independent. This implies that both Theorem 1 and Theorem 2 hold for strong ϵ-close to k-wise independence for any value of ϵ.

4 Proof of Theorem 1

A central parameter in the analysis of the mixing-time of a random walk on a d-regular graph G is the *conductance* of a graph $\Phi(G)$ which is defined as follows.

$$\Phi(G) = \min_{A \subseteq V(G), |A| \leq |V|/2} \frac{|E(A, \overline{A})|}{d \cdot |A|}, \tag{2}$$

where $\overline{A} = V(G) \setminus A$, and $E(A, \overline{A}) = \{(v, u) \in E(G) : v \in A \text{ and } u \notin A\}$. A fundamental result relating conductance and rate of mixing is the following. We say that a random walk is lazy if for some constant $\delta > 0$ we have $\Pr[X_{t+1} = v | X_t = v] \geq \delta$ for all $v \in V(G)$.

Theorem 3. *(Jerrum and Sinclair [11]) If the random walk on G is lazy then* $\tau(\epsilon) = O\left(\Phi^{-2} \cdot \log(|V(G)|/\epsilon)\right)$.

One method to derive a lower bound on the conductance is the canonical path technique of Jerrum and Sinclair [9]. This technique essentially states the following simple min-cut $\geq$ max-flow fact. If one thinks of a d-regular graph as a network where edges have capacity Λ and it is possible to transfer one unit of flow between every pair of vertices, then the conductance of the graph is at least $\frac{|V|}{2d\Lambda}$. This is simply because the the capacity of the cut between A and $\overline{A}$ must accommodate a total flow of size $|A||\overline{A}|$ and so $\Lambda \cdot |E(A, \overline{A})| \geq |A||\overline{A}| \geq |A| \frac{|V|}{2}$. Therefore, in order to bound the conductance one can show a valid flow that requires a small value of Λ (this is sometimes referred to as the *load* of the flow).

Being a Schreier graph, our graph lends itself to a special type of flow that we now introduce. Let $G = sc(S, X)$ and consider a probability distribution μ over finite sequences of elements from S. For any $x \in X$, the distribution μ induces a distribution μ_x of the end points of paths starting at x, where the probability of the path $x\overline{s}$ is $\mu(\overline{s})$. Suppose first that for every $x \in X$, μ_x is the uniform distribution over X. Then for each $x, y \in X$ we can assign a flow of $\mu(\overline{s})$ to the path $x\overline{s}$ (from x) and a flow of the same value to the path $y\overline{s}$ (towards y). Owing to the assumption that μ_x is uniform, this is a valid flow from x to y (satisfies conservation of matter). The load on an edge $e = (u, us)$ is $2 \cdot \sum_y \sum_x \eta_{x,u,s} = 2 \cdot |X| \cdot \sum_x \eta_{x,u,s}$, with $\eta_{x,u,s}$ being the expected number of occurrences of e in a random path $x\omega$ where ω has distribution μ. The factor of 2 follows since the first and second halves contribute the same load to e.

More generally, assume that that for all x the distribution μ_x is ϵ-close to uniform in total variation distance . Then for any vertex z, we compare $\mu_y(z)$ and $\mu_x(z)$. We define the same flow from x to y as in the uniform case except that to get a valid flow we multiply the flow in the paths from x to z by

$\min(1, \mu_y(z)/\mu_x(z))$, and the flow from z to y by $\min(1, \mu_x(z)/\mu_y(z))$. This will result in a flow of at least $1-2\epsilon$ from x to y. By scaling back to 1, we get a valid flow, where the load of e is bounded by $(1-2\epsilon)^{-1} \cdot 2 \cdot |X| \cdot \sum_x \eta_{x,u,s}$.

Lemma 1. *If μ, μ_x, Λ are as above, and for every $x \in X$ the distribution μ_x is ϵ-close to uniform, then $\Lambda \leq (1-2\epsilon)^{-1} \cdot |X| \cdot 2L$, where $L = \max_{s \in S} L(s)$ and $L(s)$ is the expected number of occurrences of s in a random sequence with distribution μ.*

Proof. Since the load on the edge $e = (u, us)$ is bounded by $(1-2\epsilon)^{-1} \cdot 2 \cdot |X| \cdot \sum_x \eta_{x,u,s}$, it is sufficient to show that $\sum_x \eta_{x,u,s} \leq L$ for every u, s. Indeed, consider the process where we start from a randomly chosen $x \in X$ and follow a random sequence from μ. Notice that $\frac{1}{|X|} \cdot \sum_x \eta_{x,u,s}$ is the expected number of times we hit e in this process. Since the initial vertex is chosen according to the stationary distribution, the distribution of the vertex we traverse in the l'th move is always uniform. Hence $\sum_x \eta_{x,u,s} = |X| \cdot \frac{1}{|X|} \cdot L(s) \leq L$. ∎

From Lemma 1 we get the following lower bound on the conductance:

$$\Phi \geq \frac{|X|}{2d\Lambda} \geq \frac{1-2\epsilon}{4 \cdot |S| \cdot L}. \tag{3}$$

Note 1. It is possible to improve (3) by a factor of two, if, rather than constructing a valid flow, we assign flow $\mu(\bar{s})$ to the path $x\bar{s}$ for all x and $\bar{s}$. It is easy to see that for every vertex subset $Y \subset X$, the flow from Y to its complement $\bar{Y}$ is at least $|Y| \cdot (|\bar{Y}|/|X| - \epsilon)$.

Denote by $L(G, \epsilon)$ the minimal L achievable by any distribution on sequences of elements from S such that for every $x \in X$ the distribution of $x\omega$ is ϵ-close to the uniform distribution. Theorem 3 together with inequality (3) gives

Corollary 1. $\tau(\epsilon) \leq O(|S|^2 \cdot L(G, 1/4)^2 \cdot \log(|X|/\epsilon))$ *whenever the random walk is lazy.*

In order to prove that the composition of elements from $\mathcal{F}_2$ approaches k-wise independence quickly we construct the Schreier graph $G_{k,n} = \mathrm{sc}(\mathcal{F}_2, \Omega^{(k)})$, where $\Omega^{(k)}$ is the set of k-tuple with k distinct elements from the base set $\Omega = \{0,1\}^n$. It is convenient to think of $\Omega^{(k)}$ as the set of k by n matrices with distinct rows. A basic permutation acts on $\Omega^{(k)}$ by acting on each of the rows.

Our goal now is to define a distribution over sequences of permutations from $\mathcal{F}_2$ with the following properties: (i) the application of a random sequence to any $x \in \Omega^{(k)}$ yields a matrix that is almost uniformly distributed over $\Omega^{(k)}$ and (ii) the *load* (the expected number of occurrences) is small for every $s \in \mathcal{F}_2$. More specifically, we want to show that

$$L(G_{k,n}, 1/4) = \tilde{O}(\frac{kn}{|\mathcal{F}_2|}) = \tilde{O}(\frac{k}{n^2}), \tag{4}$$

which by Corollary 1 proves Theorem 1.

For brevity, we denote $L(G_{k,n}, \epsilon)$ by $L(k, n, \epsilon)$. Note that by (1) we have

$$L(k, n, \epsilon) \leq \lceil \log(1/\epsilon) \rceil \cdot L(k, n, 1/4). \tag{5}$$

The rest of this section is devoted to proving (4). Here is an overview. A naive way to get a random sequence that will turn any matrix to random would be to go over all its entries one by one and to flip each entry independently with probability half. Such an approach ignores the fact that whenever we apply an element $s \in \mathcal{F}_2$ to the matrix we act simultaneously on all the rows, so independence is highly unlikely. But what if we apply what we call a *characteristic permutation,* which is a permutation that flips a bit exactly when a specified set of a other bits have the values $\nu = (\nu_1, \nu_2, \ldots, \nu_a)$? Intuitively most of the rows will not be affected by such a permutation. This leads to a way of approximating the naive scheme. Here is how. First notice that since characteristic permutations do not belong to $\mathcal{F}_2$ we need to compose elements of $\mathcal{F}_2$ in order to get them. To this end we use a theorem of Barenco et al. [2] that any such permutation is a composition of $O(a)$ elements from $\mathcal{F}_2$.[5] We start our sequence by a relatively short sequence of elements from $\mathcal{F}_2$ achieving almost 2-wise independence. Therefore, taking a set of a columns for sufficiently large a, we get that w.h.p. any string ν of length a cannot occur in more than one row, and we get our required handle on the rows. This is done in Lemma 3. Unfortunately the value of a needed turns out to be big, making the length of the resulting sequences long. This issue is overcome in Lemmas 4 that bootstraps off of Lemma 3.

Next, with the benefit of foresight, we point out the following.

Observation 1 *In Lemmas 2, 3 and 4 we will present distributions μ on sequences of elements from $\mathcal{F}_2$ where certain $s \in \mathcal{F}_2$ may receive an undue load, as these permutations operate on specified indices (columns) of interest. This is easy to overcome when we simply imagine the lemmas applying over all possible permutations of the indices. Therefore, since there will always be three indices of interest, we get that the load on any particular permutation in $\mathcal{F}_2$ is at most $O(\lambda/n^3)$ where λ is the maximal length of the sequences of μ.*

We turn to the lemmas establishing bounds on the needed load of the sequence distributions.

Lemma 2.

$$L(2, n, 1/4) = O(\log n/n^2).$$

Proof. Using Observation 1, it is enough to give a distribution over length $O(n \log n)$ sequences of permutations from $\mathcal{F}_2$ that take any initial $2 \times n$ matrix with two distinct rows to a matrix 1/4-close to a uniformly distributed matrix with two distinct rows. The mixing time of the graph $G_{2,n}$ is $O(n \log n)$; this is a rather immediate corollary of the same bound holds for the so called "Aldous cube" [4] (proof omitted). Therefore the uniform distribution over length $O(n \log n)$ sequences of permutations from $\mathcal{F}_2$ has the desired property. ∎

[5] This is an improvement over a previous result of Cleve [5] that gives an $O(a^2)$ bound.

We now get to two lemmas that embed "algorithms" in the construction of the stochastic sequences.

Lemma 3. *If* $k \leq 2^{(n-8)/4}$ *then*

$$L(k, n, 1/4) \leq L(2, n, 1/8k^2) + O(k^2 \cdot \log k/n^2).$$

Proof. Let a be the integer satisfying $8k^2 \leq 2^a < 16k^2$. We construct a random sequence ω by starting with ω_1 which is an $L(2, n, 1/8k^2)$ sequence. Given any $k \times n$ matrix x we know that the rows of $x\omega_1$ are $1/8k^2$-close to 2-wise independent. Let X be the expected number of pairs of rows of $x\omega_1$ that coincide in their first a coordinates. Then

$$E[X] \leq \binom{k}{2} \cdot (2^{-a} + \frac{1}{8k^2}) \leq \frac{k^2}{2} \cdot \frac{2}{8k^2} = \frac{1}{8}.$$

Therefore the probability that the first a columns of $x\omega_1$ to have distinct rows is at least $\frac{7}{8}$. After ω_1 we perform the following procedure ω_2:

For $i = a+1, \ldots, n$
  For $\alpha \in \{0,1\}^a$
    with probability $\frac{1}{2}$ do $g_{i,\alpha}$,

where $g_{i,\alpha} : \{0,1\}^n \to \{0,1\}^n$ is the permutation that flips the i'th coordinate iff $(x_1, \ldots, x_a)$ is equal to α. The permutation $g_{i,\alpha}$ is implemented as a concatenation of $O(a) = O(\log k)$ basic permutations using the result of Barenco et al. [2], section VIIA. If the first a columns of $x\omega_1$ have distinct rows then the last $n-a$ columns of $x\omega_1\omega_2$ have a uniform distribution.

We end the sequence ω by performing ω_3

For $i = 1, \ldots, a$
  For $\alpha \in \{0,1\}^a$
    with probability $\frac{1}{2}$ do $h_{i,\alpha}$,

where $h_{i,\alpha}$ flips the i'th coordinate iff the last a coordinates are equal to α. As before $h_{i,\alpha}$ is implemented as a concatenation of $O(\log k)$ basic permutations. After applying ω_3, the first a columns have uniform distribution if all the rows of the last a columns of $x\omega_1\omega_2$ are distinct. Given that the first condition holds, i.e. that all the rows of the first a columns of $x\omega_1$ are distinct, the second condition fails with probability bounded by $\frac{k^2}{2} \cdot 2^{-a} \leq \frac{1}{16}$. Therefore, for $\omega = \omega_1\omega_2\omega_3$, we have that with probability at least $1 - \frac{1}{8} - \frac{1}{16}$ the distribution of $x\omega$ is uniform. Therefore the distribution of $x\omega$ is $\frac{3}{16}$-close to uniform. [6] The only condition we have to check is that the first and last a columns are disjoint, i.e. $2a \leq n$. This is guaranteed if $16k^2 \leq 2^{n/2}$.

The length of the sequence $\omega_2\omega_3$ is bounded by $O(k^2 n \log k)$. By Observation 1 the load is $O(k^2 n \log k/n^3)$. ∎

[6] This argument actually proves that $x\omega$ is $\frac{3}{16}$-close to the uniform distribution on Ω^k. However, the uniform distribution on Ω^k and $\Omega^{(k)}$ are $o(1)$-close.

Lemma 4. *If* $k \leq 2^{(n-16)/4}$ *then*

$$L(k,n,1/4) \leq L(b,n,\epsilon) + O\left(\frac{k}{n^2} \cdot \log k\right),$$

where $b = 3 + \lceil \frac{1}{3} \log k \rceil$ *and* $\epsilon = \frac{1}{32} \cdot k^{-b-1}$.

Proof. Let $a = 3 + \lceil \log k \rceil$. Since $4a \leq n$, we can partition the columns of the matrix to four sets $C_1, \ldots, C_4$ of size a and the leftover C.

We start ω by ω_1 which is an $L(b,n,\epsilon)$ sequence. For $p \in \{1,2,3,4\}$, $i \notin C_p$ and $\alpha \in \{0,1\}^a$ let $g_{i,\alpha,C_p} : \{0,1\}^n \to \{0,1\}^n$ be the permutation that flips the ith bit of x if the restriction of x to C_p is equal to α. As before we implement $g_{i,\alpha,p}$ as the concatenation of $O(\log k)$ basic permutations.

Let ω_2 be the following randomized procedure.

For $i \in [n] \setminus (C_1 \cup C_2)$,
  For $\alpha \in \{0,1\}^a$ with probability $\frac{1}{2}$ do g_{i,α,C_1}
  For $\beta \in \{0,1\}^a$ with probability $\frac{1}{2}$ do g_{i,β,C_2}.

We argue that for any $k \times n$ matrix x the distribution of the columns $[n] \setminus (C_1 \cup C_2)$ of $x\omega_1\omega_2$ is uniform with high probability.

Given the matrix $x\omega_1$ we build a bipartite multi-graph H over the sets V_1 and V_2 where $V_1 = V_2 = \{0,1\}^a$, and where H has k edges, one for each row of the matrix. The edge associated with a row of $x\omega_1$ is between $s_1 \in V_1$ and $s_2 \in V_2$ if its restriction to C_p is s_p for $p = 1,2$. For perspective we relate our schema here to the previous lemma. There, we essentially looked at a block of the size of $C_1 \cup C_2$ and went over all possible values to this number of bits, hence a range which is of size k^2 instead of k here. In terms of H, the claim there was that w.h.p. it does not contain any multi edges and for that we needed the pairwise independence of the rows. Here we need a stronger property, namely that H is cycle-free, and this will be possible to show using the stronger condition on $x\omega_1$, namely that it is an almost b-wise independent matrix.

We first argue if H is cycle free then the distribution of the columns not in $C_1 \cup C_2$ of $x\omega_1\omega_2$ is uniform. Fix i to be the column of interest. Let $r_{\alpha,i}$ and $s_{\beta,i}$ be the $2 \cdot 2^a$ random bits used to generate the part of ω_2 that is responsible for column i. For any edge $e = (\alpha, \beta)$

$$(x\omega_1\omega_2)_{e,i} = (x\omega_1)_{e,i} \oplus r_{\alpha,i} \oplus s_{\beta,i}. \tag{6}$$

For a given $x\omega_1$, the probability that the i'th column has a certain value v is proportional to the number of solutions in the variables $r_{\alpha,i}, s_{\beta,i}$ for the linear system (6). This number is independent of the specific value of v if the linear system has full rank. It is easy to see that the matrix defining this system is exactly the incidence matrix of H. We now only need to use the well known fact that this matrix has a full rank iff H does not contain a cycle.

We now turn to show that H is cycle free w.h.p.. Recall that H is a random bipartite graph with k edges that is close to b-wise independent in the sense that any event in which at most b edges are involved happens with almost the

same probability it happens in a completely random graph with k edges. Let E_l be the expected number of l-cycles for $2 \leq l < b$ in the graph. We have $k \cdot (k-1) \cdots (k-l+1)$ ways to choose the l edges of the cycle. The edges connect properly with probability at most $2^{-al} + \epsilon$. Thus

$$E_l \leq k^l \cdot (2^{-al} + \epsilon) \leq 8^{-l} + \frac{1}{32} \cdot k^{l-b-1}.$$

For cycles longer than b we cannot use the b-wise independence in the same way. Instead we bound the probability of having b edges creating a path to get a bound on the expected number of all cycles of length $\geq b$ which is $k^b \cdot (2^{-a(b-1)} + \epsilon) \leq k \cdot 8^{-(b-1)} + \frac{1}{32} \leq \frac{3}{64}$. Therefore the total number of cycles is bounded by

$$\frac{3}{64} + \sum_{l=2}^{b-1} 8^{-l} + \frac{1}{32} \cdot k^{l-b-1} \leq \frac{1}{8},$$

for a sufficiently large k.

As in the proof of lemma 3, we continue with the sequence ω_3, which uses the two column sets C_3 and C_4 to change the columns C_1 and C_2 to the uniform distribution. Assume that H had no cycle and therefore that ω_2 succeeded. Then the graph H' formed by the C_3 and C_4 columns of $x\omega_1\omega_2$ has uniform distribution over all bipartite graphs with vertex sets of size 2^a and k edges. Therefore the probability that H' has a cycle is certainly smaller than $\frac{1}{8}$, and we get that with probability at least $\frac{3}{4}$ the matrix $x\omega_1\omega_2\omega_3$ is uniform. Therefore its distance from the uniform distribution is $\leq \frac{1}{4}$ (see footnote 6). Yet again, by Observation 1 we conclude the contribution of ω_2, ω_3 to L is $O(k \log k/n^2)$ and we are done. ∎

Proof. (of Theorem 1)

We combine lemmas 2, 3 and 4 with inequality (5) to get

$$\begin{aligned} L(k,n,1/4) &\leq O(L(b,n,1/4)\log^2 k + \frac{k}{n^2}\log k) \\ &\leq O(L(2,n,1/8b^2)\log^2 k + \frac{b^2}{n^2}\log b \log^2 k + \frac{k}{n^2}\log k) \\ &\leq O((\log n \log^2 k \log\log k + \log^4 k \log\log k + k\log k)/n^2) \\ &\leq O((\log n \log^2 k \log\log k + k\log k)/n^2). \end{aligned}$$

By corollary 1, the mixing time of $G_{n,k}$ is bounded by

$$\tau(\epsilon) = O(n^6 \cdot L(k,n,1/4)^2 \cdot (nk + \log(1/\epsilon)) = \tilde{O}(n^2k^2 \cdot (nk + \log(1/\epsilon))).$$ ∎

5 More on Motivation, Cryptography, and Possible Extensions

A principle motivation for this work is the philosophy behind the construction of "permutation generators" such as DES and its successors. The goal is that

the permutation generated from a random key should look like a randomly chosen permutation, when examined by a computationally limited adversary; this property is called "pseudo-randomness". The idea used by DES is to start with a very simple function generator G, and then compose functions independently and randomly chosen from G. (Actually, in order to keep the key short, the functions are not chosen independently, but we will ignore this for now.) Because the adversary is allowed much more time than was taken to compute the function, (almost) k-wise independence is neither necessary nor sufficient in order to achieve pseudo-randomness. Regardless, k-wise independence is a very natural measure of randomness, and one appealing question is what can (almost) k-wise independence tell us about pseudo-randomness.

Here is one possible conjecture. Let us assume that the generator G we start with is such that each possible permutation is "simple", where "simple" might mean that each output bit depends on a constant number of input bits. Say that T compositions from G suffice to achieve almost 4-wise independence. Then we conjecture that T compositions suffice to achieve pseudo-randomness. Of course proving this would show P different from NP, so this is unlikely. The conjecture is, however, susceptible to disproof.

Why do we choose "4-wise" in the above conjecture? For one thing, it is not hard to find examples where 3-wise is not good enough. Also, there is a theorem – proven using the classification of finite simple groups – that any collection of permutations satisfying 4-transitivity will, when composed together, eventually yield at least the alternating group [3,7].

References

1. D. Aldous and J. A. Fill. Reversible markov chains and random walks on graphs. http://stat-www.berkeley.edu/users/aldous/RWG/book.html.
2. A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A,* 52(5):3457–3467, 1995.
3. P. J. Cameron. *Permutation groups,* volume 45 of *London Mathematical Society Student Texts.* Cambridge University Press, Cambridge, 1999.
4. F. R. K. Chung and R. L. Graham. Stratified random walks on the n-cube. *Random Structures Algorithms,* 11(3):199–222, 1997.
5. R. Cleve. Complexity theoretic issues concerning block ciphers related to D.E.S. In A. Menezes and S. Vanstone, editors, *Advances in Cryptology - CRYPTO '90 Proceedings, LNCS,* volume 537, pages 530–544. Springer-Verlag, 1990.
6. D. Coppersmith and E. Grossman. Generators for certain alternating groups with applications to cryptography. *SIAM J. Appl. Math.,* 29(4):624–627, 1975.
7. J. D. Dixon and B. Mortimer. *Permutation groups,* volume 163 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, 1996.
8. W. T. Gowers. An almost m-wise independent random permutation of the cube. *Combin. Probab. Comput.,* 5(2):119–130, 1996.
9. M. Jerrum. *Counting, sampling and integrating: algorithms and complexity.* Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 2003.

10. U. Maurer and K. Pietrzak. Composition of random systems: When two weak make one strong. In *The First Theory of Cryptography Conference,* 2004.
11. A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Inform. and Comput.,* 82(1):93–133, 1989.
12. S. Vaudenay. Adaptive-attack norm for decorrelation and super-pseudorandomness. In *Selected Areas of Cryptography, LNCS,* volume 1758, pages 49–61. Springer-Verlag, 1999.

Closest Pair Problems in Very High Dimensions*

Piotr Indyk[1], Moshe Lewenstein[2], Ohad Lipsky[2], and Ely Porat[2]

[1] MIT
indyk@theory.lcs.mit.edu
[2] Bar-Ilan University
{moshe,lipsky,porately}@cs.biu.ac.il

Abstract. The problem of finding the closest pair among a collection of points in $\Re^d$ is a well-known problem. There are better-than-naive-solutions for constant d and approximate solutions in general. We propose the first better-than-naive-solutions for the problem for large d. In particular, we present algorithms for the metrics L_1 and L_∞ with running times of $O(n^{(\omega+3)/2})$ and $O(n^{(\omega+3)/2} \log D)$ respectively, where $O(n^\omega)$ is the running time of matrix multiplication and D is the diameter of the points.

1 Introduction

The problem of finding the closest pair in a given set of points from $\Re^d$ is a fundamental and well-studied problem in computational geometry. It has been known for at least two decades [Rab76] that closest pair can be solved in $O(n)$ time (for any L_p norm) as long as the dimension d is constant. However, the complexity of this problem becomes much more mysterious when d is large. Shamos and Bentley [SB76] conjectured that, for $d = n$, the problem can be solved in $O(n^2 \log n)$ time. So far (dis)-proving this conjecture remains elusive.

In recent years, several *approximate* algorithms were designed for the high-dimensional closest pair problem [Kle97,IM98,BOR99]. In particular, the work [IM98] (cf. [Ind01]) provided a c-approximate algorithm for this problem with running time $O(dn^{1+1/c} \log^{O(1)} n)$ in L_1 and L_2 norm; recently, the time was reduced to $O(dn^{1+1/c} \log n)$ for the L_2 norm [DIIM03]. However, for the exact case of $c = 1$, the aforementioned algorithms do not provide any improvement over a naive $O(dn^2)$-time bound.

In this paper we investigate the complexity of the closest pair and related problems in very high dimensions. For simplicity, we focus on the case $d = n$. It has been known (although not published anywhere) that for the L_2 norm, the complexity of closest pair is closely related to the complexity of matrix multiplication. Specifically, one matrix multiplication enables computing dot-products of all pairs of input vectors, from which the $O(n^2)$ pairwise distances can be easily extracted in $O(n^2)$ time.

* This work was supported by the German-Israel Foundation (G.I.F.) young scientists program research grant agreement no. 2055-1168.6/2002.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 782–792, 2004.

Here we show that the relationship between closest pair and matrix multiplication is not limited to the particular L_2 case. In particular, assuming that matrix multiplication can be solved in $O(n^\omega)$ time, we show:

- $O(n^{(\omega+3)/2})$-time algorithm for computing all pairwise distances according to the L_1 norm; thus the closest pair (or the diameter) can be found within the same time.
- $O(n^{(\omega+3)/2} \log D)$-time algorithm for finding closest pair under the L_∞ norm where D is the diameter of the points.
- $O(n^\omega \log n/\epsilon)$-time algorithm for $(1+\epsilon)$-approximate computing of all pairwise distances according to the L_∞ norm.

Thus, our results provide the first non-trivial bounds for the *exact* high-dimensional closest pair problem in L_1 and L_∞.

The methods we show here can be used to solve similar problems in pattern matching. For details see [LP04].

Our Techniques. Our algorithms exploit the parallel between matrix multiplication and convolution of strings [Mut95], see also [MP94]. This allows us to use the "high/low-frequency" pattern matching technique of Abrahamson [Abr87] in the context of matrix multiplication. We note that while the "high/low-frequency" pattern matching technique has been used in many applications, e.g. [Kos90,DGM94,AF95,AAL,APL04], this seems to be its first application in conjunction with matrix multiplication.

2 Preliminaries and Problem Definition

We assume the RAM model of computation, which allows arithmetic on $\log N$ bit numbers in $O(1)$ time, where N is the order of the maximum problem size. Let $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ be two points in Σ^n. Then the Minkowsky L_p distance metric between x and y is defined as: $L_p(x,y) = \sqrt[p]{\Sigma_{i=1}^n |x_i - y_i|^p}$. The L_1, L_2 and L_∞ metrics are the well-known Manhattan, Euclidean and Max metrics, respectively. We deal with numerical alphabets, so $\Sigma \subset \Re$.

Notation: For a matrix X we define $X^{(b)}$ by $X^{(b)}[i,j] = (X[i,j])^b$, and X^T by $X^T[i,j] = X[j,i]$. We use ω to denote the exponent of n which is needed in order to multiply 2 matrices of size $n \times n$.

2.1 Problem Definitions

- The *all pairs L_p distance* problem is defined as follows:
  Input: Set of n points $S = \{p_1, p_2, \ldots, p_n\}$ where $p_i \in \Sigma^n$.
  Output: Matrix $M[1 \ldots n, 1 \ldots n]$, where for every $i,j \in \{1, \ldots, n\}$, $M[i,j] = \|p_i - p_j\|_p$.
- The *approximate all pairs L_p distance* is defined as follows:
  Input: Set of n points $S = \{p_1, p_2, \ldots, p_n\}$, accuracy parameter $0 < \epsilon < 1$.
  Output: Matrix $\widehat{M}[1 \ldots n, 1 \ldots n]$, s.t. $M[i,j] \leq \widehat{M}[i,j] \leq (1+\epsilon)M[i,j]$, where $M[i,j] = \|p_i - p_j\|_p$.

– The *closest pair under* L_p is defined as follows:
Input: Set of n points $S = \{p_1, p_2, \ldots, p_n\}$ where $p_i \in \Sigma^n$.
Output: All Pairs (k, l), $1 \leq k, l \leq n$ where $\|p_k - p_l\|_p \leq \|p_i - p_j\|$ for all $i, j \in \{1, \ldots, n\}$.

For all these problems, we let $A = \{a_{i,j}\}$ denote an $n \times n$ matrix whose rows are $p_1, \ldots, p_n$, i.e. $a_{i,j} = p_i[j]$.

3 All Pairs Under L_1 Distance

Since the dimension, d, of each of the points is n it is straightforward to solve the problem of all pairs under the L_1 distance in $O(n^3)$ by directly computing the distance for each of the pairs.

In this section we provide faster algorithms for this problem. Let $A = \{a_{i,j}\}$ denote a $n \times n$ matrix whose rows are $p_1, \ldots, p_n$, i.e. $a_{i,j} = p_i[j]$. We begin by showing an $O(Sn^\omega)$ time algorithm for the problem, where $S = \max_{1 \leq j \leq n}\{\#\text{different numbers in column } j \text{ of } A\}$. We then present a different algorithm that merges the ideas of the first algorithm with an algorithm that uses the method of dividing by frequency. The running time of the second algorithm is $O(n^{(\omega+3)/2})$. Note that if $S \leq n^{\frac{3-\omega}{2}}$ then the running time of the first algorithm is better.

A central idea of both algorithms is a "divide and conquer" approach on the computation of the matrix multiplication by observing that elements of column k of A are relevant only to elements of this same column. More precisely, it is necessary to compute the matrix M, where $M[i, j] = \sum_{k=1}^{n} |a_{i,k} - a_{j,k}|$. Now, observe that for matrices X, Y,

$$X \times Y^T = \begin{pmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & & \vdots \\ x_{n,1} & \cdots & x_{n,n} \end{pmatrix} \times \begin{pmatrix} y_{1,1} & \cdots & y_{n,1} \\ \vdots & & \vdots \\ y_{1,n} & \cdots & y_{n,n} \end{pmatrix} =$$

$$\sum_{k=1}^{n} \begin{pmatrix} 0 \cdots 0 & x_{1,k} & 0 \cdots 0 \\ & & \\ \vdots & \vdots & \vdots \\ & & \\ 0 \cdots 0 & x_{n,k} & 0 \cdots 0 \end{pmatrix} \times \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \\ y_{1,k} & \cdots & y_{n,k} \\ 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$$

From the above equation, we see that matrix multiplication of two matrices X, Y^T involves multiplying of elements of column k of X only with elements of row k of Y^T (or column k of Y). This leads us to the following algorithm which works in $O(Sn^\omega)$, where $S = \max_{j=1}^{n} |\Sigma_j|$, $\Sigma_j = \{a_{i,j} \mid 1 \leq i \leq n\}$.

3.1 $O(Sn^{\omega})$ Algorithm

The following algorithm uses the divide and conquer approach described above and computes the necessary info for each column and value separately.

Algorithm I

1. Let $B_1, \ldots, B_S, C_1, \ldots, C_S, D_1, \ldots, D_S, E_1, \ldots, E_S$ be matrices of size $n \times n$.
2. For all $j = 1, 2, \ldots, n$
   a) For every $\sigma_\alpha^j \in \Sigma_j = \{\sigma_1^j, \sigma_2^j, \ldots, \sigma_{|\Sigma_j|}^j\}$
      i. For every $i = 1, 2, \ldots, n$
         A. $B_\alpha[i,j] \leftarrow 1$, if $a_{i,j} = \sigma_\alpha^j$ or $B_\alpha[i,j] \leftarrow 0$ otherwise.
         B. $C_\alpha[i,j] \leftarrow \begin{cases} a_{i,j} & a_{i,j} > \sigma_\alpha^j \\ 0 & a_{i,j} = \sigma_\alpha^j \\ -a_{i,j} & a_{i,j} < \sigma_\alpha^j \end{cases}$
         C. $D_\alpha[i,j] \leftarrow a_{i,j}$ if $a_{i,j} = \sigma_\alpha^j$ or $D_\alpha[i,j] \leftarrow 0$ otherwise.
         D. $E_\alpha[i,j] \leftarrow \begin{cases} -1 & a_{i,j} > \sigma_\alpha^j \\ 0 & a_{i,j} = \sigma_\alpha^j \\ 1 & a_{i,j} < \sigma_\alpha^j \end{cases}$
3. For every $1 \le \alpha \le S$ compute $M_\alpha = B_\alpha \times C_\alpha^T + D_\alpha \times E_\alpha^T$.
4. Compute $M = \sum_{\alpha=1}^{S} M_\alpha$.

To show that the algorithm is correct it is sufficient to show the following.

Lemma 1. *The matrix M output by the algorithm satisfies:* $M[i,j] = \sum_{k=1}^{n} |a_{i,k} - a_{j,k}|$.

Proof.

$$M[i,j] = \sum_{\alpha=1}^{S} M_\alpha[i,j] = \sum_{\alpha=1}^{S} (B_\alpha \times C_\alpha^T[i,j] + D_\alpha \times E_\alpha^T[i,j]) =$$

$$\sum_{\alpha=1}^{S} \left(\sum_{k=1}^{n} (B_\alpha[i,k] C_\alpha[j,k] + D_\alpha[i,k] E_\alpha[j,k])\right) \quad (**)$$

Now from the way $B_\alpha, C_\alpha, D_\alpha$ and E_α are assigned we know that

$$B_\alpha[i,k] C_\alpha[j,k] + D_\alpha[i,k] E_\alpha[j,k] = \begin{cases} 0 & a_{i,k} \neq \sigma_\alpha^k \\ -a_{j,k} + a_{i,k} & \sigma_\alpha^k = a_{i,k} > a_{j,k} \\ a_{j,k} - a_{i,k} & \sigma_\alpha^k = a_{i,k} < a_{j,k} \end{cases}$$

It means that $B_\alpha[i,k] C_\alpha[j,k] + D_\alpha[i,k] E_\alpha[j,k] = \chi_=(a_{i,k}, \sigma_\alpha^k) |a_{i,k} - a_{j,k}|$ (where $\chi_=(x,y) = 1$ if $x = y$ and 0 otherwise). That leads to:
$(**) = \sum_{\alpha=1}^{S} (\sum_{k=1}^{n} \chi_=(a_{i,k}, \sigma_\alpha^k) |a_{i,k} - a_{j,k}|) = \sum_{k=1}^{n} (\sum_{\alpha=1}^{S} \chi_=(a_{i,k}, \sigma_\alpha^k) |a_{i,k} - a_{j,k}|) = \sum_{k=1}^{n} (|a_{i,k} - a_{j,k}| \sum_{\alpha=1}^{S} \chi_=(a_{i,k}, \sigma_\alpha^k)) = \sum_{k=1}^{n} |a_{i,k} - a_{j,k}|$
(since $a_{i,k}$ equals to exactly one of $\sigma_1^k, \ldots, \sigma_{|\Sigma_k|}^k$). □

Note that the same result can be achieved, and even be simpler, using only half of the matrix multiplications used in the proof. However, we use this form in order to make the proofs in the $O(n^{(\omega+3)/2})$ algorithm simpler.

Time: Our time complexity is derived from the maximal number of different numbers in each row of A and from the time needed to multiply two matrices of order n. The total time needed is $O(Sn^{\omega})$. For a constant S the time complexity is $O(n^{\omega})$, but for large S, e.g. $S = n$, it is worse than the naive algorithm, which runs in time $O(n^3)$.

3.2 Dividing Columns of A into Groups by Frequency of Numbers

In order to reduce the time needed in case that $S > n^{\frac{3-\omega}{2}}$ we apply a technique of grouping non-frequent alphabet elements together. The process, described in detail below, divides the elements that appear in each column j of A to frequent and non-frequent numbers, then it divides the non-frequent numbers into $O(k)$ intervals, in a manner that none of the intervals contain a frequent number and at most $2n/k$ elements of the non-frequent numbers are contained in each interval.

For each column j of A, $j = 1, 2, \ldots, n$ do the following:

1. Define $S_j = \{< a_{ij}, i > | 1 \leq i \leq n\}$.
2. Sort elements of S_j by a_{ij}.
   From now on, we refer to S_j as the sorted sequence $< s_1, l_1 >, < s_2, l_2 >, \cdots, < s_n, l_n >$, i.e. $s_1 \leq s_2 \leq \ldots \leq s_n$.
   We define a number, f, to be *frequent* if at least n/k elements in S_j have $s_i = f$. Otherwise, we say that it is *non-frequent.*
3. Divide the sequence S_j into continuous groups as follows:
   a) For each frequent number f all elements with $s_i = f$ will form a group F_f^j.
      Now, there are at most k groups of frequent numbers, with elements of non-frequent numbers scattered into, at most, $k+1$ intervals between them. Let $I_1, \ldots, I_r$ denote these intervals $(r \leq k+1)$.
   b) Divide the elements of each I_v into smaller interval-groups as follows:
      $t \leftarrow 1$
      While there are at least n/k elements in I_v that are not in groups:
      - Take the leftmost n/k elements in I_v that are not yet in any group and form a new group NF_t^j.
      - Let m be the maximum s_i s.t. $< s_i, l_i > \in NF_t^j$.
      - Add to NF_t^j all the elements with $s_i = m$ (there are at most n/k, since m is non-frequent).
      - $t \leftarrow t+1$

      Form a group NF_t^j of the, less than n/k, remaining elements in I_v.

At the end of this process we have for each row j at most k groups $F_{f_1}^j, F_{f_2}^j, \ldots$ because we have n elements, and the size of each F_f^j is at least n/k. In addition we have at most $k+1$ groups NF_t^j of size $< n/k$ (The groups of the "remainders"). The rest of the elements are spread over groups NF_t^j of size at least

n/k. Therefore, the total number of groups is $O(k)$. From the construction of the groups it is easy to see that the following properties are satisfied.

- All the groups are intervals.
- None of the groups intersect each other. Moreover, if $< x, y >$ is in one group and $< x', y' >$ in another then $x \neq x'$.
- Each group of non-frequent numbers contains at most $2n/k$ elements.

3.3 $O(n^{(\omega+3)/2})$ Algorithm

We now describe the second algorithm. The algorithm consists of two stages. In the first stage we divide the numbers of each column into frequent numbers and intervals of non-frequent numbers as described in section 3.2. Then we apply a procedure similar to the $O(Sn^{\omega})$ algorithm on the intervals. However, this will cause errors because we ignore the distance made by pairs of elements from the same interval. In the second stage we take care of correcting the errors caused by the first stage.

Algorithm Steps:
First, divide each column $j \in \{1, \ldots, n\}$ of A into groups of frequent elements $F^j_{f_1}, \ldots, F^j_{f_{e(j)}}$ $(e(j) \leq k)$, and groups of non-frequent elements $NF^j_1, \ldots, NF^j_{e'(j)}$ $(e'(j) \leq 2k+1)$, as described in section 3.2.
Define $\max(NF^j_c) = max\{x \mid < x, y > \in NF^j_c\}$, $\min(NF^j_c) = min\{x \mid < x, y > \in NF^j_c\}$, and $\min(F^j_f) = \max(F^j_f) = f$. Let $z = 3k+1$.

First stage

1. Let $B_1, \ldots, B_z, C_1, \ldots, C_z, D_1, \ldots, D_z, E_1, \ldots, E_z$ be matrices of size $n \times n$.
2. For all $j = 1, 2, \ldots, n$
   a) For every $G^j_\alpha \in \{G^j_1, \ldots, G^j_{e(j)+e'(j)}\} = \{F^j_{f_1}, \ldots, F^j_{f_{e(j)}}, NF^j_1, \ldots, NF^j_{e'(j)}\}$
      i. For every $i = 1, 2, \ldots, n$
         A. $B_\alpha[i,j] \leftarrow 1$, if $a_{i,j} \in G^j_\alpha$ or $B_\alpha[i,j] \leftarrow 0$ otherwise.
         B. $C_\alpha[i,j] \leftarrow \begin{cases} a_{i,j} & a_{i,j} > \max(G^j_\alpha) \\ 0 & a_{i,j} \in G^j_\alpha \\ -a_{i,j} & a_{i,j} < \min(G^j_\alpha) \end{cases}$
         C. $D_\alpha[i,j] \leftarrow a_{i,j}$ if $a_{i,j} \in G^j_\alpha$ or $D_\alpha[i,j] \leftarrow 0$ otherwise.
         D. $E_\alpha[i,j] \leftarrow \begin{cases} -1 & a_{i,j} > \max(G^j_\alpha) \\ 0 & a_{i,j} \in G^j_\alpha \\ 1 & a_{i,j} < \min(G^j_\alpha) \end{cases}$
3. For every $1 \leq \alpha \leq z$ compute $M_\alpha = B_\alpha \times C^T_\alpha + D_\alpha \times E^T_\alpha$.
4. Compute $M_{stage1} = \sum_{\alpha=1}^{z} M_\alpha$.

We now show the correctness of the algorithm. Beforehand, we define $\chi_{group}(x, y) = 1$ if x and y are in the same NF^k_α for some $\alpha \in \{1, \ldots, e'(k)\}$, and 0 otherwise.

Lemma 2. *The output of the first stage,* $M_{stage1}[i,j] = \sum_{k=1}^{n} |a_{i,k} - a_{j,k}| \overline{\chi_{group}(a_{i,k}, a_{j,k})}$,

Proof. The proof is similar to the proof given in section 3.1 (note, if x and y are both members of some F_α^k then it means that $x = y$). □

In the second stage we compute $M_{stage2}[i,j] = \sum_{k=1}^{n} |a_{i,k} - a_{j,k}| \chi_{group}(a_{i,k}, a_{j,k})$.

Second Stage: We compute M_{stage2} in a straightforward way.

For every $1 \le j \le n$
  For every $1 \le \alpha \le e'(j)$
    For every $< a, i_a >, < b, i_b > \in NF_\alpha^j$
      $M_{stage2}[i_a, i_b] \leftarrow M_{stage2}[i_a, i_b] + |a - b|$
      $M_{stage2}[i_b, i_a] \leftarrow M_{stage2}[i_b, i_a] + |a - b|$

Finally, the algorithm computes $M = M_{stage1} + M_{stage2}$, which yields $\sum_{k=1}^{n} |a_{i,k} - a_{j,k}| \overline{\chi_{group}(a_i k - a_j k)} + |a_{i,k} - a_{j,k}| \chi_{group}(a_{i,k}, a_{j,k}) = \sum_{k=1}^{n} |a_{i,k} - a_{j,k}|$ and the desired result is obtained.

Time: The first stage is implemented with $O(k)$ matrix multiplications. Therefore, the running time of the first stage is $O(kn^\omega)$. The second stage takes time $O(n \max_j (e'(j)) \max_{j,\alpha}^2 (|NF_\alpha^j|)) = O(nk(\frac{2n}{k} + 1)^2) = O(n^3/k)$. Choosing $k = n^{\frac{3-\omega}{2}}$ yields an $O(n^{(\omega+3)/2})$ time for this algorithm.

4 All Pairs Under L_2 Distance

For completeness, we present here an algorithm that solves the problem of *all pairs L_2 distance* in $O(n^\omega)$. The time complexity is derived from the time needed to multiply two matrices of size $n \times n$.

By definition we need to compute $M[i,j] = \sqrt{\sum_{k=1}^{n} |a_{i,k} - a_{j,k}|^2}$ for every $i, j \in \{1, \ldots, n\}$. We show how to compute the matrix $M^{(2)}$, where $M^{(2)}[i,j] = M[i,j]^2 = \sum_{k=1}^{n} |a_{i,k} - a_{j,k}|^2$ and in one linear time pass on this matrix we compute the desired output matrix, M.

Note that $$M[i,j]^2 = \sum_{k=1}^{n} |a_{i,k} - a_{j,k}|^2 = \underbrace{\sum_{k=1}^{n} a_{i,k}^2}_{(1)} + \underbrace{\sum_{k=1}^{n} a_{j,k}^2}_{(2)} - \underbrace{2 \sum_{k=1}^{n} a_{i,k} a_{j,k}}_{(3)}$$

We compute (1), (2) and (3) separately, each of them for all $i, j \in 1, \ldots, n$, getting three matrices of values M_1, M_2 and M_3, summing them together gives $M^{(2)}$.

Computing M_1 and M_2 is easily done in linear time. M_3 can be computed by matrix multiplication in $O(n^\omega)$ time by observing that $M_3 = -2A \times A^T$.

5 Closest Pair Under L_∞ Distance

We define the problem of *pairs with distance of at least* d *with* L_∞ to be: finding all pairs with a distance at least d. We solve this problem in $O(n^{(\omega+3)/2})$ using the technique of dividing the columns of A into frequent and non-frequent elements. Next, by running a binary search on d, we find the minimal d such that pairs of points with distance d exists in our set and, hence, solve the closest pair problem. (Note that there can be more than one pair of minimal distance). The time complexity is $O(n^{(\omega+3)/2} \log D)$, where $D = \max_{j=1}^{n}(\max_{i=1}^{n} a_{i,j} - \min_{i=1}^{n} a_{i,j})$ is the diameter.

5.1 Pairs with Distance at Least d with L_∞

Our objective is to compute the matrix M defined by $M[i,j] = \bigvee_{k=1}^{n} \chi_d(a_{i,k}, a_{j,k})$, where $\chi_d(x,y) = 1$ if $|x-y| \geq d$ and 0 otherwise.

First, use the method described in section 3.2 in order to divide the elements of every column j of A to, at most, $3k+1$ groups, $G^j = \{G_1^j, \ldots, G_{e(j)+e'(j)}^j\} = \{F_{f_1}^j, \ldots, F_{f_{e(j)}}^j, NF_1^j, \ldots, NF_{e'(j)}^j\}$. We can assume G^j is sorted by $\max(G_\alpha^j)$ (if not, sort it). Our algorithm contains two stages, in the first stage we seek for elements with distance $\geq d$ by their group, treating "non-sure" cases as distance $< d$ and in the second stage we check those "non-sure" cases one by one. Total time is $O(n^{(\omega+3)/2})$, independent of d.

Algorithm Outline:
Let $S_1, \ldots, S_{3k+1}, T_1, \ldots, T_{3k+1}, R_1, \ldots, R_{3k+1}$ be matrices of size $n \times n$.

First Stage

1. For every $j = 1, 2, \ldots, n$
   a) For every $G_\alpha^j \in G^j$
      i. For every $i = 1, 2, \ldots, n$
         $S_\alpha[i,j] \leftarrow 1$ if $< a_{i,j}, i > \in G_\alpha^j$ or 0 otherwise.
         $T_\alpha[i,j] \leftarrow \begin{cases} 1 \ (max(G_\alpha^j) + d \leq a_{i,j}) \vee (a_{i,j} \leq \min(G_\alpha^j) - d) \\ 0 \ otherwise \end{cases}$
      ii. Compute $R_\alpha \leftarrow S_\alpha \times T_\alpha$
   b) Compute R defined by $R[i,j] = \bigvee_{\alpha=1}^{3k+1} R_\alpha[i,j]$.

Second Stage

1. For every $j \in \{1, \ldots, n\}$
   For every $G_\alpha^j \in G^j$
   For every $< a, i_a > \in G_\alpha^j$
   For every $a_{i,j} \in [\min(G_\alpha^j) - d, \max(G_\alpha^j) - d] \cup [\min(G_\alpha^j) + d, \max(G_\alpha^j) + d]$
   a) $R[i_a, i] \leftarrow \chi_d(a_{i_a,j}, a_{i,j})$
   b) $R[i, i_a] \leftarrow \chi_d(a_{i_a,j}, a_{i,j})$

Algorithm Correctness

Lemma 3. *At the end of the first stage* $R[i,j] = \bigvee_{k=1}^{n}(\max(a_{i,k}) + d < a_{j,k} \vee a_{j,k} < \min(a_{i,k}) - d)$, *where* $\max(a_{i,k})$ *and* $\min(a_{i,k})$ *are* $\max(G_\alpha^k)$ *and* $\min(G_\alpha^k)$ *respectively for* $a_{i,k} \in G_\alpha^k$.

Proof. Directly from the way we defined $S_1, \ldots, S_{3k+1}, T_1, \ldots, T_{3k+1}$. In other words, instead of comparing each $a_{j,k}$ with $a_{i,k}$ we only checked if $a_{j,k}$ is at least d-bigger than the maximal value of $a_{i,k}$'s group or d-smaller from the minimal value of $a_{i,k}$'s group. Now, if $\min(a_{i,k}) - d < a_{j,k} < \max(a_{i,k}) + d$ we treated it in the first stage as $|a_{i,k} - a_{j,k}| < d$. We fix it in the second stage. If we were checking in the second stage each element from G_α^j with all the elements in the interval $[\min(G_\alpha^j) - d, \max(G_\alpha^j) + d]$ we are done. But, we checked for the intervals $[\min(G_\alpha^j) - d, max(G_\alpha^j) - d]$ and $[\min(G_\alpha^j) + d, max(G_\alpha^j) + d]$. Now, we have two cases:

1. If $d < \max(G_\alpha^j) - min(G_\alpha^j)$ then $\max(G_\alpha^j) - d > min(G_\alpha^j)$ and $max(G_\alpha^j) > min(G_\alpha^j) + d$. Therefore $[\min(G_\alpha^j) - d, \max(G_\alpha^j) + d] = [\min(G_\alpha^j) - d, max(G_\alpha^j) - d] \cup [\min(G_\alpha^j) + d, max(G_\alpha^j) + d]$.
2. If $d > \max(G_\alpha^j) - min(G_\alpha^j)$ then the elements in $[max(G_\alpha^j) - d, min(G_\alpha^j)] \cup [max(G_\alpha^j), min(G_\alpha^j) + d]$ are at distance less than d from all elements of G_α^j and, therefore, do not need to be checked. □

Time: The first stage has $O(k)$ matrix multiplications. Therefore the time is $O(kn^\omega)$. In the second stage we check all elements of each group G_α^j with all elements from the interval $[\min(G_\alpha^j) - d, max(G_\alpha^j) - d]$ ("$-d$" interval) and from $[\min(G_\alpha^j) + d, max(G_\alpha^j) + d]$ ("+d" interval). Now, since for every j the groups $G_1^j, \ldots, G_{e(j)+e'(j)}^j$ do not intersect, neither their "-d" intervals intersect nor their "+d" intervals intersect (it is possible, though that a "-d" interval of one group will intersect with a "+d" interval of another group. this fact and the fact that there are $O(n/k)$ elements in each G_α^j gives a running time of $O(\sum_{j=1}^{n} 2 \times O(n/k) \times n) = O(n^3/k)$. Choosing $k = n^{(3-\omega)/2}$ yields a total time of this algorithm $O(n^{(\omega+3)/2})$.

6 Approximate All Pairs Under L_∞ Distance

In this section we show how to approximate the values of $\|p_i - p_j\|_\infty$ for all $i, j \in \{1, \ldots, n\}$, up to a factor of $1+\varepsilon$ in $O(\frac{1}{\varepsilon} n^\omega \log n)$. First, we show that it is enough to compute $\|p_i - p_j\|_p$ for $p \geq \log n/\varepsilon$ in order to approximate $\|p_i - p_j\|_\infty$ up to a factor of $1+\varepsilon$. Second, we give an $O(pn^\omega)$ time algorithm to compute *all pairs* L_p *distance,* for even p's. Choosing an even p leads to the required results.

Lemma 4. *For every* $p \geq \log n/\varepsilon$, $v, u \in \Re^n$,

$$\|v - u\|_\infty \leq \|v - u\|_p \leq \|v - u\|_\infty (1 + \varepsilon)$$

Proof: It is clear that for any p it holds that $\|v-u\|_\infty \leq \|v-u\|_p$. We need to show that $\|v-u\|_p \leq \|v-u\|_\infty(1+\varepsilon)$. Let $v=(v_1,\ldots,v_n), u=(u_1,\ldots,u_n)$ and let $b=\max_{1\leq i\leq n}|v_i-u_i| = \|v-u\|_\infty$. Now, $\|v-u\|_p = \sqrt[p]{\sum_{i=1}^n |v_i-u_i|^p} \leq \sqrt[p]{\sum_{i=1}^n b^p} = \sqrt[p]{nb^p} = \sqrt[p]{n}b = \sqrt[p]{n}\|v-u\|_\infty$. We have left to show that $\sqrt[p]{n} \leq (1+\varepsilon)$ and we are done. If $p \geq \log n/\varepsilon$ then $\sqrt[p]{n} = n^{\frac{1}{p}} \leq n^{\frac{\varepsilon}{\log n}} = 2^\varepsilon \leq ((1+\varepsilon)^{\frac{1}{\varepsilon}})^\varepsilon = 1+\varepsilon$. □

6.1 All Pairs Under L_p Distance for Even p's

We use a technique similar to the one shown in section 4 to build this algorithm. As before, $A=\{a_{i,j}\}$ will denote a matrix whose rows are $p_1,\ldots,p_n$. By definition we need to compute $M[i,j] = \sqrt[p]{\sum_{k=1}^n |a_{i,k}-a_{j,k}|^p}$ for every $i,j \in \{1,\ldots,n\}$. Note that $M[i,j]^p = \sum_{k=1}^n |a_{i,k}-aj,k|^p = \sum_{k=1}^n (a_{i,k}-aj,k)^p$ since p is even. Now, $M[i,j]^p = \sum_{k=1}^n (a_{i,k}-a_{j,k})^p = \sum_{k=1}^n \sum_{r=0}^p (-1)^r \binom{p}{r} a_{i,k}^r a_{j,k}^{p-r} =$ $\sum_{r=0}^p \sum_{k=1}^n (-1)^r \binom{p}{r} a_{i,k}^r a_{j,k}^{p-r} = \sum_{r=0}^p (-1)^r \binom{p}{r} \sum_{k=1}^n a_{i,k}^r a_{j,k}^{p-r}$ (*).

The computation of M is done as follows:

1. Compute $A^{(r)}$ for every $0 \leq r \leq p$.
2. Compute $B_r = A^{(r)} \times (A^{(p-r)})^T$ for $0 \leq r \leq p$ (T denote transpose).
3. Compute $M = \sum_{r=0}^p (-1)^r \binom{p}{r} B_r$.
4. For every $i,j \in \{1,\cdots,n\}, M[i,j] \leftarrow \sqrt[p]{M[i,j]}$.

It is clear from equation (*) that the computation is correct.

Time: The time needed for the first and third steps of the computation is $O(pn^2)$ and the time needed for the second step is $O(pn^\omega)$. Therefore, the total time of the algorithm is $O(pn^\omega)$.

References

[Abr87] K. Abrahamson. Generalized string matching. *SIAM J. Computing,* 16(6):1039–1051, 1987.

[AAL] A. Amir, A. Apostolico, and M. Lewenstein. Inverse Pattern Matching. *J. of Algorithms,* 24(2):325-339, 1997.

[AF95] A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation,* 118(1):1-11, 1995.

[APL04] A. Amir, E. Porat and M. Lewenstein. Faster algorithms for string matching with k mismatches. *J. of Algorithms, special SODA 2000 issue,* to appear.

[BOR99] A. Borodin, R. Ostrovsky, and Y. Rabani. Subquadratic approximation algorithms for clustering problems in high dimensional spaces. *Proceedings of the Symposium on Theory of Computing,* 1999.

[DIIM03] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *Proc. of Symposium of Computational Geometry (SOCG),* 2004, to appear.

[DGM94] M. Dubiner, Z. Galil, and E. Magen. Faster tree pattern matching. *J. of the ACM,* 41(2):205-213, 1994.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proceedings of the Symposium on Theory of Computing,* 1998.

[Ind01] P. Indyk. *High-dimensional computational geometry.* Ph.D. Thesis, Department of Computer Science, Stanford University, 2001.

[Kos90] R. Kosaraju. Efficient tree pattern matching. *Proc. of Symosium on Foundation of Computer Science,* 1990.

[Kle97] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing,* 1997.

[LP04] O. Lipsky and E. Porat. Approximated Pattern Matching with the L_1 L_2 and L_∞ Metrics. Manuscript, submitted to *SODA 2004.*

[Mut95] S. Muthukrishnan. New results and open problems related to non-standard stringology. *CPM,* 1995.

[MP94] S. Muthukrishnan and K. Palem. Non-standard stringology: algorithms and complexity. *Proc. of the ACM Symposium on Theory of Computing,* 770-779, 1994.

[Rab76] M. O. Rabin. Probabilistic algorithms. *Algorithms and Complexity, J. F. Traub, editor. Academic Press,* pages 21–39, 1976.

[SB76] I. Shamos and J. Bentley. Divide-and-conquer in multidimensional space. *Proceedings of the Symposium on Theory of Computing,* pages 220–230, 1976.

Universality in Quantum Computation

Emmanuel Jeandel

LIP (UMR CNRS, ENS Lyon, INRIA, Univ. Claude Bernard Lyon 1),
École Normale Supérieure de Lyon,
46 allée d'Italie 69364 LYON cedex 07 FRANCE
Emmanuel.Jeandel@ens-lyon.fr

Abstract. We introduce several new definitions of universality for sets of quantum gates, and prove separation results for these definitions. In particular, we prove that realisability with ancillas is different from the classical notion of completeness. We give a polynomial time algorithm of independent interest which decides if a subgroup of a classical group $(\mathbf{SO}_n, \mathbf{SU}_n, \mathbf{SL}_n \ldots)$ is Zariski dense, thus solving the decision problem for the completeness. We also present partial methods for the realisability with ancillas.

Quantum circuits are the quantum analogues of boolean circuits. It is well known that all boolean functions can be obtained from circuits using only the two gates OR and NOT. The problem of finding which gates have this property was solved when Post gave in [1] a complete classification of all classes of boolean functions closed under arbitrary composition. Such a set of gates was also found [2] for reversible computation: There exists a set of reversible gates that can generate any reversible boolean operator.

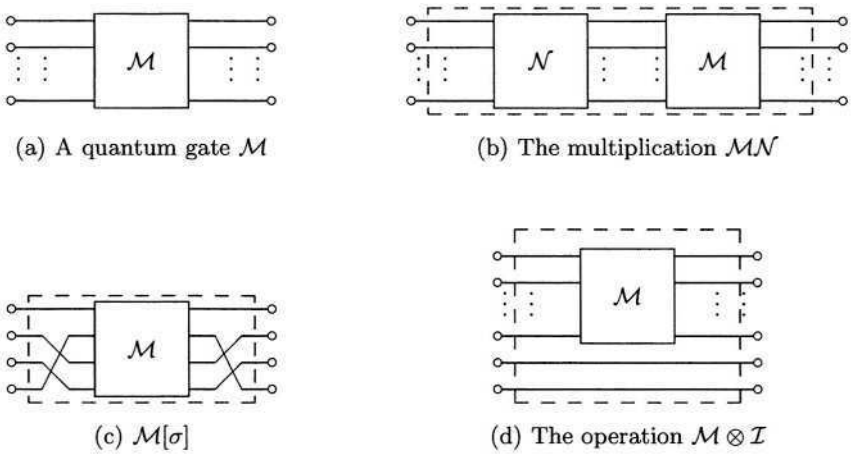
The existence of such sets for quantum computation was answered positively, and many sets of gates have been found [3,4,5,6]. However, a finite set of gates cannot produce all quantum gates and a notion of approximation is needed. Different notions of universality were then introduced and we define in this paper a wider notion of universality that extends many previous definitions while trying to clarify them. We give some separation results to show that all these notions are distinct, thus answering an open question in [7] about the use of ancillas. We then give an algorithm of independent interest that decides for many linear algebraic groups G whether a subgroup of G is dense, thereby providing a procedure to decide whether a set of quantum gates is complete.

1 Universality

1.1 Quantum Gates

A qubit is a vector of norm 1 in the Hilbert space $\mathbb{C}^2$. The canonical basis of this space is written $|\mathbf{0}\rangle, |\mathbf{1}\rangle$. A quantum state $|\phi\rangle$ is a vector of norm 1 in the space $(\mathbb{C}^2)^{\otimes n}$. We will denote by $\|\cdot\|$ the norm corresponding to the usual inner product. A basis of this space is given by $|\omega_1\rangle \otimes |\omega_2\rangle \cdots |\omega_n\rangle = |\omega\rangle$ where ω ranges

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 793–804, 2004.



(a) A quantum gate $\mathcal{M}$ (b) The multiplication $\mathcal{MN}$

(c) $\mathcal{M}[\sigma]$ (d) The operation $\mathcal{M} \otimes \mathcal{I}$

Fig. 1. Operations on quantum gates

over all words of length n over $\{0,1\}$. We will index the matrices by words of length n rather than integers from 1 to 2^n. With these notations, for any matrix $\mathcal{M}$, we have $\mathcal{M}|\omega\rangle = \sum_\nu \mathcal{M}_{\nu,\omega}|\nu\rangle$.

A quantum gate $\mathcal{M}$ over n qubits is an element of $\mathbf{U}_{2^n}$, that is an unitary matrix of dimension 2^n. It therefore maps quantum states to quantum states. We will represent it as a black box as in figure 1(a). The identity matrix will be written as $\mathcal{I}_{2^n}$ or $\mathcal{I}$ when there is no ambiguity about the dimension.

There are three natural operations on quantum gates:

- The multiplication $\mathcal{MN}$ of two quantum gates represents the composition of the gates, as $(\mathcal{MN})|\phi\rangle = \mathcal{M}(\mathcal{N}|\phi\rangle)$ (see fig. 1(b)).
- If σ is a permutation of $\{1 \dots n\}$, $\mathcal{M}[\sigma]$ represents the action of $\mathcal{M}$ where the qubits are swapped: for all words ω, ν, $\mathcal{M}[\sigma]_{\omega,\nu} = \mathcal{M}_{\sigma(\omega),\sigma(\nu)}$ where $\sigma(\omega) = \omega_{\sigma(1)} \cdots \omega_{\sigma(n)}$ (see fig. 1(c))
- The tensor product $\mathcal{M} \otimes \mathcal{I}_{2^{m-n}}$ represents the action of the matrix $\mathcal{M}$ over m qubits: As $(\mathcal{M} \otimes \mathcal{I})(|\phi\rangle \otimes |\psi\rangle) = \mathcal{M}|\phi\rangle \otimes |\psi\rangle$, this means intuitively that $\mathcal{M}$ acts on the n first qubits and leaves the others qubits unchanged (see fig. 1(d)). (Note that not all quantum states can be decomposed into the form $|\phi\rangle \otimes |\psi\rangle$; those that cannot be decomposed are called entangled states.)

All these operations are natural when viewed as circuit manipulation, and pretty natural compared to the physical model. However, an overall phase shift is unimportant from the physical point of view: The state $|\phi\rangle$ has the same property as the state $\lambda|\phi\rangle$ for λ a scalar of norm 1, that is $\lambda \in \mathbf{U}_1$. All the quantum gates must then be seen as elements of $\mathbf{U}_{2^n}/\mathbf{U}_1$ rather than elements of $\mathbf{U}_{2^n}$.

1.2 Quantum Circuits and Universality

If $\mathcal{S}$ is a set of quantum gates, a quantum circuit over $\mathcal{S}$ is any gate obtained starting from gates in $\mathcal{S}$ and using the previous constructions. Let $G_p(\mathcal{S})$ be the set of quantum circuits over p qubits with gates in $\mathcal{S}$. It is easy to see that $G_p(\mathcal{S})$ is the semigroup generated by $2^p \times 2^p$ matrices of the form $(\mathcal{M} \times \mathcal{I})[\sigma]$ for $\mathcal{M} \in \mathcal{S}$ and $\sigma \in \mathfrak{S}_p$; $G_p(\mathcal{S})$ is then a finitely generated semigroup. If $\mathcal{S}$ is closed under inversion ($\mathcal{M}^{-1} \in \mathcal{S}$ for $\mathcal{M} \in \mathcal{S}$), which we will implicitly assume from now on, $G_p(\mathcal{S})$ is a finitely generated group.

If we can find an easy way to represent a set $\mathcal{S}$ of gates which, in some sense, succeed in producing any quantum circuit, then we can represent in a compact fashion the quantum circuit, by explaining how it is obtained from the gates in $\mathcal{S}$. However, any finite set of gates $\mathcal{S}$ cannot produce all quantum circuits (as we may obtain in this way only a countable set of quantum circuits), and some notion of approximation is needed.

The easiest notion of universality is the following: $\mathcal{S}$ is universal if for every gate $\mathcal{M}$, for every ϵ, we can approach $\mathcal{M}$ by a circuit $\mathcal{N}$ in $\mathcal{S}$ in the sense that $\|\mathcal{M} - \mathcal{N}\|_m < \epsilon$, with $\|\cdot\|_m$ some chosen norm over $\mathbf{M}_{2^n}(\mathbb{C})$. This notion is also called completeness. Many sets of complete gates have been discovered [4,5,6,8], and it was shown that, in some sense, almost all sets of gates over 2 qubits is complete [3,9]. Note that we are interested here only in the production of any quantum gate, not in the efficiency of the realizations. However, the Solovay-Kitaev theorem [7] states that any complete set of gates can approximate any quantum gate upto a precision ϵ using polylogarithmically many gates.

Many other notions of universality were defined. One may help the gate $\mathcal{N}$ by giving it some auxiliary qubits initialised to $\mathbf{0}$: We can ask that $\mathcal{N}(|\psi\rangle \otimes |\mathbf{0}\rangle)$ is near $\mathcal{M}(|\psi\rangle) \otimes |\mathbf{0}\rangle$ for all $|\psi\rangle$. This is called universality using ancillas.

We now give a general definition of universality

Definition 1 (Universality). *Let $\mathcal{S}$ be a set of quantum gates over n qubits.*

$\mathcal{S}$ is (k,p)-universal if for every ϵ, for every gate $\mathcal{M}$ over $k+n$ qubits, there exists $l \leq p$ and a quantum circuit $\mathcal{N}$ with gates in $\mathcal{S}$ over $k+n+l$ qubits such that for all state $|\phi\rangle$ over $k+n$ qubits, $\|(\mathcal{M}|\phi\rangle) \otimes |\mathbf{0}\cdots\mathbf{0}\rangle - \mathcal{N}|\phi\rangle|\mathbf{0}\cdots\mathbf{0}\rangle\| \leq \epsilon$.

The notion of (k,∞)-universality is the most general: The number of auxiliary qubits to be used strongly depends on the precision we want to achieve. The notion of completeness coincides in this context with (0,0)-universality. We will now state the condition in a topological way. Let $P_{i,p}$ be the function that associates to any $2^p \times 2^p$ matrix the $2^i \times 2^i$ matrix in its upper left corner.

Proposition 1 *Let $\mathcal{S}$ be a set of quantum gates over n qubits.*

$\mathcal{S}$ is (k,p)-universal if $P_{n+k,n+k+p}(G_{k+n+p}(\mathcal{S}))$ is dense in $\mathbf{U}_{2^{n+k}}/\mathbf{U}_1$.

$\mathcal{S}$ is (k,∞)-universal if $\bigcup_p P_{n+k,n+k+p}(G_{k+n+p}(\mathcal{S}))$ is dense in $\mathbf{U}_{2^{n+k}}/\mathbf{U}_1$.

We now give some basic properties of these classes.

Proposition 2 *If $k \leq k'$, a (k,p)-universal set is also (k',p)-universal.*

If $p \leq p'$, a (k,p)-universal set is also (k,p')-universal.

A (k,p)-universal set is also $(k-i,p+i)$-universal.

The first two properties axe elementary. The third one says that if we can find a circuit that works for any state, then in particular it will work for states of the form $|\psi \mathbf{0} \cdots \mathbf{0}\rangle$.

It is not clear at this point whether all these notions of universality are in fact distinct. It was in particular asked in [7] whether the use of ancillas can help the computation. We will prove in the following section the following results:

Theorem 3. *There exists a set of gates over $n+3$ qubits that is $(0,n)$-universal but not (k,p)-universal for $p < n$.*

There exists a set of gates over 6 *qubits that is* (3, 0)*-universal but not* (0, 0)*-universal.*

The second result is somewhat surprising: There exists a set of gates over 6 qubits that cannot approximate all quantum gates over 6 qubits but that can approximate any quantum gate over 9 qubits. The proofs rely on some combinatorial properties of permutations.

We will then establish some decision procedures for universality. We gave in [10] a general purpose algorithm that computes the closure of every finitely generated matrix group. This entailed the decidability of (k,p)-universality.

However, as (k,p)-universality is an easier problem, an algorithm with a better complexity can be found. We will prove

Theorem 4. *There exists an algorithm that works in polynomial time and that decides, given generators $\mathcal{X}_1 \dots \mathcal{X}_k$, whether the group generated by $\mathcal{X}_1 \dots \mathcal{X}_k$ is dense over $\mathbf{U}_n$.*

The input is given as a set of matrices over an algebraic number field. Note that the algorithm will be polynomial over its input, hence exponential over n when dealing with $2^n \times 2^n$ matrices (that is quantum gates over n qubits). However, it is likely to be used for a fixed value of n ($n = 2$ or 3). This gives a decision procedure for $(k,0)$-universality. We will give only a partial result for (k,p)-universality.

2 Separation Results for Universality

We prove in this section Theorem 3. While the results are stated for quantum gates, we would like to point out that they still hold for reversible computation.

2.1 On the Contribution of Ancillas

Theorem 5. *There exists a set of gates over $n+2$ qubits that is $(0,n)$-universal but not (k,p)-universal for $p < n$.*

Fix $\mathcal{A}_1 \dots \mathcal{A}_l$ any set of quantum gates over 2 qubits that is (0, 0)-universal (that is complete). Many such sets can be found in the literature, see for example [6], It is important to note that as quantum circuits over $\mathcal{A}_1 \dots \mathcal{A}_l$ can approach any

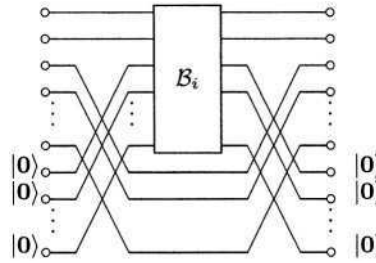


Fig. 2. The gate $\mathcal{M}_i$

quantum gate over 2 qubits, it is also the case for quantum gates over p qubits for $p \geq 2$.

We now construct the gates $\mathcal{B}_i$ over $2+n$ qubits in the following way: $\mathcal{B}_i$ acts as $\mathcal{A}_i$ only if the n last qubits are all $\mathbf{0}$. $\mathcal{B}_i$ is then defined over the canonical basis by $\mathcal{B}_i |\nu\rangle |\mathbf{0}\cdots\mathbf{0}\rangle = (\mathcal{A}_i |\nu\rangle) \otimes |\mathbf{0}\cdots\mathbf{0}\rangle$ and $\mathcal{B}_i |\nu\rangle |\omega\rangle = |\nu\rangle |\omega\rangle$ for all other words ω. Using the notation for controlled operators of [7], these gates would be written $\Lambda^n(A_i)$.

If we have n qubits with the value $|\mathbf{0}\rangle$, then we can apply $\mathcal{A}_i$. As $\{\mathcal{A}_i\}$ was chosen to be a complete set, we obtain:

Lemma 1. *The set* $\{\mathcal{B}_i\}$ *is* $(0, n)$*-universal.*

Proof. It suffices to show that we can obtain quantum circuits over $2n+2$ qubits representing the gates $\mathcal{A}_i \otimes \mathcal{I}$.

Let σ denote any permutation that permutes the sets $\{3 \ldots n+2\}$ and $\{n+3 \ldots 2n+2\}$, that is σ permutes the wires numbered from 3 to n with the last n wires (see figure 2). Define $\mathcal{M}_i = (\mathcal{B}_i \otimes \mathcal{I}_{2^n})[\sigma]$. Then it is easy to see that $\mathcal{M}_i(|\omega\rangle |\nu\rangle |\mathbf{0}\cdots\mathbf{0}\rangle) = \mathcal{A}_i |\omega\rangle \otimes (|\nu\rangle |\mathbf{0}\cdots\mathbf{0}\rangle) = (\mathcal{A}_i \otimes \mathcal{I})(|\omega\rangle |\nu\rangle) |\mathbf{0}\cdots\mathbf{0}\rangle$.

The previous formula is valid for the whole basis of $(\mathbb{C}^2)^{\otimes n+2}$, hence is true for all $|\phi\rangle$, that is $\mathcal{M}_i(|\phi\rangle |\mathbf{0}\cdots\mathbf{0}\rangle) = (\mathcal{A}_i \otimes \mathcal{I})(|\phi\rangle) \otimes |\mathbf{0}\cdots\mathbf{0}\rangle$. This completes the proof. □

We now prove the other statement. We use the fact that, for some entries, we can have less than n qubits set to $\mathbf{0}$, which prevents from applying $\mathcal{A}_i$.

Lemma 2. *The set* $\{\mathcal{B}_i\}$ *is not* (k, p)*-universal for* $p < n$.

Proof. Let E denote the subspace of $\mathbb{C}^{\otimes k+p+(n+2)}$ generated by all states $|\omega\rangle$ where the word ω has less than n bits to $\mathbf{0}$. It is easy to see that every gate of the form $(\mathcal{B}_i \otimes \mathcal{I})[\sigma]$ preserves E (as this is true for the generating set). Hence any quantum circuit over $\{\mathcal{B}_i\}$ preserves E.

However, there exists a quantum gate over $(n+2)+k$ qubits $\mathcal{X}$ such that $\mathcal{X} |\mathbf{1}\cdots\mathbf{1}\rangle = |\mathbf{0}\cdots\mathbf{0}\rangle$, so that $\mathcal{X} |\mathbf{1}\cdots\mathbf{1}\rangle \otimes |\mathbf{0}\cdots\mathbf{0}\rangle = |\mathbf{0}\cdots\mathbf{0}\rangle \otimes |\mathbf{0}\cdots\mathbf{0}\rangle \in E^{\perp}$. But every quantum circuit $\mathcal{N}$ over $\{\mathcal{B}_i\}$ satisfies $\mathcal{N}(|\mathbf{1}\cdots\mathbf{1}\rangle \otimes |\mathbf{0}\cdots\mathbf{0}\rangle) \in E$. The two states are orthogonal: $\| \mathcal{N}(|\mathbf{1}\cdots\mathbf{1}\rangle \otimes |\mathbf{0}\cdots\mathbf{0}\rangle) - \mathcal{X} |\mathbf{1}\cdots\mathbf{1}\rangle \otimes |\mathbf{0}\cdots\mathbf{0}\rangle \| = \sqrt{2}$.

No circuit in $\{\mathcal{B}_i\}$ can therefore approach $\mathcal{X}$ from less than $\sqrt{2}$. □

This proof strategy is essential and will be used in the next section.

2.2 Contribution of the Number of Qubits

We will prove here the last part of the theorem, which is the most intriguing part.

Again, we start from $\mathcal{A}_1 \dots \mathcal{A}_p$ a set of quantum gates over 2 qubits that is (0,0)-universal and such that $\mathcal{A}_i^2 = \mathcal{I}$. This is not a strong restriction as it is easy to write any matrix of $\mathbf{SU_4}$ as a product of finitely many matrices $\mathcal{X}_i$ such that $\mathcal{X}_i^2 = \mathcal{I}$.

$\{\mathcal{C}_i\}$ is then a set of gates over 6 qubits such that $\mathcal{C}_i$ acts as $\mathcal{A}_i$ only if the 4 last qubits are equal to $|\mathbf{0}\rangle$ or equal to $|\mathbf{1}\rangle$. $\mathcal{C}_i$ is then defined over the canonical basis by $\mathcal{C}_i |\nu\rangle |\omega\rangle = (\mathcal{A}_i |\nu\rangle) \otimes |\omega\rangle$ if ω is identically $\mathbf{0}$ or identically $\mathbf{1}$, and $\mathcal{C}_i |\nu\rangle |\omega\rangle = |\nu\rangle |\omega\rangle$ otherwise.

The following point is clear:

Lemma 3. $\{\mathcal{C}_i\}$ *is not* (0, 0)*-universal.*

Proof. We use the same idea as in the preceding proof. If we denote by E the vector space generated by all states $|\omega\rangle$ where ω has 3 letters set to $\mathbf{0}$ and 3 letters set to $\mathbf{1}$, then every quantum circuit over $\{\mathcal{C}_i\}$ preserves E (as this is true over the generating set).

Hence any quantum gate which sends some vector of E to a vector of $E^\perp$ cannot be approached by circuits over $\{\mathcal{C}_i\}$. □

However, it is not easy to see why circuits in $\{\mathcal{C}_i\}$ can approach any quantum gates over 9 qubits. This is our next lemma.

Lemma 4. $\{\mathcal{C}_i\}$ *is* (3, 0)*-universal.*

Proof. The idea is the following: Given 7 bits, we can find 4 bits to $\mathbf{0}$ or 4 bits to $\mathbf{1}$. As we do not know which bits are equal, we test all combinations.

Formally, for any choice S of 4 elements into $\{3 \dots 7\}$, we consider a permutation σ_S such that $\sigma_S(1) = 1, \sigma_S(2) = 2, \sigma_S(2+k) = S_k$ for $k \in \{1, 4\}$. Let $\mathcal{D}_i^S$ be the gate $(\mathcal{C}_i \otimes \mathcal{I})[\sigma_S]$. Intuitively, $\mathcal{D}_i^S$ connects the last 4 wires of the gate $\mathcal{C}_i$ to the wires S_1, S_2, S_3, S_4. Now consider the circuit $\mathcal{D}_i = \prod_S \mathcal{D}_i^S$.

Let ω be any word of length 7 which has more bits set to $\mathbf{1}$ than bits set to $\mathbf{0}$. If ω is identically $\mathbf{1}$ then any choice of 4 letters will give 4 bits to $\mathbf{1}$. There are $\binom{7}{4} = 35$ such choices. If ω has only 6 bits set to $\mathbf{1}$, then only $\binom{6}{4} = 15$ choices of 4 bits will give 4 bits to $\mathbf{1}$. We obtain respectively 5 and 1 choices that give 4 bits to $\mathbf{1}$ for the other cases. A similar result is obtained when ω has more bits to $\mathbf{0}$ than bits to $\mathbf{1}$. This means that on input $|\nu\rangle |\omega\rangle$, the circuit $\mathcal{D}_i$ will give the qubit $\mathcal{D}_i |\nu\rangle |\omega\rangle = (\mathcal{A}_i^k |\nu\rangle) \otimes |\omega\rangle$, with $k \in \{1, 5, 15, 35\}$. As $\mathcal{A}_i^2 = \mathcal{I}$, this gives $\mathcal{D}_i |\nu\rangle |\omega\rangle = (\mathcal{A}_i |\nu\rangle) \otimes |\omega\rangle = (\mathcal{A}_i \otimes \mathcal{I}) |\nu\rangle |\omega\rangle$. As this is true over a basis, this is true for the whole space, that is $\mathcal{D}_i = (\mathcal{A}_i \otimes \mathcal{I})$.

We have obtained quantum circuits representing the gates $(\mathcal{A}_i \otimes \mathcal{I})$. That establishes universality of the set $\{\mathcal{C}_i\}$. □

Remark. The key point is that $\binom{2^i+q}{2^i}$ is always odd. The whole argumentation can be reproduced to prove for all $i \geq 2$ the existence of a set of quantum gates over $2^i + 2$ qubits which is $(2^i - 1, 0)$-universal but not $(2^{i-1} - 2, 0)$-universal.

3 Deciding Density

We now give a polynomial time algorithm to decide whether a subgroup of a unitary group is dense. The algorithm is in fact more powerful as it deals with a larger class of groups, namely groups with a simple Lie algebra.

Note that for many classical groups, the adequate notion is the density for the Zariski topology of real spaces, rather than the density for the usual, Euclidean, topology. For compact matrix groups, these notions are equivalent [11].

The algorithm is then extended to the case of $\mathbf{SO}_4$, the only group in $\mathbf{SO}_n$ or $\mathbf{SU}_n$ that cannot be handled by the previous approach (as its Lie algebra is not simple). We are then able to give an algorithm for universality.

The main technical tool is the study of the conjugation action over these groups. It has been used for many others algorithms for matrix groups [12].

3.1 Lie Algebras

We give here some elements of the theory of Lie algebras that we will use later. We refer the reader to [11] for more precise definitions.

Let A be a subset of $\mathbb{R}^n$. A is Zariski-closed if there exists a polynomial p such that $(x_1 \dots x_n) \in A \iff p(x_1 \dots x_n) = 0$. The Zariski closure of a set A is the smallest Zariski-closed set that contains A.

A linear algebraic group is a subgroup of $\mathbf{GL}_n(\mathbb{R})$ (seen as a subset of $\mathbb{R}^{n^2}$) which is also Zariski-closed. Note that we will see subgroups of $\mathbf{GL}_n(\mathbb{C})$ as subgroups of $\mathbf{GL}_{2n}(\mathbb{R})$, as $\mathbf{U}_n$ is Zariski-closed considered as a real variety, but not as a complex one. Many classical groups (and all compact groups) are Zariski-closed: It is the case for $\mathbf{SL}_n, \mathbf{U}_n, \mathbf{O}_n, \mathbf{SO}_n \dots$.

The Lie algebra $\mathfrak{g}$ of a group G is the tangent space at the identity: $\mathcal{M}$ is in $\mathfrak{g}$ if there exists a differentiable curve τ in G, $\tau : [a, b] \mapsto G$ such that $\tau(0) = \mathcal{I}$ and its derivative $\tau'(0)$ is equal to $\mathcal{M}$ (Thus $\mathfrak{g}$ is a subset of $\mathbf{M}_n(\mathbb{R})$).

If G is an infinite linear algebraic group, $\mathfrak{g}$ is not trivial. Lie algebras determine in some way the groups. Indeed, if H is a subgroup of G such that G and H have the same Lie algebra and are connected, then $H = G$.

There is a natural structure over $\mathfrak{g}$: $\mathfrak{g}$ is a vector space closed under the bracket operator $[\mathcal{A}, \mathcal{B}] = \mathcal{AB} - \mathcal{BA}$. A sub-algebra of $\mathfrak{g}$ is a subspace closed by the bracket operator. A sub-algebra $\mathfrak{h}$ such that for all $\mathcal{M} \in \mathfrak{g}, \mathcal{N} \in \mathfrak{h}$, $[\mathcal{M}, \mathcal{N}] \in \mathfrak{h}$ is called an ideal of $\mathfrak{g}$. The Lie algebra of a closed normal subgroup of G is an ideal of $\mathfrak{g}$, hence ideals of Lie Algebras in some way measure the simplicity.

We will use the following property:

Proposition 6 *Let G be a linear algebraic group and $\mathfrak{g}$ its Lie algebra.*

Let $\mathfrak{h}$ be a sub-algebra of $\mathfrak{g}$. If $\mathfrak{h}$ is closed under conjugation by G (that is by the homomorphisms $\mathcal{X} \mapsto \mathcal{M}\mathcal{X}\mathcal{M}^{-1}$ for $\mathcal{M} \in G$), then $\mathfrak{h}$ is an ideal of $\mathfrak{g}$.

Proof. Let $\mathcal{M} \in \mathfrak{g}$, and τ such that $\tau(0) = \mathcal{I}, \tau'(0) = \mathcal{M}$. Let $\mathcal{X} \in \mathfrak{h}$. We know that the function $\theta : t \mapsto \tau(t)\mathcal{X}\tau(t)^{-1}$ takes value on $\mathfrak{h}$, hence $\theta'(0) \in \mathfrak{h}$. A straightforward computation gives $\theta'(t) = \tau'(t)\mathcal{X}\tau(t)^{-1} - \tau(t)\mathcal{X}\tau(t)^{-1}\tau'(t)\tau(t)^{-1}$ Hence, $\theta'(0) = \mathcal{M}\mathcal{X} - \mathcal{X}\mathcal{M} \in \mathfrak{h}$. □

A Lie algebra such that the bracket operator is identically zero is called abelian. A Lie algebra with no proper nonzero ideals and which is non-abelian is called simple. Simple Lie algebras will be the cornerstone of the proofs below.

3.2 Deciding Density for Connected Groups with a Simple Lie Algebra

We now introduce a standard construction that will simplify the statements of the theorem and of the proofs. Let ϕ be the function over $n \times n$ matrices given by $\phi(\mathcal{X}) = \mathcal{X} \otimes \left(\mathcal{X}^{-1}\right)^T$. If $\mathcal{X}$ is unitary, $\phi(\mathcal{X})$ is simply $\mathcal{X} \otimes \overline{\mathcal{X}}$ where $\overline{\mathcal{X}}$ is the conjugate of $\mathcal{X}$. For any matrix $\mathcal{Y}$, let $\mathrm{Vec}\,\mathcal{Y}$ be the column vector consisting of the juxtaposition of all columns of $\mathcal{Y}$. The following lemma is clear [13]:

Lemma 5. $\phi(\mathcal{X})\,\mathrm{Vec}\,\mathcal{Y} = \mathrm{Vec}\,\mathcal{X}^{-1}\mathcal{Y}\mathcal{X}$.

Hence, the operator ϕ captures the notion of conjugation in a vector space. For any group G, let env G be the enveloping algebra of G, that is the linear closure over $\mathbb{R}$ of all matrices of G.

Theorem 7. *Let $G \subseteq \mathbf{GL}_n(\mathbb{R})$ be a (Zariski) closed connected group with a simple Lie algebra. Then a subgroup H of G is (Zariski) dense in G if and only if H is infinite and* $\mathrm{env}\,\phi(H) = \mathrm{env}\,\phi(G)$.

Remark. The requirement on H to be infinite is necessary. In $\mathbf{SO}_3$, the group of isometries of the icosahedron is finite but satisfies $\mathrm{env}\,\phi(H) = \mathrm{env}\,\phi(\mathbf{SO}_3)$.

Proof. Suppose H is dense in G. Obviously, H is infinite. Furthermore $\mathrm{env}\,\phi(H)$ is a vector space, and is hence Zariski-closed. It must therefore contain the Zariski-closure of $\phi(H)$, that is $\phi(G)$. The equality $\mathrm{env}\,\phi(H) = \mathrm{env}\,\phi(G)$ follows.

Conversely, denote by Γ the Zariski-closure of H. As H is infinite, Γ is of non-zero dimension, hence its Lie algebra γ is not trivial. Now, $\mathrm{env}\,\phi(H) = \mathrm{env}\,\phi(G)$ entails that γ is stable by conjugation by G. Indeed, as γ is stable by conjugation by H, $\mathrm{Vec}\,\gamma$ is stable by multiplication by $\phi(h)$ for any $h \in H$ using lemma 5. By linearity, $\mathrm{Vec}\,\gamma$ is then stable by multiplication by any element of $\mathrm{env}\,\phi(H)$, hence by multiplication by any $\phi(g), g \in G$, as $\phi(G) \subseteq \mathrm{env}\,\phi(H)$. By lemma 5, this means that γ is stable by conjugation by G.

Then γ is an ideal of $\mathfrak{g}$, the Lie algebra of G, by Proposition 6. γ is not trivial, and $\mathfrak{g}$ is simple, hence $\gamma = \mathfrak{g}$, and $\Gamma = G$. □

Remark. The hypotheses of the theorem can be loosened by requiring that the Lie algebra of G has no proper nonzero ideals. The hypothesis of simplicity also implies that the Lie algebra is non-abelian but we do not use this fact here. Hence, the theorem is also true for $G = \mathbf{SU}_1, G = \mathbf{SO}_2 \ldots$

Many classical groups have a simple Lie algebra ($\mathbf{SO}_n(\mathbb{R})$ for $n \notin \{1, 2, 4\}$, $\mathbf{SU}_n$ for $n \geq 2, \ldots$) and this theorem directly provides an algorithm for these groups.

Theorem 8. *There is a polynomial-time algorithm which given a finitely generated subgroup of* $\mathbf{SU}_n$ *decides if* H *is dense over* $\mathbf{SU}_n$. *The algorithm checks if* H *is infinite then computes the enveloping algebra of* $\phi(H)$ *and compare it with the enveloping algebra of* $\phi(G)$.

There are many polynomial time algorithms to decide whether a group is finite or not [14], and computing the enveloping algebra can also be done in polynomial time, hence providing the announced complexity.

3.3 Deciding Density for $\mathbf{SO}_4(\mathbb{R})$

If the Lie algebra $\mathfrak{g}$ of G is not simple, the algorithm will not work: The Lie algebra $\mathfrak{h}$ might be a proper ideal of $\mathfrak{g}$ and we cannot directly deduce to the equality $\mathfrak{g} = \mathfrak{h}$.

This is the case for $G = \mathbf{SO}_4(\mathbb{R})$. First set

$$\sigma_1(a,b,c,d) = \begin{pmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{pmatrix} \quad \sigma_2(a,b,c,d) = \begin{pmatrix} a & b & c & -d \\ -b & a & -d & c \\ -c & d & a & -b \\ -d & -c & b & a \end{pmatrix}$$

G is not simple and contains two infinite closed normal subgroups [15] defined by $G_i = \{\sigma_i(a,b,c,d), a^2+b^2+c^2+d^2 = 1\}$ for $i \in \{1,2\}$, which give rise to the simple Lie algebras $\mathfrak{g}_i = \{\sigma_i(0,a,b,c), (a,b,c) \in \mathbb{R}^3\}$ $(\mathfrak{so}_4 = \mathfrak{g}_1 + \mathfrak{g}_2)$.

Now, some computations prove that the subgroup H generated by G_1 and the matrices $\sigma_2(0,0,1,0)$ and $\sigma_2(1/2,1/2,1/2,1/2)$ is infinite (as it contains G_1), with $\mathfrak{g}_1$ as its Lie algebra, and satisfies $\mathrm{env}\,\phi(H) = \mathrm{env}\,\phi(\mathbf{SO}_4)$. We must therefore strengthen the hypotheses.

If the Lie algebra of H is $\mathfrak{g}_1$, then H/G_1 (the image of H in the quotient group G/G_1) is finite. We will therefore require H/G_1 to be infinite. This gives:

Theorem 9. *Let* H *be a subgroup of* $\mathbf{SO}_4$. *Then* H *is dense if and only if* $\mathrm{env}\,\phi(H) = \mathrm{env}\,\phi(\mathbf{SO}_4)$ *and the groups* H/G_1 *and* H/G_2 *are infinite.*

Deciding if H/G_1 is infinite is easy. Indeed, $\mathbf{SO}_4/G_1$ is an algebraic group, and for this particular case, we know a representation of $\mathbf{SO}_4/G_1$.

Theorem 10. *Let* ψ_1 *be the morphism*

$$\begin{aligned} \psi_1 : \quad \mathbf{SO}_4 \quad &\mapsto \mathbf{SO}_4 \\ \begin{pmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{pmatrix} &\mapsto \sigma_1(a,b,c,d)^T \begin{pmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{pmatrix} \end{aligned}$$

Then ψ_1 *is a morphism of kernel* G_1.

Hence, testing for H/G_1 to be infinite is the same as testing for $\psi_1(H)$ to be infinite. The same result holds for G_2 with the morphism:

$$\psi_2 : \begin{array}{c} \mathbf{SO}_4 \\ \begin{pmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{pmatrix} \end{array} \mapsto \begin{array}{c} \mathbf{SO}_4 \\ \begin{pmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{pmatrix} \end{array} \sigma_2(a,e,i,m)^T$$

This gives the theorem:

Theorem 11. *A subgroup H of $\mathbf{SO}_4(\mathbb{R})$ is dense in $\mathbf{SO}_4$ if and only if $\mathrm{env}\,\phi(H) = \mathrm{env}\,\phi(\mathbf{SO}_4(\mathbb{R}))$ and the groups $\psi_1(H)$ and $\psi_2(H)$ are infinite.*

We thus obtain an algorithm to test the density of a subgroup of $\mathbf{SO}_4(\mathbb{R})$.

3.4 Universality

The discussion of the previous section can be generalised to $\mathbf{U}_n$, which has two normal connected subgroups $\mathbf{SU}_n$ and $\mathbf{U}_1\mathcal{I}$.

Theorem 12. *Let H be a subgroup of $\mathbf{U}_n$. Then H is dense if and only if $\mathrm{env}\,\phi(H) = \mathrm{env}\,\phi(G)$ and the groups* det H *and $\phi(H)$ are infinite.*

Proof. Just note that $\phi(\mathcal{M}) = \mathcal{I}$ if and only if $\mathcal{M} \in \mathbf{U}_1\mathcal{I}$. Hence $\phi(H)$ infinite means that $\mathbf{U}_n/\mathbf{U}_1\mathcal{I}$ is infinite, whereas $\det H$ infinite means that $\mathbf{U}_n/\mathbf{SU}_n$ is infinite. □

The following corollary is useful for our purposes

Corollary 13 *Let H be a subgroup of $\mathbf{U}_n$. Then $\mathbf{U}_1 H$ is dense if and only if $\mathrm{env}\,\phi(H) = \mathrm{env}\,\phi(G)$, and $\phi(H)$ is infinite.*

This gives an algorithm for $(k,0)$-universality:

Theorem 14. *There is an algorithm which, given a set S of gates over n qubits, decides if S is $(k,0)$-universal.*

The algorithm works as follows: compute the gates $(\mathcal{M} \otimes \mathcal{I}_{2^k})[\sigma]$ for any $\mathcal{M} \in S$ and $\sigma \in \mathfrak{S}_{k+n}$, then decide whether the group H generated by these gates is such that $\mathbf{U}_1 H$ is dense by computing $\mathrm{env}\,\phi(H)$, $\mathrm{env}\,\phi(\mathbf{U}_{2^{k+n}})$ and $\phi(H)$ and applying corollary 13. The size of the matrices is 2^{k+n}, and there are $(k+n)!\,|S|$ such matrices, hence the algorithm will be at least exponential in k and n. However, if k and n are fixed, the algorithm is polynomial in $|S|$ and the size of the coefficients.

Based on the same ideas, we can give a theorem about (0, 1)-universality. Let us write $\mathcal{A} \odot \mathcal{B}$ the block diagonal matrix $\mathrm{diag}(\mathcal{A}, \mathcal{B})$. Note that (0, 1)-universality means that for any quantum gate $\mathcal{M}$ over n qubits, there exists $\lambda \in \mathbf{U}_1$ and

a quantum gate $\mathcal{N}$ in $\overline{G_{n+1}(S)}$ such that $P_{n,n+1}(\mathcal{N}) = \lambda\mathcal{M}$ (that is, as $\mathcal{M}$ is unitary, $\mathcal{N} = \lambda\mathcal{M} \odot \mathcal{C}$ for some $\mathcal{C}$).

Let Δ_n be the subspace of $\mathbf{M}_{2^{n+2}}(\mathbb{C})$ spanned by matrices of the form $\mathcal{A}_1 \odot \mathcal{A}_2 \odot \mathcal{A}_3 \odot \mathcal{A}_4$, $\mathcal{A}_i \in \mathbf{M}_{2^n}(\mathbb{C})$. For every subspace H of $\mathbf{M}_{2^{n+2}}(\mathbb{C})$, we will denote by $E_n(H)$ the set of matrices $\mathcal{M}$ such that $\mathcal{M} \odot \mathcal{A}_2 \odot \mathcal{A}_3 \odot \mathcal{A}_4 \in H$ for some $\mathcal{A}_i \in \mathbf{M}_{2^n(\mathbb{C})}$

Theorem 15. *Let $\mathcal{S}$ be a set of quantum gates over n qubits and $H = G_{n+1}(\mathcal{S})$. Then $\mathcal{S}$ is (0,1)-universal if and only if the set $E_{2n}(\phi(H))$ is infinite and the condition* $\operatorname{env}\phi(\mathbf{U}_{2^n}) \subseteq P_{2n,2n+2}(\operatorname{env}\phi(H) \cap \Delta_{2n})$ *holds.*

The second condition basically means that for every $\mathcal{G} \in \mathbf{U}_{2^n}$, there exists an $\mathcal{M} \in \operatorname{env}\phi(H)$ such that $\mathcal{M} = \phi(\mathcal{G}) \odot \mathcal{H}_1 \odot \mathcal{H}_2 \odot \mathcal{H}_3$ for some $\mathcal{H}_i$.

Proof. The first condition is obviously necessary. If S is (0, 1)-universal, for every $\mathcal{G} \in \mathbf{U}_{2^n}$, there exists a $\mathcal{N} \in \overline{H}$ such that $\mathcal{N} = \lambda\mathcal{G} \odot \mathcal{C}$ for some $\mathcal{C}$ and λ. Then $\phi(\mathcal{N}) = \phi(\mathcal{G}) \odot (\lambda\mathcal{G} \otimes \overline{\mathcal{C}}) \odot (\mathcal{C} \otimes \overline{\lambda\mathcal{G}}) \odot \phi(\mathcal{C})$, hence the second condition.

Conversely, let X be the set of matrices $\mathcal{A}$ such that there exists $\mathcal{B}$ with $\mathcal{A} \odot \mathcal{B} \in \mathfrak{h}$, the Lie Algebra of $\overline{H}$. X is obviously a subalgebra of $\mathfrak{u}_{2^n}$. The first condition states that $X \cap \mathfrak{su}_{2^n} \neq \{0\}$.

We now interpret the second condition. A straightforward computation shows that $(\phi(\mathcal{G}) \odot \mathcal{H}_1 \odot \mathcal{H}_2 \odot \mathcal{H}_3)\operatorname{Vec}(\mathcal{A} \odot \mathcal{B}) = \operatorname{Vec}(\mathcal{G}^T\mathcal{A}\mathcal{G} \odot \mathcal{C})$ with $\operatorname{Vec}\mathcal{C} = \mathcal{H}_3 \operatorname{Vec}\mathcal{B}$. Hence X is stable by conjugation by any $\mathcal{G} \in \mathbf{U}_{2^n}$.

X is therefore an ideal of $\mathfrak{u}_{2^n}$, and by simplicity we obtain $\mathfrak{su}_{2^n} \subseteq X$. By definition of X, H contains for all $\mathcal{G} \in \mathbf{SU}_{2^n}$ an element of the form $\mathcal{G} \odot \mathcal{C}$ for some $\mathcal{C}$. Hence $\mathcal{G}U_1H$ contains for all $\mathcal{G} \in \mathbf{U}_{2^n}$ an element of the form $\mathcal{G} \odot \mathcal{C}$. □

The condition about the enveloping algebra uses basic notions of linear algebra, hence is easy to test. The finiteness of $E_{2n}(\phi(H))$ is more difficult to test, as we do not know generators of this group. However, if S is (0, 1)-universal, then $\phi(H)$ must obviously be infinite.

Conversely, suppose $\operatorname{env}\phi(\mathbf{U}_{2^n}) \subseteq P_{2n,2n+2}(\operatorname{env}\phi(H) \cap \Delta_{2n})$ and $\phi(H)$ is infinite. Then, using the same kind of arguments, we can prove that S is (1,1)-universal. We then have the following theorem

Theorem 16. *Let S be a set of quantum gates over n qubits, $H = G_{n+1}(S)$.*

- *If S is (0, 1)-universal then* $\operatorname{env}\phi(\mathbf{U}_{2^n}) \subseteq P_{2n,2n+2}(\operatorname{env}\phi(H) \cap \Delta_{2n})$ *and $\phi(H)$ is infinite.*
- *If* $\operatorname{env}\phi(\mathbf{U}_{2^n}) \subseteq P_{2n,2n+2}(\operatorname{env}\phi(H) \cap \Delta_{2n})$ *and $\phi(H)$ is infinite, then S is (1,1)-universal.*

This theorem can be extended to (k,p)-universality. It doesn't provide a direct characterisation of (k,p)-universality, but if one is just interested in the number of ancillas (that is p) necessary for S to be universal, this will give a decent algorithm.

4 Conclusion

We prove that many of the notions of universality are distinct. The basic open question is about (k, ∞)-universality. If we can approach any gate with some set of gates S using ancillas, is the number of necessary ancillas related to the precision, or can we find a bound for the number of ancillas ? We conjecture that a set S is (k, ∞)-universal if and only if it is (k, p)-universal for some p.

We also give an algorithm which decides in polynomial time if some subgroup of a classical group $(\mathbf{SU}_n, \mathbf{SL}_n, \mathbf{SO}_n \ldots)$ is (Zariski-)dense. This gives an algorithm to decide $(k, 0)$-universality. We only provide a partial result for (k, p)-universality. It would be interesting to give polynomial time algorithms for other decision problems about Zariski-closed groups, such as deciding if the intersection of two Zariski-closed groups is non-trivial, as this would lead to a polynomial time algorithm for the (k, p)-universality.

References

1. Post, E.: The two-valued iterative systems of mathematical logic. Volume 5 of Annals Mathematical Studies. Princeton University Press (1941)
2. Fredkin, E., Toffoli, T.: Conservative Logic. International Journal of Theoretical Physics **21** (1982) 219–253
3. Lloyd, S.: Almost any quantum logic gate is universal. Physical Review Letters **75** (1995) 346–349
4. Deutsch, D., Barenco, A., Ekert, A.: Universality in quantum computation. Proceedings of the Royal Society of London, Series A **449** (1995) 669–677
5. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N.H., Shor, P.W., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Physical Review A **52** (1995) 3457–3467
6. Barenco, A.: A universal two-bit gate for quantum computation. Proceedings of the Royal Society of London, Series A **449** (1995) 679–683
7. Kitaev, A., Shen, Vyalyi: Classical and Quantum computation. Volume 47 of Graduate Studies in Mathematics. American Mathematical Society (2003)
8. Shi, Y.: Both Toffoli and Controlled-NOT need little help to do universal quantum computation. Quantum Information and Computation **3** (2003) 84–92
9. Brylinski, J.L., Brylinski, R.: Universal Quantum Gates. In: Mathematics of Quantum Computation. Chapman & Hall (2002)
10. Derksen, H., Jeandel, E., Koiran, P.: Quantum automata and algebraic groups, to appear in Journal of Symbolic Computation (2004)
11. Onishchik, A., Vinberg, E.: Lie groups and algebraic groups. Springer-Verlag, Berlin (1990)
12. Beals, R.: Algorithms for Matrix Groups and the Tits Alternative. Journal of Computer and System Sciences **58** (1999) 260–279
13. Graham, A.: Kronecker Products and Matrix Calculus: with Applications. Ellis Horwood Limited (1981)
14. Babai, L., Beals, R., Rockmore, D.N.: Deciding finiteness for matrix groups in deterministic polynomial time. In: ISSAC' 93, ACM Press (1993) 117–126
15. Mneimné, R., Testard, F.: Introduction à la théorie des groupes de Lie classiques. Hermann (1986)

Approximation Algorithms for the Capacitated Minimum Spanning Tree Problem and Its Variants in Network Design*

Raja Jothi and Balaji Raghavachari

University of Texas at Dallas, Richardson, TX 75083.
{raja,rbk}@utdallas.edu

Abstract. Given an undirected graph $G = (V,E)$ with non-negative costs on its edges, a root node $r \in V$, a set of demands $D \subseteq V$ with demand $v \in D$ wishing to route $w(v)$ units of flow (weight) to r, and a positive number k, the *Capacitated Minimum Steiner Tree* (CMStT) problem asks for a minimum Steiner tree, rooted at r, spanning the vertices in $D \cup \{r\}$, in which the sum of the vertex weights in every subtree hanging off r is at most k. When $D = V$, this problem is known as the *Capacitated Minimum Spanning Tree* (CMST) problem. Both CMStT and CMST problems are NP-hard. In this paper, we present approximation algorithms for these problems and several of their variants in network design. Our main results are the following.

- We give a $(\gamma\rho_{ST} + 2)$-approximation algorithm for the CMStT problem, where γ is the *inverse Steiner ratio* and ρ_{ST} is the best achievable approximation ratio for the Steiner tree problem. Our ratio improves the current best ratio of $2\rho_{ST} + 2$ for this problem.
- In particular, we obtain $(\gamma + 2)$-approximation ratio for the CMST problem, which is an improvement over the current best ratio of 4 for this problem. For points in Euclidean and Rectilinear planes, our result translates into ratios of 3.1548 and 3.5, respectively.
- For instances in the plane, under the L_p norm, with the vertices in D having uniform weights, we give a non-trivial $(\frac{7}{5}\rho_{ST} + \frac{3}{2})$-approximation algorithm for the CMStT problem. This translates into a ratio of 2.9 for the CMST problem with uniform vertex weights in the L_p metric plane. Our ratio of 2.9 solves the long standing open problem of obtaining a ratio any better than 3 for this case.

1 Introduction

In this paper, we consider the *Capacitated Minimum Steiner Tree* (CMStT) problem, one of the extensively-studied network design problem in telecommunications. The CMStT problem can formally be defined as follows.

CMStT: Given an undirected graph $G = (V, E)$ with non-negative costs on its edges, a root node $r \in V$, a set of demands $D \subseteq V$ with with demand $v \in D$

* Full version of the paper available at http://www.utdallas.edu/~rajа/Pub/cmst.ps. Research supported in part by the NSF under grant CCR-9820902.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 805–818, 2004.

wishing to route $w(v)$ units of flow (weight) to r, and a positive number k, the *Capacitated minimum Steiner tree* (CMStT) problem asks for a minimum Steiner tree, rooted at r, spanning the vertices in $D \cup \{r\}$, in which the sum of the vertex weights in every subtree hanging off r is at most k.

The capacity constraint k must be at least as much as the largest vertex weight for the CMStTproblem to be feasible. The CMStT problem is NP-hard as the case with $k = \infty$ is the minimum Steiner tree problem, which is NP-hard. When $D = V$, the CMStT problem is the well-known *Capacitated Minimum Spanning Tree* (CMST) problem. The CMST problem is NP-hard [3,8] even for the case when vertices have unit weights and $k = 3$. The problem is polynomial-time solvable if all vertices have unit weights and $k = 2$ [3]. The problem can also be solved in polynomial time if vertices have 0,1 weights and $k = 1$, but remains NP-hard if vertices have 0,1 weights, $k = 2$ and all edge lengths are 0 or 1 [3]. Even the geometric version of the problem, in which the edge costs are defined to be the Euclidean distance between the vertices they connect, remains NP-hard.

The CMST problem has been well studied in Computer Science and Operations Research for the past 40 years. Numerous heuristics and exact algorithms have been proposed (see full version of paper `http://www.utdallas.edu/~raja/Pub/cms.ps` for survey on the literature). Although most of the heuristics solve several well known instances close to optimum, they do not provide any approximation guarantee on the quality of the solutions obtained. Exact procedures are limited to solving smaller instances because of their exponential running time. In this paper, we present improved approximation algorithms for the CMStTand CMST problems and their variants.

1.1 Previous Results

For the CMST problem with uniform vertex weights, Gavish and Altinkemer [4] presented a modified *parallel savings algorithm* (PSA) with approximation ratio $4 - 1/(2^{\lceil \log k \rceil - 1})$. In 1988, Altinkemer and Gavish [1] gave improved approximation algorithms with ratios $3 - \frac{2}{k}$ and 4 for the uniform and non-uniform vertex weight cases, respectively. They construct a traveling salesman tour (TSP) with length of at most twice the minimum spanning tree (MST), and partition the tour into segments (subtrees) of weight at most k. Partitioned subtrees are then connected to the root vertex using direct edges. Hassin, Ravi and Salman [6] presented algorithms for the 1-cable *Single-Sink Buy-at-Bulk* problem. The algorithms in [1] and [6] can be used to obtain ratios of $2\rho_{ST} + 1$ and $2\rho_{ST} + 2$ for the respective uniform and non-uniform vertex weight CMStT problems.

1.2 Our Contributions

In this paper, we solve the long-standing open problem of obtaining better approximation ratios for the CMST problem. Our main results are the following.

- We give a $(\gamma\rho_{ST}+2)$-approximation algorithm for the CMStT problem, where γ is the *inverse Steiner ratio*[1] and ρ_{ST} is the best achievable approximation ratio for the Steiner tree problem. Our ratio improves the current best ratio of $2\rho_{ST}+2$ for this problem.
- In particular, we obtain $(\gamma+2)$-approximation ratio for the CMST problem, which is an improvement over the current best ratio of 4 for this problem. For points in Euclidean and Rectilinear planes, our result translates into ratios of 3.1548 and 3.5, respectively.
- For instances in the plane, under the L_p norm, with the vertices in D having uniform weights, we give a non-trivial $(\frac{7}{5}\rho_{ST}+\frac{3}{2})$-approximation algorithm for the CMStT problem. This translates into a ratio of 2.9 for the CMST problem with uniform vertex weights in the L_p metric plane. Our ratio of 2.9 solves the long standing open problem of obtaining a ratio any better than 3 for this case.
- For the CMST problem, we show how to obtain a 2-approximation for graphs in metric spaces with unit vertex weights and $k=3,4$.
- For the *budgeted* CMST problem, in which the weights of the subtrees hanging off r could be up to αk instead of k $(\alpha \geq 1)$, we obtain a ratio of $\gamma+\frac{2}{\alpha}$.

Of the above results, the 2.9-approximation result for the CMST problem is of most significance. This is due to the fact that obtaining a ratio any better than 3 for graphs defined in the Euclidean plane (with uniform vertex weights) is not straightforward. There are several ways one can obtain a ratio of 3 for this problem ([1], modified algorithm of [6], our algorithm in Section 3.1). But the question was whether one can ever obtain a ratio smaller than $3 - o(1)$ for this version of the CMST problem. We present an example (in Section 4), which shows that, with the currently available lower bounds for the CMST problem, it is not possible to obtain an approximation ratio any better than 2. We introduce a novel concept of *X-trees* to overcome the difficulties in obtaining a ratio better than 3.

Achieving ratios better than 3 and 4 for the uniform and non-uniform vertex weighted CMST problems, respectively, has been an open problem for 15 years now. One major reason for the difficulty in finding better approximations is that there is no non-trivial lower bound for an optimal solution. There are instances for which the cost of an optimal solution can be as much as $\Omega(n/k)$ times than that of an MST. Inability to find better lower bounds has greatly impeded the process of finding better approximation ratios for this problem. Even though we were not able to completely eliminate the use of MST as a lower bound, we found ways to exploit its geometric structure, thereby achieving better performance ratios. Unlike the algorithms in [1], in which the MST lower bound contributes a factor of 2 to the final ratio, our algorithms minimizes the use of MST lower bound, thereby achieving better ratios.

[1] The Steiner ratio is the maximum ratio of the costs of the minimum cost Steinei tree versus the minimum cost spanning tree for the same instance.

2 Preliminaries

Let $|uv|$ denote the distance between vertices u and v. Length of an edge is also its cost. The terms points, nodes and vertices will be used interchangeably in this paper. For a given k, let OPT and APP denote optimal and approximate solutions, respectively, and let C_{opt} and C_{app} denote their respective costs. Let C_{mst} and C_{ST} denote the costs of an MST and an optimal Steiner tree, respectively.

In a rooted tree T, let T_v denote the subtree rooted at v. Let C_T denote the cost of tree T. Let $w(v)$ denote the weight of vertex v, and let $w(T_v)$ denote the sum of vertex weights in the subtree rooted at v. For the CMStT problem, the weight of a vertex the is not in D is assumed to be 0. By weight of a subtree, we mean the sum of the vertex weights in that subtree. We call as *spokes,* the edges incident on r of a CMStT. By *level* of a vertex, in a tree T rooted at r, we mean the number of tree edges on its path to r (also known as depth).

By "metric completion" of a given graph (whose edges obey triangle inequality) we refer to a complete graph. Throughout this paper, without loss of generality, we assume that the metric completion of the input graph is available, and that the weights of vertices in $V \backslash D$ is zero. All our algorithms in this paper are for the CMStT problem—a generalization of the CMST problem. The following lemma gives a lower bound on the cost of an optimal solution.

Lemma 1. $C_{opt} \geq \frac{1}{k} \sum_{v \in V} w(v)|rv|$.

3 CMStT Algorithms

We first construct a ρ_{ST}-approximate Steiner tree T spanning all the vertices in $D \cup \{r\}$, and then root T at the root vertex r. Next, we prune subtrees of weight at most k in a bottom-up fashion, and add edges to connect r to the closest node in each of the pruned subtrees. In simple terms, we basically cut T into subtrees of weight at most k and connect them to the root vertex.

It is safe to assume that nodes have integer weights. The assumption is not restrictive as any CMStT problem with rational weights can be converted to an equivalent problem with integer node weights. The optimal solution for the scaled problem is identical to that of the original problem [1].

Since our algorithm for the uniform vertex weights case is quite complex, we first present the algorithm for the general case (non-uniform vertex weights), which will help in an easier understanding of our algorithm for the uniform vertex weights case. Note that all our algorithms start with a ρ_{ST}-approximate Steiner tree of constant degree. Before we proceed to the algorithms, we present the following important lemma.

Lemma 2. *For a given graph $G = (V, E)$, a set of demands $D \subseteq V$, $r \in V$, and a k, let T_f be a feasible* CMStT *and let $t_1, t_2, \ldots, t_m$ be the subtrees hanging off r in T_f. Let $w(t_q)$ be the weight of a minimum weight subtree t_q hanging off r. For all i, if the cost of the edge connecting subtree t_i to r is minimal, then the cost C_{sp} of all the edges incident on r (spokes) in T_f is at most $k/w(t_q)$ times the cost of an optimal solution.*

Proof. Let Γ be the set of vertices in $t_1, \ldots, t_m$. For all i, let v_i be the vertex in t_i through which t_i is connected to r. Recall that edge rv_i is a spoke, and that it is a minimal cost edge crossing the cut between r and t_i. Then,

$$|rv_i| \leq \frac{\sum_{v \in t_i} w(v)|rv|}{\sum_{v \in t_i} w(v)} \leq \frac{\sum_{v \in t_i} w(v)|rv|}{w(t_q)}.$$

The cost of the all the edges incident on r is given by

$$C_{sp} = \sum_{i=1}^{m} |rv_i| \leq \frac{\sum_{v \in \Gamma} w(v)|rv|}{w(t_q)} = \frac{k}{w(t_q)} \times \frac{\sum_{v \in D} w(v)|rv|}{k}$$
$$\leq \frac{k}{w(t_q)} \times C_{opt}. \quad \text{(by Lemma 1)}$$

3.1 Non-uniform Vertex Weights

The algorithm given below outputs a feasible CMStT for a given instance, whose edges obey triangle inequality. Note that during the course of the algorithm, we replace real vertices with *dummy* vertices of zero weight. These dummy vertices can be thought of as Steiner points. In the algorithm, we use c_i to denote the subtree rooted at child i of vertex v, and p_v to denote v's parent.

Algorithm CMStT-NONUNIFORM
Input: ρ_{ST}-approximate Steiner tree T rooted at r.

1. Choose a maximum level vertex $v \neq r$ such that $w(T_v) \geq k$. If there exists no such vertex then STOP.
2. If $w(T_v) = k$, then replace the Steiner tree edges incident on the vertices in T_v with edges of a minimal cost tree τ spanning only the vertices in $T_v \cap D$. Add a new edge connecting r to the closest vertex in τ.
3. Else if, for some i, $w(c_i) \geq k/2$, then replace the Steiner tree edges incident on the vertices in c_i with edges of a minimal cost tree τ spanning only the vertices in $c_i \cap D$. Add a new edge connecting r to the closest vertex in τ.
4. Else if $\sum w(c_i) < k/2$, which means $w(v) > k/2$, then replace v with a dummy vertex. In the final solution, add v and an edge connecting v to r.
5. Else collect a subset s of subtrees, each of which is rooted at one of v's children, such that $k/2 \leq w(s) \leq k$. Replace the Steiner tree edges incident on the vertices in s with edges of a minimal cost tree τ spanning only the vertices in $s \cap D$. Add a new edge connecting r to the closest vertex in τ.
6. Go to step 1.

It can be verified that our algorithm outputs a feasible CMStT for a given k.

Theorem 1. *For a given* CMStT *instance, Algorithm* CMStT-NONUNIFORM *guarantees an approximation ratio of* $(\gamma\rho_{ST} + 2)$.

Proof. We show that the cost of the tree output by Algorithm CMStT-NONUNIFORM is at most $\gamma\rho_{ST} + 2$ times the cost of an optimal CMStT. The input to the algorithm is a ρ_{ST}-approximate Steiner tree T.

It can be easily verified from the algorithm that all the new edges added to the original tree T are either new spokes, or edges that interconnect vertices within the subtrees for which the new spokes were added. In what follows, we account for the cost of the new spokes added to T, followed by the cost of other edges in the final solution output by the algorithm.

A new spoke, incident on a subtree, is added to the original Steiner tree if and only if the weight of the subtree it connects is at least $k/2$. Notice that the algorithm outputs a tree with each subtree hanging off r being disjoint and the weight of every such subtree, for which a new spoke was added, is at least $k/2$. Let C_{sp} be the cost of the spokes that the algorithm "adds" to the Steiner tree. Note that C_{sp} does not include the cost of the spokes that are already in the Steiner tree that was given as input to the algorithm. By Lemma 2, $C_{sp} \leq 2 \times C_{opt}$.

Now, we account for the cost of other edges in the final solution. These edges are either the Steiner tree edges or the edges that replaced the Steiner tree edges. We show that the total cost of all these edges together is at most γ times the cost of the initial Steiner tree. To prove this, it suffices to prove that the cost of the edges that replace the Steiner tree edges is at most γ times the cost of the Steiner tree edges that it replaces. For every subtree formed, notice that the algorithm replaced the edges of the Steiner tree spanning the vertices in that subtree by the edges of an MST spanning only the non-zero weight vertices in that subtree. Since γ was defined to be the inverse Steiner ratio (ratio of the cost of an MST versus the cost of an optimal Steiner tree), by Steiner ratio argument, the cost of the MST spanning only the non-zero weight vertices in a subtree is at most γ times the cost of an optimal Steiner tree spanning the non-zero weight vertices in that subtree. Thus, we can conclude that the cost of the new edges is at most γ times the cost of the ρ_{ST}-approximate Steiner tree edges it replaces. The final cost of the tree output by the algorithm is given by

$$C_{app} \leq C_{sp} + \gamma\rho_{ST}C_{ST} \leq 2C_{opt} + \gamma\rho_{ST}C_{opt} \leq (\gamma\rho_{ST} + 2)C_{opt}.$$

Corollary 1. For *the* CMStT *problem with uniform vertex weights, Algorithm* CMStT-NONUNIFORM *with little modification guarantees a* $(\rho_{ST}+2)$*-approximation ratio.*

Proof. Since we are dealing with uniform vertex weights, without loss of generality, we can assume that they are of unit weight, and thus we can eliminate Step. 4 from Algorithm CMStT-NONUNIFORM. Therefore no dummy vertices are introduced by the algorithm. Once a subtree t of size at least $k/2$ is found, instead of replacing the Steiner tree spanning the vertices in t with a MST spanning the non-zero weight vertices in t, we can just use the edges in t, minus the edge that connects t to its parent, as they are. This eliminates the γ from the final ratio.

Corollary 2. *For the* CMST *problem, Algorithm* CMStT-NONUNIFORM *guarantees a* $(\gamma + 2)$*-approximation ratio. In particular, for points in Euclidean and rectilinear planes, it guarantees a ratio of 3.1548 and 3.5, respectively.*

3.2 Uniform Vertex Weights

Although our algorithm for uniform vertex weights case is similar to Algorithm CMStT-NONUNIFORM at the top-level, contrary to expectations, there are some complicated issues that have to be handled in order to obtain an approximation ratio strictly less than $\rho_{ST} + 2$. From our analysis for the non-uniform vertex weights case, we can see that the weight of the minimum weight subtree hanging off r plays a crucial role in the calculation of the approximation ratio. An obvious heuristic is to prune subtrees of weight as close as possible to k, so that the ratio drops considerably. We will soon see why pruning subtrees of weight strictly greater than $k/2$ is more difficult than pruning subtrees of weight greater than or equal to $k/2$. To overcome the difficulty of pruning subtrees of size strictly greater than $k/2$, we introduce the concept of *X-trees,* which we define below. We call a subtree, T_v, rooted at vertex v as an *X-tree,* x, if all of the following properties are satisfied (follow Fig. 1).

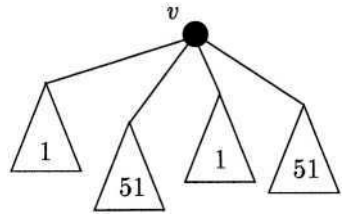


Fig. 1. An X-tree with $k = 100$.

- $k < w(T_v) < \frac{4}{3}k$.
- Weight of no subtree hanging off v is between $\frac{2}{3}k$ and k.
- Sum of the weights of no two subtrees hanging off v is between $\frac{2}{3}k$ and k.
- Sum of the weights of no three subtrees hanging off v is between $\frac{2}{3}k$ and k.

The following proposition follows from the definition of an X-tree.

Proposition 1. *Let* v_1 *be a maximum level vertex in an X-tree rooted at* v *such that* T_{v_1} *is also an X-tree (*v_1 *could be* v *itself). If there is no subtree (non-X-tree) of weight greater than* k *rooted at one of* v_1*'s children, then there always exist two subtrees,* t_α *and* t_β*, hanging off* v_1 *such that* $k < w(t_\alpha) + w(t_\beta) < \frac{4}{3}k$ *and* $\frac{1}{3}k < w(t_\alpha), w(t_\beta) < \frac{2}{3}k$.

Since the vertices are of uniform weight, without loss of generality, we can assume that they are of unit weight, and scale k accordingly. We also assume

that a ρ_{ST}-approximate Steiner tree is given as part of the input. Note that we are trying to solves instances in L_p metric plane. Even though, the maximum nodal degree in a Steiner tree on a plane is 3, we will continue as if it is 5. This is to ensure that our algorithm solves CMST instances on a plane, as the maximum degree of an MST on a L_p plane is 5 [7,9]. Note that every vertex but root in a tree, with vertex degrees at most 5, has at most 4 children. The algorithm given below finds a feasible CMStT for instances defined on a L_p plane. In the algorithm, we use c_i to denote the subtree rooted at child i of vertex v, and x_j to denote the X-tree rooted at child j of vertex v.

Algorithm CMStT-UNIFORM
Input: ρ_{ST}-approximate Steiner tree T rooted at r

1. Choose a maximum level vertex $v \neq r$ such that T_v is a non-X-tree with $w(T_v) \geq k$. If there exists no such vertex then go to step 11.
2. If $w(T_v) = k$, then add a new edge connecting r to the closest node in T_v. Remove edge vp_v from T.
3. Else if, for some i, $2k/3 \leq w(c_i) \leq k$, then add a new edge connecting r to the closest node in c_i. Remove the edge connecting v to c_i from T.
4. Else if, for some i and j $(i \neq j)$, $2k/3 \leq w(c_i) + w(c_j) \leq k$, then replace edges vc_i and vc_j by a minimal cost edge connecting c_i and c_j, merging the two subtrees into a single tree s. Add a new edge to connect r to the closest node in s.
5. Else if, for some i, j and z $(i \neq j \neq z)$, $2k/3 \leq w(c_i) + w(c_j) + w(c_z) \leq k$, then replace the Steiner tree edges incident on the vertices in c_i, c_j and c_z by a minimal cost tree s spanning all the vertices in c_i, c_j and c_z. Add a new edge to connect r to the closest node in s.
6. Else if, for some i, j and z $(i \neq j \neq z)$, $4k/3 \leq w(c_i) + w(c_j) + w(c_z) \leq 2k$, then do the following.
   Let E_i be the set of edges incident on vertices in c_i. We define E_j (E_z) with respect to c_j $(c_z$ resp.) analogously. Without loss of generality, let E_j be the low-cost edge set among E_i, E_j and E_z. Use DFS on c_j to partition the vertices in c_j into two sets g_1 and g_2 such that the total weight of vertices in $(c_i \cup g_1) \cap D$ is almost the same as the total weight of vertices in $(c_z \cup g_2) \cap D$. Remove all the edges incident on the vertices in subtrees c_i, c_j and c_z. Construct a minimal cost spanning tree s_1 comprising the vertices in c_i and g_1. Similarly, construct a minimal cost spanning tree s_2 comprising the vertices in c_z and g_2. Add new edges to connect r to the closest nodes in s_1 and s_2.
7. Else if, for some i and j $(i \neq j)$, $2k < w(x_i) + w(x_j) < 8k/3$, do the following. Let v_1 and v_2 be two maximum level vertices in X-trees x_i and x_j respectively, such that T_{v_1} and T_{v_2} are X-trees themselves (see Fig. 2). Recall, by Proposition 1, that there exist two subtrees t_{α_1} and t_{β_1} (t_{α_2} and t_{β_2}), hanging off v_1 (v_2 resp.) such that $k < w(t_{\alpha_1}) + w(t_{\beta_1}) < \frac{4}{3}k$ ($k < w(t_{\alpha_2}) + w(t_{\beta_2}) < \frac{4}{3}k$ resp.).
   Let E_1 represent the set of edges incident on vertices in t_{α_1} (see Fig. 3). Let E_2 represent the set of edges incident on vertices in t_{β_1}. We define E_4

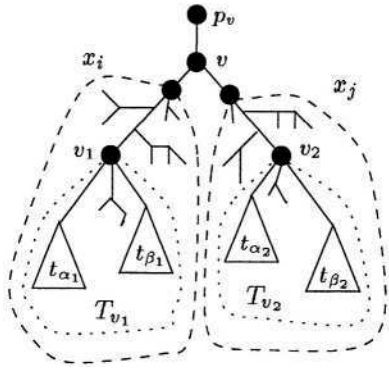


Fig. 2.

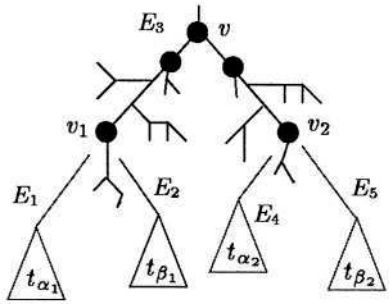


Fig. 3.

(E_5) with respect to t_{α_2} (t_{β_2} resp.) analogously. Let E_3 be the set of edges incident on vertices in x_i and x_j minus the edges in E_1, E_2, E_4 and E_5. Let $G_1 = \{E_1, E_2\}$, $G_2 = \{E_3\}$, and $G_3 = \{E_4, E_5\}$ be three groups. Out of $\{E_1, E_2, E_3, E_4, E_5\}$, double two low-cost edge sets such that they belong to different groups.

a) If E_i and E_j were the two edges sets that were doubled, with E_i in G_1 and E_j in G_3, then form three minimal cost subtrees s_1, s_2 and s_3 spanning the vertices in x_i and x_j as follows. Without loss of generality, let E_2 and E_4 be the two low-cost edge sets that were doubled (Fig. 4). Use shortcutting to form s_1 spanning all vertices in t_{α_1} and a subset of vertices in t_{β_1}, form s_3 spanning all vertices in t_{β_2} and a subset of vertices in t_{α_2}, and form s_2 with all the left-over vertices. Remove edge vp_v. Since $k < w(t_{\alpha_1}) + w(t_{\beta_1}) < 4k/3$, $k < w(t_{\alpha_2}) + w(t_{\beta_2}) < 4k/3$, and $2k \leq w(x_i) + w(x_j) \leq 8k/3$, we can form s_1, s_2 and s_3 of almost equal weight with $2k/3 \leq w(s_1), w(s_2), w(s_3) \leq k$.

b) If E_i and E_j were the two edges sets that were doubled, with E_i in G_1 or G_3, and E_j in G_2, then form three minimal cost subtrees s_1, s_2 and s_3 spanning the vertices in x_i and x_j as follows. Without loss of generality,

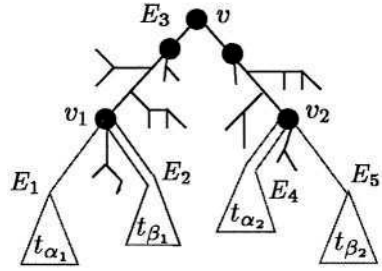


Fig. 4.

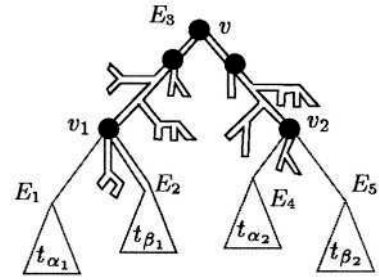


Fig. 5.

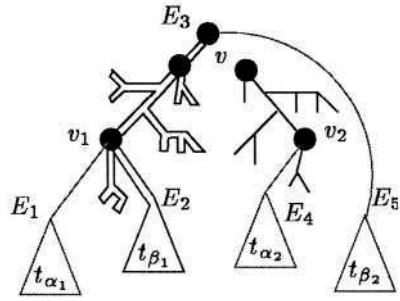


Fig. 6.

let E_2 and E_3 be the two low-cost edge sets that were doubled (see Fig. 5). From t_{α_2} and t_{β_2} find a vertex w such that $|wr|$ is minimum. Without loss of generality, let t_{α_2} contain w. Use shortcutting to form s_3 spanning all the vertices in x_j minus the vertices in t_{β_2} (see Fig. 6). Note that $k/3 < w(s_3) < k$, as x_j and T_{v_2} are X-trees and $k/3 < w(t_{\alpha_2}), w(t_{\beta_2}) < 2k/3$. Also, since $k/3 < w(t_{\beta_2}) < 2k/3$ and $k < w(x_i) < 4k/3$, subtrees s_1 and s_2 together will be of weight at least $4k/3$ and at most $2k$ (see Fig. 6). Form subtrees s_1 and s_2, using the ideas in Step. 6, such that $2k/3 \leq w(s_1), w(s_2) \leq k$ and $4k/3 \leq w(s_2) + w(s_3) \leq 2k$.

c) Add new edges to connect r to the closest nodes in s_1, s_2 and s_3.

8. Else if, for some i and j $(i \neq j)$, $4k/3 \leq w(x_i)+w(c_j) < 2k$, do the following. Let v_1 be a maximum level vertex in X-tree x_i such that T_{v_1} is an X-tree itself. Recall, by Proposition 1, that there exist two subtrees t_{α_1} and t_{β_1}, hanging off v_1 such that $k < w(t_{\alpha_1}) + w(t_{\beta_1}) < \frac{4}{3}k$.
   Let E_1 represent the set of edges incident on vertices in t_{α_1}. Let E_2 represent the set of edges incident on vertices in t_{β_1}. Let E_3 be the set of edges incident on vertices in x_i and c_j minus the edges in E_1 and E_2. Form subtrees s_1 and s_2 using the ideas in Step. 6. Add new edges to connect r to the closest nodes in s_1 and s_2.
9. Else if, $4k/3 \leq w(T_v) \leq 2k$, do the following. Let v_1 be a maximum level vertex in X-tree x_i such that T_{v_1} is an X-tree itself. Recall, by Proposition 1, that there exist two subtrees t_{α_1} and t_{β_1}, hanging off v_1 such that $k < w(t_{\alpha_1}) + w(t_{\beta_1}) < \frac{4}{3}k$.
   Let E_1 represent the set of edges incident on vertices in t_{α_1}. Let E_2 represent the set of edges incident on vertices in t_{β_1}. Let E_3 be the set of edges incident on vertices in T_v minus the edges in E_1 and E_2. Form subtrees s_1 and s_2 using the ideas in Step. 6. Add new edges to connect r to the closest nodes in s_1 and s_2.
10. Go to step 1.
11. While there is an X-tree, x, hanging off r, pick a maximum level vertex v_1 in x such that T_{v_1} is also an X-tree. Out of the two subtrees, t_α and t_β, hanging off v_1 (by Proposition 1), without loss of generality, let t_α be the subtree that is closer to r. Remove the edge connecting t_α to v_1, and add a new edge to connect r to the closest node in t_α.

Theorem 2. *For a given* CMStT *instance on a L_p plane, Algorithm* CMStT-UNIFORM *guarantees an approximation ratio of* $(\frac{7}{5}\rho_{ST} + \frac{3}{2})$.

Proof. We show that the cost of the tree output by Algorithm CMStT-UNIFORM is at most $(\frac{7}{5}\rho_{ST} + \frac{3}{2})$ times the cost of an optimal CMStT. The input to the algorithm is a ρ_{ST}-approximate Steiner tree T with maximum nodal degree at most 5.

The algorithm "adds" a new spoke to the tree whenever it prunes a subtree of weight at least $2k/3$. There are certain situations (Steps 6 and 11) where the algorithm adds a spoke for pruned subtrees of weight less than $2k/3$. We continue our analysis as if all of the pruned subtrees are of weight at least $2k/3$. This supposition makes the analysis of spoke cost simpler. We will soon justify this supposition (in Cases 5 and 8) in a manner that it does not affect the overall analysis in any way.

The cost of the spokes that were added to the initial Steiner tree is given by $C_{sp} \leq \frac{3}{2} \times C_{opt}$ by an argument analogous to that proving the cost of the spokes that the algorithm adds to the initial Steiner tree in Theorem 1. The above inequality follows immediately from the fact that a new spoke is added to the tree if and only if the subtree it connects to r is of weight at least $2k/3$.

Now, we account for the cost of other edges—all the edges in the final solution, except for the spokes added by the algorithm—in the final solution. We show

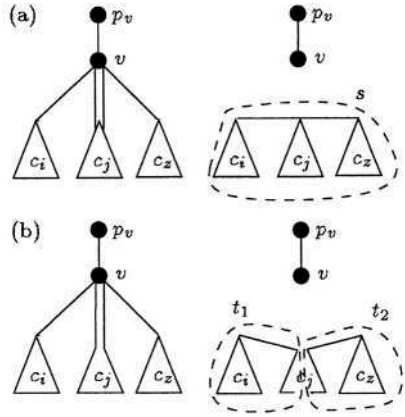


Fig. 7. Illustration (**a**) Step 5, (**b**) Step 6

that the cost of these edges is at most 7/5 times the cost of the Steiner tree edges that the algorithm started with. To prove this, it suffices to show that the cost of the edges that replace the Steiner tree edges is at most 7/5 times the cost of the edges that are replaced. In what follows, we show this by presenting a case-by-case analysis depending upon which step of the algorithm was executed.

Case 1. Steps 1, 2, 3 and 10 do not add any non-spoke edges. The weight of the subtrees for which Steps 1 and 2 adds spokes to the tree is at least $2k/3$.

Case 2. The minimal cost edge connecting c_i and c_j in Step 4 is at most the sum of the two Steiner tree edges that connects c_i and c_j to v (by triangle inequality). Hence no additional cost is involved.

Case 3. In Step 5, the cost of the tree s spanning all the vertices in c_i, c_j and c_z is at most the cost of the tree obtained by doubling the minimum cost edge out of the 3 Steiner tree edges that connect the 3 subtrees to v (see Fig. 7(a)). Hence, we can conclude that the cost of the tree constructed in Step 5 is at most 4/3 times the cost of the Steiner tree edges it replaces.

Case 4. In Step 6, the total cost of the trees s_1 and s_2 spanning all the vertices in c_i, c_j and c_z is at most the total cost of the trees t_1 and t_2 obtained by doubling the minimum cost edge set out of the 3 edge sets that are incident on the vertices in c_i, c_j and c_z, respectively (see Fig. 7(b)). Hence, we can conclude that the cost of the tree constructed in Step 6 is at most 4/3 times the cost of the Steiner tree edges it replaces.

Case 5. Step 7 forms three subtrees s_1, s_2 and s_3 from X-trees x_i and x_j. Since s_1, s_2 and s_3 can be formed by doubling two low-cost edge sets (belonging to two different groups) out of the 5 possible edge sets and shortcutting, we can conclude that the cost of the subtrees s_1, s_2 and s_3 constructed in Step 7 is at most 7/5 times the cost of the Steiner tree edges it replaces.

Accounting for the cost of the spokes added to the Steiner tree requires that each subtree pruned from the Steiner tree is of weight at least $2k/3$. We already proved that the cost of the spokes added to the Steiner tree is at most 3/2 times

the cost of an optimal solution. Without loss of generality, the requirement that each pruned subtree is of weight at least $2k/3$ can be interpreted as that of "charging" the spoke cost incident on a subtree to at least $2k/3$ vertices. Notice that this interpretation is valid only if the spoke connecting the subtree to the root is of minimal cost (r is connected to the closest node in the subtree).

Step 7(a) of the algorithm constructs three subtrees s_1, s_2 and s_3, each containing at least $2k/3$ vertices. This ensures that there are at least $2k/3$ vertices to which each of these subtrees can charge their spoke cost. This is not the case with Step 7(b) of the algorithm. As can be seen, subtree s_3 might be of weight less than $2k/3$. Since s_2 contains at least $2k/3$ vertices and $w(s_2)+w(s_3) \geq 4k/3$, and w is a vertex in x_j such that $|wv|$ is minimum, we can always charge the spoke costs of s_2 and s_3 to at least $4k/3$ vertices. Hence, our initial assumption that every pruned subtree is of weight at least $2k/3$ does not affect the analysis since there are at least $2k/3$ vertices for every spoke to charge.

Case 6. Analysis for Steps 8 and 9 are similar to that for Step 6 (Case 4).

Case 8. Step 11 prunes one subtree off X-tree x. The cost of the spoke $|rw|$ to connect t_α to r can be charged to all the vertices in the X-tree x as per the following argument. After disconnecting t_α from the X-tree, we are left with a subtree of $w(x) - w(t_\alpha) < k$ vertices. We do not need a new spoke for the left-over subtree as it is already connected to r using the Steiner tree edge. Hence, even for this case, our initial assumption that every pruned subtree is of weight at least $2k/3$ does not affect the analysis since there are at least $\frac{2}{3}k$ vertices to charge for the spoke added.

In all of the above cases, the cost of the edges that replace the Steiner tree edges is at most 7/5 times the cost of the Steiner tree edges that the algorithm started with. Thus, the total cost of the tree output by the algorithm is

$$C_{app} \leq \frac{7}{5}\rho_{ST}C_{ST} + \frac{3}{2}\,C_{opt} \leq \left(\frac{7}{5}\rho_{ST} + \frac{3}{2}\right)C_{opt}.$$

Corollary 3. *For the* CMST *problem in L_p plane with uniform vertex weights, Algorithm* CMStT-UNIFORM *guarantees a* 2.9*-approximation ratio.*

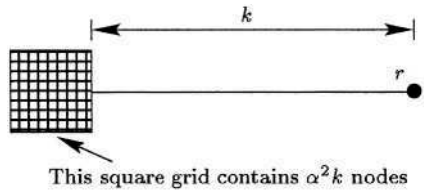


Fig. 8. A tight example

4 Conclusion

Our ratios are, certainly, not tight. We believe that there is room for improvement, at least for the CMST problem with uniform vertex weights, for which we obtain a ratio of 2.9. The cost of an optimal CMST can be lower bounded by one of the following two quantities: (i) the MST cost and (ii) the spoke lower bound (Lemma 1). Consider Fig. 8, which contains $\alpha^2 k$ points in a unit-spaced grid. MST cost of the points in the grid alone is $\alpha^2 k - 1$. Let k be the distance between r and the closest node in the grid. For capacity constraint k, the cost of an optimal solution would be $2\alpha^2 k - \alpha^2$, whereas the MST cost would be $(\alpha^2 + 1)k - 1$ and the spoke lower bound would be $\alpha^2 k$. This shows that with the current lower bounds, one cannot get a ratio any better than 2. It should be interesting to see whether we can find a unified lower bound by combining the MST cost and the spoke cost in a some way, instead of just analyzing them separately. We do not see a reason why our of ratio of 2.9 cannot be improved to 2.

References

1. K. Altinkemer and B. Gavish, *Heuristics with constant error guarantees for the design of tree networks,* Management Science **34**, pp. 331-341, 1988.
2. L.R. Esau and K.C. Williams, *On teleprocessing system design,* IBM Sys. Journal, **5**, pp. 142-147, 1966.
3. M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the. theory of NP-completeness,* W.H. Freeman, San Francisco, 1979.
4. B. Gavish and K. Altinkemer, *Parallel savings heuristics for the topological design of local access tree networks.,* Proc. IEEE INFOCOM, pp. 130-139, 1986.
5. M. Goemans and D. Williamson, *A General Approximation Technique for Constrained Forest Problems,* SIAM J. on Comput., **24**, pp. 296-317, 1995.
6. R. Hassin, R. Ravi and F.S. Salman, *Approximation algorithms for capacitated network design problems,* APPROX, pp. 167-176, 2000.
7. C. Monma and S. Suri, *Transitions in geometric minimum spanning trees,* Disc. Comput. Geom., **8**, pp. 265-293, 1992.
8. C.H. Papadimitriou, *The complexity of the capacitated tree problem,* Networks, **8**, pp. 217-230, 1978.
9. G. Robins and J. S. Salowe, *Low-degree minimum spanning trees,* Disc. Comput. Geom., **14**, pp. 151-166, 1995.

Fairness to All While Downsizing

Bala Kalyanasundaram* and Mahe Velauthapillai**

Computer Science Department,
Georgetown University,
Washington DC., USA
{kalyan,mahe}@cs.georgetown.edu

Abstract. Whenever changes are made to an existing schedule, it creates many logistical nightmares. This paper describes how to trim a schedule without making additional changes when we lose a resource. We consider a generalization of traditional scheduling of multiple identical resources (say broadcast channels, machines, memory, or power etc.) for jobs where the importance factor (or consumption rate) of a resource by jobs can vary with time. At any time the rate of consumption (or importance factor) of resource by two different jobs can differ. Given a schedule (or assignment) of k identical resources for n jobs, we consider the problem of fair reduction, loss, or downsizing of the resource from k to $k-1$ without time-wise altering the given schedule. Here, fairness means that every job loses roughly a fraction $1/k$ of the assigned resource (measured by the consumption rate or the importance factor) from the original schedule. We prove constructively a combinatorial Fairness Theorem that shows that a fair reduction for all jobs is possible for any schedule. The algorithm is simple and the resulting bound is almost tight since there is a trivial lower bound of loss of a fraction $1/k$ of assigned resource for some jobs.

1 Introduction

Consider the problem of scheduling k identical resources for n jobs. The schedule can be viewed as a matrix S with each row corresponding to a time step and each column corresponding to the k identical resources. For now, an entry in the matrix corresponds to at most one job. Since resources are identical, we allow swapping of entries within a row. Such exchanges are not considered to be changing the given schedule in the time domain (i.e., time-wise). Now we ask the following question: Given a schedule S, what happens when we lose a resource?

At each time step (i.e., a row), we delete an entry in S. That is, at each time step, any one of the k scheduled jobs can be chosen to lose the resource. Because the resources are identical, these deletions need not be all in the same column. Apart from these deletions, we do not allow any other changes in S. How can

* Supported in part by NSF under grant CCR-0098271 and Craves Family Professorship.
** Supported in part McBride Family funds.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 819–830, 2004.

we determine who loses the resource for each time unit such that our choice is fair for all jobs?

The importance of a resource for a job varies from time to time. As a consequence, the fairness perceived by a job depends on how it rates the availability of the resource at a given time. So, we allow each job to assign an importance factor (or consumption rate, processing rate) to the schedule S. As a consequence, we extend every entry of the schedule matrix to a pair (a, b), where a is the job and b is the importance factor that a assigns to the availability of the resource at this time. We can now measure the fairness based on the sum of all importance factors. That is, each job must roughly lose no more than $1/k$ of its original sum of all its importance factors.

Now we look at the benefit or cost of assigning resources to the set of all jobs from a global point of view. Even though an assignment of a resource to a job is beneficial to that job from local point of view, it may not be beneficial to the same degree from the global point of view. In order to capture the global benefit factor, we once again extend every entry of the schedule matrix to a triple (a, b, c) where c represents the global benefit factor for assigning the resource for job a at that time step. Now we ask the following question. Can we lose a resource such that each job incurs a fair loss mentioned above and the loss in global benefit factor is at most $1/k$th of its original total value?

Before we present our results, we motivate our problem with some examples. In classical job scheduling it may be the case that the availability of a processor (i.e., resource) to a job at time t or t' may not be significantly distinguishable. So jobs may treat the loss of processing power at different times identically. In this paper, we consider the case where the loss of resource significantly differs from time to time. For instance, loss of electrical power for homes at night is more unacceptable than during day time. In addition, the consumption of power during a day may be more than night. So, there need not be a direct correlation between the rate of consumption of resource and preferred resource availability. This is where the global benefit factor plays a critical role. Assume that the city is divided into k regions and the power company wants to reduce the total consumed power by assigning global benefit factor based on power consumption. Can we find a fair power-cut policy so that the total consumed power is reduced by a ! desired factor $1/k$?

The scheduling problem we consider is an extension of the traditional multiprocessor scheduling in the following two ways. We allow the rate of processing of a job to vary with time and, we also allow the rate of processing to differ from job to job at any time. This type of generalization arises in many context. For instance, in order to support high-rate data applications, next-generation wireless networks will allow sharing of data channels among different users (See Qualcomm's High Data Rate Scheme [1]). In this case, the effective bandwidth (service rate) of a channel $(f_i(t))$ for the user i at time t vary arbitrarily due to mobility, interference, distance, Raleigh fading etc. In addition, for two different users i and j, the service rates are not necessarily the same, i.e., $f_i(t)$ and $f_j(t)$ are not correlated.

Typically, a cellular tower has multiple (identical) channels allocated to data applications. Since the channels are identical, for any time interval, we can assign a job to any one of the k channels. However, moving the transmission schedule from one channel to another can be viewed as **not altering** the schedule in the time domain.

It is usually the case that the channels are shared among data and voice applications. Also, it is typical that channels are assigned to voice applications in preference to data applications. So, it is expected that data applications experience loss or gain of channels. This raises the following question: given a schedule of data transmission for n clients on k channels, is it possible to find a fair reduction of schedule in the data transmission of n clients on $k-1$ channels?

Even though the scheduling problems in the wireless scenario is an online problem, the model that we consider in this paper captures the off-line version of the problem.

Our next example is drawn from a context other than scheduling. Consider a company that has n different departments such as production, development, research, management etc. The company operates in many different cities and each city houses at most k different departments among the n possible departments. Due to poor economy, the company wants to downsize its operation. For political reasons, the company wants to eliminate one department per city. Based on the cost of operation of each department in each city, the company wants to downsize in such a way that its overall cost is reduced by a factor of $1/k$ using the global cost factor. However, each department of the company may view that its operation in one city may be more valuable than another. As a consequence, different departments may assign different weights (i.e., importance factor) on its operation on different cities. Given this situation, how can a company downsize fairly so that each department loses only a fair portion of its perceived value while the company reduces its cost (i.e. global cost factor) by a factor of $1/k$?

1.1 The Matrix Game

The combinatorial problem we consider in this paper can be formulated as a matrix game. Consider a matix $S_{r\times k}$ with r rows and k columns. The rows correspond to time units where r represents the length of the schedule and the columns correspond to (identical) machines. An entry of the matrix S is an ordered triple (a,b,c) where integer a is from $\{1,2,\ldots,n\}$, real b in the range [0,1] and c is a positive real number. In the case of the scheduling problem, a corresponds to a job, b corresponds to the rate of processing of the job at that time and c corresponds to the global benefit factor for processing at that time. Since the columns of the matrix correspond to k identical machines (or resources), k entries of any row can be permuted.

For a job $a \in \{1,2,\ldots,n\}$, we **define**

$$L_a^S = \sum_{i=1}^{i=r}\sum_{j=1}^{j=k}\sum_{S(i,j)=(a,b,c)} b \quad and \quad C^S = \sum_{i=1}^{i=r}\sum_{j=1}^{j=k}\sum_{S(i,j)=(a,b,c)} c.$$

In other words, L_a^S is the sum of b's from all entries (a, b, c) in S that correspond to job a. Also, C^S is the sum of c's from all entries of S. Also, let $b_a^{max} = max\{b : (a, b, c)\ in\ S\}$ be the maximum transmission rate for job a in S. The goal of the matrix game is to construct another matrix $T_{r \times k-1}$ with r rows and $k-1$ columns such that the ith row of T is formed by taking $k-1$ entries out of k entries of the ith row of S. However, in the global cost model, the matrix T must have the following properties:

$$\forall_a \quad L_a^T \geq \frac{k-1}{k} L_a^S - b_a^{max} O(1) \tag{1}$$

$$C^T \leq \frac{k-1}{k} C^S \tag{2}$$

Note that, for the global profit model, the inequality in the second condition will be reversed. However, if such T does not exist for every S, then is it possible to find T so that following condition is satisfied instead of the first condition?

$$\forall_a \quad L_a^T \geq \frac{k-1}{k} L_a^S - b_a^{max} f(k) \tag{3}$$

where f is some function of k alone.

Using re-scaling of b's for each job a independently, we will assume that $b_a^{max} = 1$ for all jobs a in S. This simplifies inequalities 1 and 3. From now on, when we refer to inequalities 1 and 3, the factor b_a^{max} is replaced by 1.

Definition 1. *Given a schedule matrix S, we say that a schedule matrix T is isomorphic to S if and only if each row of S is a permutation of the corresponding row of T. However, the permutation may vary from row to row.*

In this paper, we view that there is no change between two schedule matrix S and T in the time domain if S and T are isomorphic. We now introduce schedule matrix operators $\circ$ and $\star$ to represent column and row wise concatenation of schedule matrices.

Definition 2. *Given a schedule matrix S with r rows and k columns and another schedule matrix T with r rows and k' columns, we use $S \circ T$ to denote a new schedule matrix with r rows and $k + k'$ columns where the first k columns are from S and the last k' columns are from T.*

Definition 3. *Given a schedule matrix S with r rows and k columns and another schedule matrix T with r' rows and k columns, we use $S \star T$ to denote a new schedule matrix with $r + r'$ rows and k columns where the first r rows are from S and the last r' rows are from T.*

1.2 Previous Work and Our Results

There are numerous results on scheduling with faulty processors [5]. Our work in this paper differs fundamentally from all of the previous work that we are aware

of. The fundamental difference is the condition that the given schedule cannot be altered except for dropping parts of work on jobs due to the loss of resources. While dealing with the loss of resources/processing power, we are searching for a fair distribution of loss among all jobs without altering the schedule in the time domain. On the surface, our work has some similarity to the work on *cake-cutting problem* and its many variants [6]. But a closer look shows that these two problems are quite different. In our problem we have a fixed schedule, which is analogous to considering a previously cut cake that can not be cut anymore. We have to find a fair way to lose pieces of cake (or resource) such that each person loses only a fair share. On the other hand, the objective function in cake cutting problem is to cut the cake so that each person gets a fair share. This fundamental difference in the objective function makes the combinatorics quite different.

In section 2, we present a simple polynomial time algorithm based on Euler circuit construction to produce an isomorphic schedule matrix T for a given schedule matrix S of two resources for n jobs. We prove that $L_i^{T_1} \geq \frac{1}{2}L_i^S - \frac{1}{2}$ and $C^{T_1} \leq \frac{1}{2}C^S$. We show that this bound is *tight* for all values of L_i^S. We primarily present our results for global cost function C. The results and algorithms can be easily extended for global profit functions too.

In section 3, we present our main algorithm FAIR-FOR-ALL to deal with $k(\geq 3)$ resources. We establish that this algorithm produces a schedule T isomorphic to a given schedule S such that $L_i^{T_{1,k-1}} \geq \frac{k-1}{k}L_i^S - \frac{k-1}{2}$ and $C^{T_{1,k-1}} \leq \frac{k-1}{k}C^S$. This matches with the bound proposed in equation 3 where $f(k) = (k-1)/2$. When we establish the bound, we show that the bound obtained by the algorithm for various ranges of values for L_i^S. For $a < k$ and $a(a-1)/2 \leq L_i^S \leq a(a+1)/2$, we show that the bound is $L_i^{T_{1,k-1}} \geq \frac{a-1}{a}L_i^S - \frac{a-1}{2} \geq \frac{k-1}{k}L_i^S - \frac{k-1}{2}$. The reason for apparent decrease in fairness (i.e., $(a-1)/a$ instead of $(k-1)/k$) is due to the fact that $f(k) = \frac{k-1}{2}$ is large for small values of L_i^S. Recall that b values in the entry (a, b, c) of the matrix can be scaled for each job independently to increase L_i^S provided all such b values are less than equal to 1.

In section 4, we present a boosting algorithm that significantly reduces $f(k)$ to $(p-1)$ where p is the largest prime factor of k.

We have not presented other interesting features of our algorithm due to page limitations. For instance, our algorithm can also be used to produce fair schedules when more than one resource is lost. Our algorithm can also be used to balance the load on each resource. Here, balance means that each resource carries only a fair share of each and every job without time-wise altering the schedule. We conjecture that $f(k) = c$ where c is a small constant.

Due to space limitations, many proofs are omitted. It is important to recall that applying appropriate rescaling of benefit factor for each job a, we assume $b_a^{max} = 1$.

2 Tight Bounds for Two Resources Case

In this section we consider the case of exactly two identical resources. Suppose we are given a schedule matrix S of these two resources for n jobs.

Definition 4. *Given a schedule matrix S of two resources for n jobs, we can split the schedule into two schedule matrices S_1 and S_2, where S_1 (respectively S_2) is the schedule submatrix in S corresponding to the first (respectively the second) resource for n jobs. That is, $S = S_1 \circ S_2$.*

Lemma 1. *Given a schedule matrix S of two resources for n jobs, there exists another schedule matrix $T = T_1 \circ T_2$ that is isomorphic to S such that*

$$\forall_a \quad L_a^{T_1} \geq \frac{1}{2} L_a^S - \frac{1}{2} \quad \text{and} \quad \forall_a \quad L_a^{T_2} \geq \frac{1}{2} L_a^S - \frac{1}{2}.$$

Proof. We say that a row of the matrix S is homogeneous if the jobs involved in the schedule for the row are identical.

We will construct a schedule matrix X (respectively Y) that is isomorphic to the submatrix of S containing non-homogeneous (respectively homogeneous) rows and for every job a, $|L_a^{X_1} - L_a^{X_2}| \leq 1$ (respectively $|L_a^{Y_1} - L_a^{Y_2}| \leq 1$). We first describe how to construct the schedule matrix $T = T_1 \circ T_2$ using X and Y. For each job a, we do the following to construct the schedule matrix T_1. Add the schedule for job a in X_1 to T_1. If either $[(L_a^{X_1} \geq L_a^{X_2})$ AND $(L_a^{Y_2} \geq L_a^{Y_1})]$ or $[(L_a^{X_2} \geq L_a^{X_1})$ AND $(L_a^{Y_1} \geq L_a^{Y_2})]$ then add the schedule for a in Y_1 to T_1. Otherwise, add the schedule for a in Y_2 to T_1. Now all other schedules of a will be added to T_2. Repeat this for all jobs. The theorem then follows from the property of X and Y.

We first consider the construction of X. Given the set of non-homgenous rows of the schedule matrix S, we will construct a multi undirected graph $G = (V, E)$. We associate a pair of weights on every edge where each weight of the pair corresponds to each of the two corners of the edge. In other words, for each pair (i, e) where i is a vertex and e is an incident edge on i, we assign a weight. The construction of this graph G is given below:

1. Each job a, corresponds to a vertex $a \in V$.
2. For a non-homogenous row $[(a_1, b_1, c_1), (a_2, b_2, c_2)]$ of the schedule matrix S, we add an edge between two vertices that correspond to jobs a_1 and a_2. We associate weight b_1 (respectively b_2) to the end of the edge that is incident on vertex a_1 (respectively a_2). We call b_1 the weight of the edge corresponding to vertex a_1. We repeat this for every row and this may result in multiple edges between a pair of vertices. For the ease of presentation, we do not identify the row (i.e., time of schedule) that corresponds to the edge.

Note that the number of vertices with odd degree is even. Arbitrarily pair odd degree vertices and add a single edge with weights (0,0) between each pair. Now, the degree of each vertex is even. Now since the degree of each vertex is

even, there exists an Euler circuit going through each edge exactly once. We will use the construction of Euler circuit to construct a schedule matrix X that is isomorphic to the sub matrix of S containing non-homogeneous rows and for every job a, $|L_a^{X_1} - L_a^{X_2}| \leq 1$.

For each node i, sort the incident edges according to the corresponding weights. Since the number of incident edges are even, group the incident edges into pairs such that adjacent edges according to the sorted sequence are paired together (i.e., first and second are in one pair and so on). We refer the set of pairs of incident edges for vertex i by P_i.

We will now describe a process to partition the graph as the union of disjoint cycles, where for each cycle and for each vertex i, adjacent edges of the cycle that share the vertex i are a pair in P_i. The cycles that the following process describe may not be simple cycles, that is, vertex may repeat many times in the cycle.

Start with a vertex i, say $i = 1$. Pick an edge (i, j) from P_i. The process of forming a cycle ends when we return to vertex i through the edge (k, i) that forms a pair with our initial edge (i, j) in P_i. Whenever we reach a vertex k through some other edge (ℓ, k), we leave k through another edge paired with (k, m) in P_k. Since all edges are paired for every vertex, we are guaranteed to return to the starting vertex i through the matching pair. For all vertices j, remove all the pairs involved in the cycle from P_j and repeat the process until there are no more edges.

Now, given a cycle we show how to build the schedules X_1 and X_2. Recall that there is a unique time step associated with each edge. So it suffices to say for each edge (i, j) which vertex is assigned to X_1 and which one to X_2. Given a cycle, imagine a walk along the cycle. For each edge (i, j) of the cycle, we leave a vertex i (say) and reach a vertex j (say) along the walk. Assign i to X_1 and j to X_2. By doing so, observe that for every vertex i and every pair of edges in P_i, i is included in X_1 for one edge of the pair and i in included in X_2 for the other edge of the pair. Recall that for each vertex, pairs are formed by sorting the edges according to the weights corresponding to the vertex. Also, recall that weights are nothing but b values of the original schedule matrix S. Therefore, each weight is from the range [0,1]. As a consequence, for each vertex (aka. each job) i, the sum of corresnonding weights in X_1 and X_2 differ by no more than 1. In other words, $|L_i^{X_1} - L_i^{X_2}| \leq 1$.

We now consider how to construct a schedule matrix Y that is isomorphic to the sub matrix of S and for all jobs a $|L_a^{Y_1} - L_a^{Y_2}| \leq 1$. Let a be the job under consideration. Let Z_a be the set of isomorphic rows in S that correspond to job a. Consider a row at a time from Z_a. We maintain a count α_1 (respectively α_2) during the construction of Y_1 (respectively Y_2). They are initially set to zero. Let $[(a, b_1, c_1), (a, b_2, c_2)]$ be the homogeneous row in S under consideration. Without loss of generality, assume $b_1 \geq b_2$. If $\alpha_1 \leq \alpha_2$ then we add (a, b_1, c_1) to T_1. Otherwise, add (a, b_2, c_2) to T_1. In either case, add the other one to T_2. Now update α_1 and α_2 by adding corresponding b values. At the end $|\alpha_1 - \alpha_2| \leq 1$ since b values are from the range $[0, 1]$. The result follows since, for $i = 1, 2$, $L_a^{Y_i}$ is nothing but α_i at the end.

Theorem 1. *Given a schedule matrix S of two resources for n jobs, we can construct an isomorphic schedule matrix T in polynomial time such that*

$$\forall_a \quad L_a^{T_1} \geq \frac{1}{2}L_a^S - \frac{1}{2} \quad \text{and} \quad C^{T_1} \leq \frac{1}{2}C^S.$$

Proof. Apply lemma 1 to get T_1 and T_2. Observe that $C^{T_1} + C^{T_2} = C^S$. Soif $C^{T_2} < C^{T_1}$ then swap T_1 and T_2. The result then follows.

The following lemma shows that the upper bound we established before is tight.

Lemma 2. *Let $\alpha > 1$. There exists a schedule matrix S with $\forall_a \; L_a^S = \alpha$ such that for any isomorphic schedule matrix T, there exists a job a such that $L_a^{T_1} \leq (1/2)[L_a^S - 1]$. This lower bound exactly matches the upper bound.*

Proof. Let $\alpha > 1$, let ℓ be the largest odd integer such that $\alpha = \ell + \epsilon$, where $0 \leq \epsilon < 2$. Let a, b, c, d be four jobs which will have the following schedule on the two channels.

1. Row $[(a,1,1),(b,1,1)]$ repeats for the first ℓ times in the schedule matrix.
2. Row $[(c,1,1),(d,1,1)]$ repeats for the next ℓ times in the schedule matrix.
3. The next four rows are: $[(a,\epsilon/2,1),(c,\epsilon/2,1)]$, $[(a,\epsilon/2,1),(d,\epsilon/2,1)]$, $[(b,\epsilon/2,1),(c,\epsilon/2,1)]$, $[(b,\epsilon/2,1),(d,\epsilon/2,1)]$.

If you lose one resource, since ℓ is odd, either a or b can be scheduled at most $(\ell-1)/2$ units of time. Similarly c or d can be scheduled at most $(\ell-1)/2$ units of time (see steps (1) and (2) above). Without loss of generality, assume that a and c are scheduled at most $(\ell-1)/2$ units of time. Then either a is scheduled for ϵ units of time or c is scheduled for ϵ units of time (see step (3) above) but not both. As a consequence, at least one of a or c can be scheduled for at most $(\ell-1)/2 + \epsilon/2 = (\alpha-1)/2$ units of time.

3 The General Case and FAIR-FOR-ALL Algorithm

In this section we consider the case where there are $k(\geq 3)$ identical resources. We present a simple recursive algorithm, which we call FAIR-FOR-ALL. This algorithm uses the algorithm that we have presented in the previous section for $k = 2$ case. The FAIR-FOR-ALL algorithm, takes two parameters as input. The first input parameter is a schedule matrix S and the second input parameter k is the number of resources involved in S. The algorithm returns a schedule matrix isomorphic to the input matrix S.

Recall the definition that $S = T_1 \circ T_2$ means that the matrix S is formed by concatenating (column-wise) the two matrices T_1 and T_2.

Definition 5. *Given a schedule matrix S of k identical resources for n jobs, we define $S_{i,j}$ to be the sub-matrix of S that contains columns i through j. In order to minimize the use of subscripts, we also define S_i to be the sub-matrix $S_{i,i}$ which contains just ith column of S.*

FAIR-FOR-ALL(**S,k**) **Algorithm**

Recall that an entry of S is of the form (a, b, c). Let m be the maximum number of bits used in the representation of b's in S. Finally, let m' be the smallest integer such that for jobs a, $L_a^S \leq 2^{m'}$.
We renumber (or rearrange) the columns of S such that $C^{S_1} \leq C^{S_2} \leq \ldots \leq C^{S_k}$.

For $i = 1$ **to** $3(m + m')$ **repeat steps (1) and (2)**

1. Apply the algorithm in the proof of Lemma 1 to the schedule matrix $S_{k-1,k}$ and let T_1 and T_2 be the two matrices returned by the algorithm where T_1 has the prefered property specified in Theorem 1. Reset $S = S_{1,k-2} \circ T_1 \circ T_2$.
2. Recursively apply FAIR-FOR-ALL$(S_{1,k-1}, k-1)$ algorithm. Let R be the matrix returned by the algorithm. Reset $S = R \circ S_k$.

End For
Repeat step (1) once and return S.

End FAIR-FOR-ALL

When the algorithm terminates, it returns a schedule matrix S that is isomorphic to the original schedule matrix and $S_{1,k-1}$ has the desired fairness property. Before we analyze the algorithm to establish the desired property, we introduce some notations and observations.

Observe that after every iteration of the loop, the resultant schedule matrix S is isomorphic to the original schedule matrix. In addition, after the first step inside the loop, the resultant schedule matrix S is also isomorphic to the original schedule matrix.

Definition 6.
1. Suppose S is the resultant schedule matrix after the end of t^{th} *iteration of the loop in the algorithm. We define* $L_j^i(t) = L_j^{S_{1,i}}$.
2. Also, we define $L_j^i(t + 1/2) = L_j^{S_{1,i}}$ *where S is the resultant schedule matrix after the end of the first step during* $(t+1)st$ *iteration of the loop.*

From the above definition, whenever we want to refer to the property of the original schedule matrix, we set $t = 0$ (e.g., $L_i^k(0)$). Similarly, whenever we want to refer to the property of the schedule matrix at the end of the algorithm, we set $t = \infty$ (i.e., $L_i^j(\infty) = L_i^j(3(m + m') + 1/2)$).

Before we analyze the algorithm, we will provide some intuition. As we iterate through the loop of the algorithm, the fairness of a job with respect to the first $k-2$ resources, measured by $L_i^{k-2}(.)$, will fluctuate. We set a threshold, call it β for now, such that if $L_i^{k-2}(.) \geq \beta$ at the end of the last iteration, then applying step 1 of the algorithm we can establish that fairness is achieved for job i at the end of the algorithm. We will prove this in Lemma 4. However, this must be true for all jobs.

In order to show that $L_i^{k-2}(.)$ reaches the threshold before the algorithm terminates, we show that the difference $\beta - L_i^{k-2}(.)$ decreases by a factor of 3/4

for every iteration of the loop in Lemma 5. Since the minimum decrease is at least 2^{-m} and maximum value of $L_i^k(0) \leq 2^{m'}$, $L_i^{k-2}(.)$ must reach the threshold on or before $3(m+m')$ iterations of the loop.

But, what if the $L_i^{k-2}(.)$ reaches the threshold at the start of some iteration and later (say later iteration) fell below it. In Lemma 3, we show that once $L_i^{k-2}(.)$ reaches the threshold, it never falls below it (eventhough it may decrease).

Finally, in order to establish the fact that the global cost is reduced by a factor of $1/k$, we prove in Lemma 6 that at the start of the iteration of main loop, $C^{S_k} \geq (1/k)C^S$ and $C^{S_{k-1}} \geq (1/(k-1))C^{S_{1,k-1}}$. The second part of the condition is used in establishing the first part of the condition during the next iteration (i.e., step 1).

We are ready to state and prove the main Theorem, which we call *Fairness Theorem.*

Theorem 2. *Let* $(k > 1)$ *be identical channels and a an integer. For all jobs* j *if* $2 \leq a \leq k$ *and* $a(a-1)/2 \leq L_j^k(0) \leq a(a+1)/2$ *then by* FAIR-FOR-ALL

$$L_j^{k-1}(\infty) \geq \frac{(a-1)}{a} L_j^k(0) - (a-1)/2$$

The proof of this theorem will follow from the next four lemmas. Note that the assumption (if $2 \leq a \leq k$ and $a(a-1)/2 \leq L_j^k(0) \leq a(a+1)/2$) that holds true for the theorem also holds true for the next four lemmas. It should be noted that the proof of these lemmas for the case of k resources make use of the Theorem 2 for $k-1$ resources. Thus, there is a global inductive step that binds the four lemmas.

First, the following lemma shows that the bound on L for a job stays above a threshold, once it crosses the threshold.

Lemma 3. *If* $L_i^{k-2}(t) \geq \frac{(a-2)}{a} L_i^k(0) - (a-2)$ *then* $L_i^{k-2}(t+1) \geq \frac{(a-2)}{a} L_i^k(0) - (a-2)$.

We then establish in the next lemma that the desired fairness for a job is obtained if threshold is met at the end of the last iteration.

Lemma 4. *If* $L_i^{k-2}(3(m+m')) \geq \frac{(a-2)}{a} L_i^k(0) - (a-2)$ *then* $L_i^{k-1}(\infty) \geq \frac{(a-1)}{a} L_i^k(0) - \frac{(a-1)}{2}$.

The next lemma shows that the desired threshold is met for every job on or before the end of $3(m+m')$th iteration of the loop.

Lemma 5. *For all* i, *there exists a step* $t \leq 3(m+m')$ *such that* $L_i^{k-2}(t) \geq \frac{(a-2)}{a} L_i^k(0) - (a-2)$.

Finally, we establish the bound on the global cost function in the following lemma.

Lemma 6.
1. At the start of every iteration of the main loop and after the first step
$$C^{S_k} \geq (1/k)C^S.$$
2. At the start of every iteration of the main loop
$$C^{S_{k-1}} \geq (1/(k-1))C^{S_{1,k-1}}.$$

4 Further Boosting

In this section we further improve the bounds that we obtained in Theorem 2 when the number of resources (i.e., k) is not a prime number. We improve the bound on $f(k)$ from $\frac{k-1}{2}$ to $\frac{(p-1)(p+2)}{2p} \leq (p-1)$ where p is the largest prime factor of k.

FAIR-FOR-ALL-BOOSTER(**S,k**) **Algorithm**

Let $k = p_1 \times p_2 \cdots \times p_s$ where p_i is a prime number and $\forall_i \ \ p_i \leq p_{i+1}$. Let $A = S$ and $z = k$. Let r be the number of rows of A. Let R be the matrix that this algorithm will return. Initially it is set to be empty.

For $p = p_1, p_2, \ldots, p_s$ **repeat steps (1) and (2)**

1. For $1 \leq i \leq z/p$, recall that $A_{(i-1)p+1,ip}$ is a submatrix of A with exactly p columns. We now construct a new matrix $B = A_{1,p} \star A_{p+1,2p} \star \cdots \star A_{z-p+1,z}$. Observe that B has exactly p columns and $r*(z/p)$ rows. Apply FAIR-FOR-ALL(B, p). Let D be matrix isomorphic to B returned by FAIR-FOR-ALL.
2. Now break the $(p-1)$-colums matrix $D_{1,p-1}$ into (z/p) submatrices $D(1), D(2), \ldots, D(z/p)$, where $D(1)$ contains the first r rows of $D_{1,p-1}$, $D(2)$ contains the second r rows and so on. $R = R \circ D(1) \circ D(2) \circ \cdots \circ D(z/p)$. Now break the one column matrix D_p into $E(1), E(2), \ldots, E(z/p)$ where $E(1)$ contains the first r rows of D_p and so on. Reset $A = E(1) \circ E(2) \circ \cdots \circ E(z/p)$ and $z = z/p$.

End For
Reset $R = R \circ A$ and return R.

End FAIR-FOR-ALL-BOOSTER

Theorem 3. *Let $k = p_1 \times p_2 \cdots \times p_s$ where $p_i (\geq 2)$ is a prime number and $\forall_i \ \ p_i \leq p_{i+1}$. Given a schedule matrix S of k resources for n jobs,* FAIR-FOR-ALL-BOOSTER *returns an isomorphic schedule matrix T such that*

$$\forall_a \quad L_a^{T_{1,k-1}} \geq \frac{k-1}{k} L_a^S - \frac{(p_s-1)(p_s+2)}{2p_s} \geq \frac{k-1}{k} L_a^S - (p_s - 1)$$

and $C^{T_{1,k-1}} \leq \frac{k-1}{k} C^S$.

Proof. After i iterations of the for loop we claim that the following invariant is true:

$$\forall_j \quad L_j^A \leq \frac{1}{p_1 p_2 \ldots p_i} L_j^S - \frac{p_i - 1}{2}[1 + \frac{1}{p_i} + \frac{1}{p_i p_{i-1}} + \ldots + \frac{1}{p_i p_{i-1} \ldots p_2}]$$

and $C^A \geq \frac{1}{p_1 p_2 \ldots p_i} C^S$. Assuming that the invariant is true for i, we will argue that it is also true for $i+1$. Observe that the execution of FAIR-FOR-ALL during $i+1$st iteration will yield

$$L_j^E \leq \frac{1}{p_{i+1}} L_j^A + \frac{p_{i+1} - 1}{2}.$$

The claim of the invariant then follows by substituting the bound for L_j^A and observing $p_j \leq p_{j+1}$. The result then follows since

$$[1 + \frac{1}{p_s} + \frac{1}{p_s p_{s-1}} + \ldots + \frac{1}{p_s p_{s-1} \ldots p_2}] \leq 1 + \frac{2}{p_s} \leq 2.$$

References

1. P. Bender, M. Black, R. Padovani, N. Sindhushyana, and A. Verterbi. Cdma/hdr: A bandwidth efficient high speed wireless data service for nomadic users. In *IEEE Communications Magazine,* July 2000.
2. S. Borst and P. Whiting. Dynamic rate control algorithms for hdr throughput optimization. In *IEEE INFOCOM,* April 2001.
3. J.J. Caffery and G. L. Stuber. Overview of radiolocation in cdma cellular systems. In *IEEE Communications Magazine,* April 1998.
4. N. Joshi, S.R. Kadaba, S. Patel, and G. S. Sundaram. Downlink Scheduling in Cdma Networks. In *MobiCom,* 2000.
5. B. Kalyanasundaram and K.R. Pruhs. Fault Tolerent Scheduling. In *STOC,* pages 115–124, 1994.
6. J. Robertson and W. Webb. *Cake-Cutting Algorithms.* Peters, A. K. Limited, 1998.

A Generalisation of Pre-logical Predicates to Simply Typed Formal Systems

Shin-ya Katsumata

Laboratory for Foundations of Computer Science
School of Informatics, The University of Edinburgh,
King's Buildings, Edinburgh EH9 3JZ, UK
S.Katsumata@sms.ed.ac.uk

Abstract. We generalise the notion of pre-logical predicates [HS02] to arbitrary simply typed formal systems and their categorical models. We establish the basic lemma of pre-logical predicates and composability of binary pre-logical relations in this generalised setting. This generalisation takes place in a categorical framework for typed higher-order abstract syntax and semantics [Fio02,MS03].

1 Introduction

Pre-logical predicates (relations) [HS02] are a generalisation of logical predicates. They are defined for the *simply typed lambda calculus* and its set-theoretic environmental models called *lambda applicative structures* [Mit96]. Two important properties are enjoyed by pre-logical predicates but not logical predicates. One is that pre-logical predicates are *equivalent* to predicates satisfying the basic lemma (interpretation of all terms respects predicates — this is the key to many applications of logical relations), and the other is that binary pre-logical relations are closed under relational composition.

We aim to generalise pre-logical predicates from the simply typed lambda calculus to arbitrary *simply typed formal systems* (we just say *typed formal system* below) and their categorical models, then show that the above important properties hold in this generalised setting.

This generalisation enables us to extend pre-logical predicates systematically to other calculi, such as lambda calculus with various type constructors and variable binders, and calculi other than lambda calculus, such as logics and process calculi. This opens up the possibility of characterising observational equivalence [HS02] and constructive data refinement [HLST00] in various non-lambda calculi.

There are three underlying elements on which pre-logical predicates are defined: syntax (normally the simply typed lambda calculus), semantics (set-theoretic environmental models) and predicates (as subsets of carrier sets). We generalise these three elements along the following dimensions:

- We generalise syntax to an arbitrary typed formal system described by a *typed binding signature* [MS03]. A typed formal system is a formal system whose inference rules fit within the following scheme:

$$\frac{\Gamma, \Gamma_1 \vdash M_1 : \tau_1 \quad \cdots \quad \Gamma, \Gamma_m \vdash M_m : \tau_m}{\Gamma \vdash o(\Gamma_1.M_1, \cdots, \Gamma_m.M_m) : \tau}$$

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 831–845, 2004.

This is general enough to subsume various simple type systems and calculi such as the simply typed lambda calculus, many-sorted first-order logic, pi-calculus, etc.
- We generalise from set-theoretic to category-theoretic semantics. Following the principle of categorical semantics, we give a semantics of a typed formal system in a Cartesian category $\mathbb{C}$ by mapping types to objects and terms to morphisms in $\mathbb{C}$.
- As we move to category theory, we need to change the notion of predicates from subsets to appropriate category-theoretic constructs. We use *subscones,* which is a mild generalisation of the injective scones of [MS93].

We represent all three elements as objects and morphisms in the category of *presentation models* $\mathbf{M}_T$, where T is the set of types [MS03]. In this category, the collection of well-formed terms modulo α-equivalence is represented as the initial algebra of the endofunctor corresponding to a typed binding signature.

After this generalisation, we formulate pre-logical predicates and predicates satisfying the basic lemma, and show their equivalence. Then we show the composability of binary pre-logical relations.

We look at three examples of pre-logical predicates, i) the relationship between pre-logical predicates for combinatory algebra and those for lambda calculus, ii) the connection between pre-logical predicates and lax logical predicates [PPST00] and iii) a characterisation of elementary submodels of first-order classical logic by a pre-logical relation.

Structure of This Paper. The generalisation of pre-logical predicates takes place in the following way. In section 2, we first introduce a category of presentation models $\mathbf{M}_T$, and typed binding signatures as a description of typed formal systems. We give a categorical semantics of typed formal systems in a very weak sense. We introduce a formulation of predicates with respect to this semantics, using subscones. All three elements (syntax, semantics and predicates) are expressed in category $\mathbf{M}_T$. Then we formulate pre-logical predicates and predicates satisfying the basic lemma in section 3. The basic idea of the formulation of pre-logical predicates is that the inverse image of a predicate along the meaning function has an algebra structure. We show that predicates satisfying the basic lemma and pre-logical predicates are equivalent. Composition of binary pre-logical relations is discussed in section 4. In section 5, we look at three examples of pre-logical predicates. Proofs are attached in the appendix.

Related Work. First, we briefly consider formulations of logical predicates. Logical predicates (and relations) have been widely used as a tool to study properties of the simply typed lambda calculus. Ma and Reynolds [MR92] formulated logical predicates as Cartesian closed functors from the free CCC $\mathbf{L}$ to $\mathbf{Pred}(G)$. Hermida [Her93] pointed out that $\mathbf{Pred}(G)$ can be replaced with the total category $\mathbb{E}$ of a fibration $p : \mathbb{E} \to \mathbb{B}$, provided that CCC structures on $\mathbb{B}$ can be lifted to $\mathbb{E}$. Plotkin et al. [PPST00] introduced a weakening of Ma and Reynolds' formulation called *Lax logical predicates,* which are functors from $\mathbf{L}$ to $\mathbf{Pred}(G)$ preserving only finite products. The basic lemma still holds for lax logical predicates, and furthermore the converse holds. In this sense lax logical predicates and pre-logical predicates are the same. They extended lax logical

predicates from the lambda calculus to the language described by a finitary monad over Cat extending finite product structure.Lax logical predicates are also extended to the computational lambda calculus [KP99]. Binary lax logical relations are closed under composition.

Kinoshita, et al. [KOPT97] proposed a generalisation of logical relations called *L-relations*. Their framework is also parameterised by a finitary monad L over **Cat**, which allows us to generalise the language from the lambda calculus. They used category objects in **Cat** to formulate the composition of L-relations.

Leiß[Lei01] extended pre-logical predicates to system $F\omega$, and characterised observational equivalence in terms of existence of binary pre-logical relation.

An application of binary pre-logical relations is to characterise observational equivalence between two models of a language [Lei01,HS02,Kat03].

This work refers to the framework by Fiore [Fio02] and Miculan and Scagnetto [MS03] on a categorical model of typed higher-order abstract syntax. This framework is a natural extension of the one considered in [FPT99,Hof99] to take types into account.

Convention. We identify a set and its discrete category. We assume that all categories appeared in this paper are locally small. By a Cartesian category we mean a category with chosen finite products. We fix a countably infinite set of variables X (ranged over by x, y, z). For a finite set A, by $|A|$ we mean the number of elements in A. We use $\vec{A}$ for a sequence of meta variables, like $A_1, \cdots, A_n$.

2 Preliminaries

Category of Presentation Models. We introduce the category of *presentation models* [MS03] plus some auxiliary categories for the following sections. We represent all three elements involved in the notion of pre-logical predicates (syntax, semantics and predicates) in this category.

Let T be the set of types, whose elements are ranged over by τ, σ. A *context* (ranged over by Γ) is a function from a finite subset of X to T. A *context renaming* from Γ to Γ' is a function $f : \mathrm{dom}(\Gamma) \to \mathrm{dom}(\Gamma')$ such that $\Gamma' \circ f = \Gamma$. They form the *category of contexts* $\mathbf{C}_T$[1] with the initial object given by the empty context $! : \emptyset \to T$ and binary coproducts given by cotupling of contexts $[\Gamma, \Gamma'] : \mathrm{dom}(\Gamma) + \mathrm{dom}(\Gamma') \to T$. By Γ, Γ' we mean the coproduct of contexts Γ and Γ' whose domains are disjoint. We fix a variable $x \in X$ and define $\langle - \rangle : T \to \mathbf{C}_T$ by $\langle \tau \rangle = \{x \mapsto \tau\}$. We assume that each variable $x \in \mathrm{dom}(\Gamma)$ has an index number denoted by $\gamma(x) \in \{1, \cdots, |\mathrm{dom}(\Gamma)|\}$.

We define the *ambient category* $\mathbf{S}_T = \mathbf{Set}^{\mathbf{C}_T}$. Category $\mathbf{S}_T$ has small limits and colimits, and has a *context extension operator* $(\delta_\tau A)(\Gamma) = A(\Gamma + \langle \tau \rangle)$. In fact $\delta_\tau A$ is isomorphic to $V_\tau \Rightarrow A$, where $V_\tau(\Gamma)$ is the presheaf of *variables of type* τ, defined to be $\mathbf{C}_T(\langle \tau \rangle, \Gamma) \cong \{x \mid \Gamma(x) = \tau\}$, thus δ_τ has a left adjoint. Moreover it has a right

[1] Category $\mathbf{C}_T$ can be described as the comma category $I \downarrow T$ where $I : \mathbf{X} \hookrightarrow \mathbf{Set}$ is the inclusion functor of the full subcategory $\mathbf{X}$ of **Set** whose objects are finite subsets of X [Fio02,MS03]. It is a free co-Cartesian category generated from T.

adjoint ([MS03], proposition 2), thus preserves both limits and colimits. We write $\delta_{\vec{\tau}}$ for the composition $\delta_{\tau_1} \circ \cdots \circ \delta_{\tau_n}$.

The category of *presentation models* $\mathbf{M}_T$ is defined to be $(\mathbf{S}_T)^T \cong \mathbf{Set}^{\mathbf{C}_T \times T}$. It also has small limits and colimits.

Syntax: Typed Binding Signature. A *typed binding signature* (ranged over by Π) is a tuple (T, O) where T is the set of *types* (ranged over by τ, σ) and O is the set of *operators* (ranged over by o), each of which is a pair of an *operator symbol* s and its *arity* $((\vec{\tau_1}, \sigma_1), \cdots, (\vec{\tau_m}, \sigma_m), \tau) \in (T^+)^* \times T$. We write $s^{(\vec{\tau_1},\sigma_1),\cdots,(\vec{\tau_m},\sigma_m)\to\tau}$ for such a pair in O [2], and $o^\tau \in O$ for an operator whose result type is τ. A *typed first-order signature* (ranged over by Σ) is just a typed binding signature (T, O) such that for all $s^{(\vec{\tau_1},\sigma_1),\cdots,(\vec{\tau_m},\sigma_m)\to\tau} \in O$, $\vec{\tau_i} = \epsilon$. It coincides with the notion of many-sorted signature.

A typed binding signature Π specifies a *typed formal system.* We first define *raw-Π term* (ranged over by M, N) by the BNF $M ::= x^\tau \mid o(\vec{x^\tau}.M, \cdots, \vec{x^\tau}.M)$. In this BNF, $\vec{x^\tau}.M$ means binding of variables $\vec{x^\tau}$ in M. As usual, we identify α-**equivalent** terms. The typed formal system is a system to derive judgment $\Gamma \vdash_\Pi M : \tau$, where Γ is an object in $\mathbf{C}_T$. The system consists of the following rules for variables and operators.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash_\Pi x^\tau : \tau} \qquad \frac{\Gamma, \overrightarrow{x_1 : \tau_1} \vdash_\Pi M_1 : \sigma_1 \quad \cdots \quad \Gamma, \overrightarrow{x_n : \tau_n} \vdash_\Pi M_m : \sigma_m}{\Gamma \vdash_\Pi s^{(\vec{\tau_1},\sigma_1),\cdots,(\vec{\tau_m},\sigma_m)\to\tau}(\overrightarrow{x_1^{\tau_1}}.M_1, \cdots, \overrightarrow{x_m^{\tau_m}}.M_m) : \tau}$$

Example 1. 1. Let B be a set. By $\mathbf{Typ}^{\Rightarrow}(B)$ we mean the set defined by the BNF $\tau ::= b \mid \tau \Rightarrow \tau$ where $b \in B$. The typed binding signature Π_λ for the simply typed lambda calculus is defined to be $(\mathbf{Typ}^{\Rightarrow}(B), \{\mathsf{lam}^{(\tau,\tau')\to\tau\Rightarrow\tau'}, \mathsf{app}^{\tau\Rightarrow\tau',\tau\to\tau'}\})$ where τ, τ' ranges over $\mathbf{Typ}^{\Rightarrow}(B)$.

2. The typed first-order signature for combinatory logic is

$$\Sigma_{CL} = (\mathbf{Typ}^{\Rightarrow}(B), \{\mathsf{app}^{\tau\Rightarrow\tau',\tau\to\tau'}, \mathsf{S}^{(\tau\Rightarrow\tau'\Rightarrow\tau'')\Rightarrow(\tau\Rightarrow\tau')\Rightarrow\tau\Rightarrow\tau''}, \mathsf{K}^{\tau\Rightarrow\tau'\Rightarrow\tau}\})$$

   where τ, τ', τ'' ranges over $\mathbf{Typ}^{\Rightarrow}(B)$.

3. Let $\Sigma = (T_0, O_0)$ be a typed first-order signature. The typed binding signature for first-order classical logic over Σ is

$$\Pi_{\Sigma\text{-fol}} = (T_0 \cup \{\Omega\}, O_0 \cup \{\mathsf{exists}^{(\tau,\Omega)\to\Omega} \mid \tau \in T_0\} \cup \{\mathsf{not}^{\Omega\to\Omega}, \mathsf{or}^{\Omega,\Omega\to\Omega}\})$$

The typed formal system described by Π determines an object $S_\Pi(\tau)(\Gamma) = \{M \mid \Gamma \vdash_\Pi M : \tau\}$ in $\mathbf{M}_T$. This object can be characterised as an initial algebra of the functor associated to Π by $(\Pi A)\tau = V_\tau + \coprod_{s^{(\vec{\tau_1},\sigma_1),\cdots,(\vec{\tau_m},\sigma_m)\to\tau} \in O} \left(\prod_{i=1}^m \delta_{\vec{\tau_i}}(A\sigma_i)\right)$ together with the Π-**algebra** structure $\iota_\Pi : \Pi S_\Pi \to S_\Pi$ corresponding to the inference rules ([MS03], theorem 1).

[2] This definition of typed binding signature is a special case of the one in [MS03] where the set of types allowed for variables is equal to the set of all types.

Semantics: Very Weak Categorical Model. We formulate a semantics of a typed formal system $\Pi = (T, O)$ by a morphism to the object in $\mathbf{M}_T$ which reflects a Cartesian category. The notion of semantics considered here is very weak in the sense that it does not exploit any categorical structure other than finite products. The semantics keeps the basic principle of categorical model theory: that is, types are interpreted as objects and terms are interpreted as morphisms.

An *interpretation of types* is just a functor $F : T \to \mathbb{C}$ where $\mathbb{C}$ is a Cartesian category. We extend it to the functor $F^* : \mathbf{C}_T \to \mathbb{C}^{op}$ by $F^*\Gamma = \prod_{i=1}^{|\mathrm{dom}(\Gamma)|} F(\Gamma(\gamma^{-1}(i)))$, which preserves finite products in $(\mathbf{C}_T)^{op}$. We write $s_{\Gamma,\Gamma'} : F^*\Gamma \times F^*\Gamma' \to F^*(\Gamma + \Gamma')$ for the natural isomorphism. For an interpretation of types $F : T \to \mathbb{C}$, we define the *clone of typed operations* H^F by $H^F(\tau)(\Gamma) = \mathbb{C}(F^*\Gamma, F\tau)$. Let $\mathbb{D}$ be a Cartesian category. For a functor $G : \mathbb{C} \to \mathbb{D}$ preserving finite products strictly, we define a morphism $H^G : H^F \to H^{GF}$ in $\mathbf{M}_T$ by $H^G(\tau)(\Gamma)(f : F^*\Gamma \to F\tau) = Gf : (GF)^*\Gamma = GF^*\Gamma \to GF\tau$. A *categorical interpretation of* Π consists of a Cartesian category $\mathbb{C}$, an interpretation of types $F : T \to \mathbb{C}$ and a morphism $m : S_\Pi \to H^F$ in $\mathbf{M}_T$ called *interpretation of terms,* which assigns to a well-formed term $\Gamma \vdash_\Pi M : \tau$ a morphism $m(M) : F^*\Gamma \to F\tau$. We use the notation $\mathbb{C}[\![-]\!]_F$ to represent a categorical interpretation. We define the *product* of categorical interpretations $(\mathbb{C}, F_1, m_1), (\mathbb{D}, F_2, m_2)$ to be $(\mathbb{C} \times \mathbb{D}, \langle F_1, F_2\rangle, (m_1, m_2))$ where (m_1, m_2) is defined by $(m_1, m_2)(\tau)(\Gamma) = (m_1(\tau)(\Gamma), m_2(\tau)(\Gamma))$.

Often, H^F is equipped with a Π-**algebra** structure. In this case we can obtain an interpretation of terms by the initiality of S_Π. This is the *initial algebra semantics* for typed binding signature ([FPT99,MS03]).[3]

To specify a Π-**algebra** structure over H^F, it is sufficient to specify a morphism $u_o : \prod_{i=1}^{m} \delta_{\vec{\tau_i}}(H^F\sigma_i) \to H^F\tau$ in $\mathbf{S}_T$ for each operator $o \in O$ of arity $(\vec{\tau_1}, \sigma_1), \cdots, (\vec{\tau_m}, \sigma_m) \to \tau$. Together with the mapping $v^\tau : V_\tau \to H^F\tau$ defined to be $v^\tau(\Gamma)(x) = \pi_{\gamma(x)}$, we obtain a Π-**algebra** structure over H^F by $[v^-, u_{o^-}]_{o^- \in O} : (\Pi H^F) \to H^F$.

Example 2. (Continued from example 1)

1. Let $F_\lambda : \mathbf{Typ}^{\Rightarrow}(B) \to \mathbb{C}$ be an interpretation of types satisfying $F_\lambda(\tau \Rightarrow \tau') = F_\lambda(\tau) \Rightarrow F_\lambda(\tau')$. The morphisms in $\mathbf{S}_T$ given by $(u_{\mathsf{lam}^{(\tau,\tau')\to\tau\Rightarrow\tau'}})_\Gamma(f) = \lambda(f \circ s_{\Gamma,\langle\tau\rangle})$ and $(u_{\mathsf{app}^{\tau\Rightarrow\tau',\tau\to\tau'}})_\Gamma(f, g) = @ \circ \langle f, g\rangle$ yield a Π_λ-**algebra** structure over H^{F_λ}. The initial algebra semantics coincides with the standard semantics of the simply typed lambda calculus in $\mathbb{C}$.
2. Let $\Sigma = (T_0, O_0)$ be a typed first-order signature. A *many-sorted* Σ-*algebra* $\mathcal{A}$ consists of a T_0-**indexed** family of sets $A : T_0 \to \mathbf{Set}$ called *carrier sets* and an assignment of a function $o_\mathcal{A} : A^{\tau_1} \times \cdots \times A^{\tau_n} \to A^\tau$ to each operator $o \in O_0$ of arity $\tau_1, \cdots, \tau_n \to \tau$.
   To each operator $o \in O_0$, we assign a morphism $(u_o)_\Gamma(f_1, \cdots, f_n) = o_\mathcal{A} \circ \langle f_1, \cdots, f_n\rangle$ in $\mathbf{S}_T$. This yields a Σ-**algebra** structure over H^A and the interpretation of terms, namely $\mathcal{A}[\![-]\!] : S_\Sigma \to H^A$.

[3] We note that interpretations of terms are not restricted to algebra morphisms. The reason is to cover the interpretation of terms which is obtained by composition of morphisms of different algebras. This case is considered in example 5.

3. Let $\Sigma = (T_0, O_0)$ be a typed first-order signature and $\mathcal{A}$ be a many-sorted Σ-algebra. We give a categorical semantics of $\Pi_{\Sigma\text{-fol}}$ in **Set**, which coincides with the standard interpretation of the first-order classical logic in the model constructed over $\mathcal{A}$. The interpretation of types $I_{\mathcal{A}} : T_0 \cup \{\Omega\} \to \mathbf{Set}$ is given by $I_{\mathcal{A}}(\Omega) = \mathbf{2} = \{\top, \bot\}$ and $I_{\mathcal{A}}(\tau) = A^{\tau}$ for all $\tau \in T_0$. To give an interpretation of terms, we specify the following morphisms for each operator.

$$(u_{s^{\tau_1,\cdots,\tau_n\to\tau}})_\Gamma(f_1,\cdots,f_n) = s_{\mathcal{A}} \circ \langle f_1,\cdots,f_n\rangle \quad (s^{\tau_1,\cdots,\tau_n\to\tau} \in O_0)$$
$$(u_{\mathsf{exists}^{(\tau,\Omega)\to\Omega}})_\Gamma(f)(\rho) = \top \iff \exists x \in A^{\tau}\,.\, f(\rho, x) = \top$$
$$(u_{\mathsf{not}^{\Omega\to\Omega}})_\Gamma(f)(\rho) = \top \iff f(\rho) = \bot$$
$$(u_{\mathsf{or}^{\Omega,\Omega\to\Omega}})_\Gamma(f_1, f_2)(\rho) = \top \iff f_1(\rho) = \top \vee f_2(\rho) = \top$$

This gives the standard set-theoretic semantics of first-order classical logic over $\mathcal{A}$, namely $\mathcal{I}_{\mathcal{A}}[\![-]\!] : S_{\Pi_{\Sigma\text{-fol}}} \to H^{I_{\mathcal{A}}}$.

Predicates: Subscone. We introduce the notion of predicates over a categorical interpretation of types. When types are interpreted in set theory, the natural notion of predicate is simply a subset of each carrier set. In categorical settings, carrier sets are replaced by objects, and the notion of predicates is more subtle.

We write $\mathbf{Sub}(\mathbb{D})$ for the category of subobjects in a category $\mathbb{D}$, and $p_{\mathbb{D}} : \mathbf{Sub}(\mathbb{D}) \to \mathbb{D}$ for the forgetful functor. First we recall *injective scones* in [MS93]. The injective scone of a Cartesian category $\mathbb{C}$ is the category obtained by pulling back $p_{\mathbf{Set}}$ along the global section functor $\mathbb{C}(1, -)$ ([Jac99], example 1.5.2). In this approach, the notion of predicates over an object C in $\mathbb{C}$ is represented as subsets of global elements of C.

In this paper we use the *subscone* approach [Laf88,MR92,MS93,PPST00], which is a mild generalisation of injective scones. We replace **Set** with a category $\mathbb{D}$ with finite limits and the global section functor with finite-product preserving functor. We define the *category* $\mathbf{Pred}(G)$ *of G-predicates* by pulling back $p_{\mathbb{D}}$ along G.

$$\begin{array}{ccc} \mathbf{Pred}(G) & \longrightarrow & \mathbf{Sub}(\mathbb{D}) \\ {\scriptstyle \pi_G}\downarrow & & \downarrow{\scriptstyle p_{\mathbb{D}}} \\ \mathbb{C} & \xrightarrow[G]{} & \mathbb{D} \end{array}$$

Category $\mathbf{Pred}(G)$ has finite products which are strictly preserved by π_G[4]. We also define the category $\mathbf{Rel}_2(G)$ of binary G-relations to be $\mathbf{Pred}(\mathrm{prod} \circ G \times G)$, where $\mathrm{prod} : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ gives the binary products in $\mathbb{C}$.

We adopt the following notational convention. Let P and Q be objects in $\mathbf{Pred}(G)$ and $f : \pi_G(P) \to \pi_G(Q)$ be a morphism in $\mathbb{C}$. We write $f : P \to Q$ if there exists a morphism $g : P \to Q$ in $\mathbf{Pred}(G)$ such that $\pi_G(g) = f$.

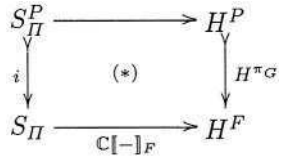
Let $F : T \to \mathbb{C}$ and $P : T \to \mathbf{Pred}(G)$ be interpretations of types. We say that P is a *G-predicate over F* (written $P \subseteq_G F$) if $\pi_G \circ P = F$. A *binary G-relation P* between $F_1, F_2 : T \to \mathbb{C}$ (written $P \subseteq_G F_1, F_2$) is just a $(\mathrm{prod} \circ G \times G)$-predicate over $\langle F_1, F_2\rangle$. For a predicate $P \subseteq_G F$, there exists a monomorphism $H^{\pi_G} : H^P \rightarrowtail H^F$.

[4] We give a proof in terms of fibred category theory. In fact $p_{\mathbb{D}}$ is a fibration with fibred finite limits, thus so is π_G (see [Jac99], section 1.8). Then it follows from lemma 8.5.2 of [Jac99].

3 Pre-logical Predicates

In this section, we fix a Cartesian category $\mathbb{C}$, a category $\mathbb{D}$ with finite limits, a finite product preserving functor $G : \mathbb{C} \to \mathbb{D}$ and a binding signature Π.

Let $\mathbb{C}[\![-]\!]_F$ be a categorical interpretation of Π and $P \subseteq_G F$ be a predicate. We consider taking pullback of H^{π_G} along $\mathbb{C}[\![-]\!]_F$ in $\mathbf{M}_T$.



The vertex S_Π^P can be calculated as $S_\Pi^P(\tau)(\Gamma) = \{M \mid \Gamma \vdash_\Pi M : \tau \wedge \mathbb{C}[\![M]\!]_F : P^*\Gamma \to P\tau\}$. This represents the collection of terms whose meanings by $\mathbb{C}[\![-]\!]_F$ respects the predicate P. Thus when this is isomorphic to S_Π, the meanings of all the well-formed terms by $\mathbb{C}[\![-]\!]_F$ respects the predicate P.

Definition 1. *Let $\mathbb{C}[\![-]\!]_F$ be a categorical interpretation of Π. We say that a predicate $P \subseteq_G F$* satisfies the basic lemma for Π along $\mathbb{C}[\![-]\!]_F$ *if in diagram $(*)$, S_Π^P is isomorphic to S_Π. This is equivalent to say that there exists a necessarily unique morphism $p : S_\Pi \to H^P$ (convention: we use the small letter of the predicate) such that $H^{\pi_G} \circ p = \mathbb{C}[\![-]\!]_F$.*

Example 3. (Continued from example 2)

1. Let $P \subseteq_{\mathbb{C}(1,-)} F_\lambda$ be a predicate satisfying the basic lemma for Π_λ along $\mathbb{C}[\![-]\!]_{F_\lambda}$. It is equivalent to the $\mathbf{Typ}^{\Rightarrow}(B)$-**indexed** family of subsets $P\tau \subseteq \mathbb{C}(1, F_\lambda\tau)$ such that for all $\rho \in P^*\Gamma$ and $\Gamma \vdash_{\Pi_\lambda} M : \tau$, we have $\mathbb{C}[\![M]\!]_{F_\lambda} \circ \rho \in P\tau$.
2. Let $P \subseteq_{\mathbf{Id_{Set}}} A$ be a predicate satisfying the basic lemma for Σ along $\mathcal{A}[\![-]\!]$. It is equivalent to the T_0-**indexed** family of subsets $P\tau \subseteq A\tau$ satisfying $\mathcal{A}[\![M]\!]\rho \in P\tau$ for all $\Gamma \vdash_\Sigma M : \tau$ and $\rho \in P^*\Gamma$.
3. Let Σ be a first-order signature, $\mathcal{A}, \mathcal{B}$ be many-sorted Σ-**algebras** and $P \subseteq_{\mathbf{Id_{Set}}} I_\mathcal{A}, I_\mathcal{B}$ be a binary relation satisfying the basic lemma for $\Pi_{\Sigma\text{-fol}}$ along $\mathcal{I}_\mathcal{A}[\![-]\!] \times \mathcal{I}_\mathcal{B}[\![-]\!]$ and $P\mathbf{\Omega} = \mathrm{id}_\mathbf{\Omega}$. It is equivalent to a $T_0 \cup \{\mathbf{\Omega}\}$-**indexed** family of subsets $P\tau \subseteq A\tau \times B\tau$ such that for all $\Gamma \vdash_{\Pi_{\Sigma\text{-fol}}} M : \tau$ and $(\rho, \rho') \in P^*\Gamma$, $(\mathcal{I}_\mathcal{A}[\![M]\!]\rho, \mathcal{I}_\mathcal{B}[\![M]\!]\rho') \in P\tau$ when $\tau \in T_0$ and $\mathcal{I}_\mathcal{A}[\![M]\!]\rho = \mathcal{I}_\mathcal{B}[\![M]\!]\rho$ when $\tau = \mathbf{\Omega}$. The latter implies that $\mathcal{A}$ and $\mathcal{B}$ are *elementary equivalent.*

Now we introduce the notion of pre-logical predicates.

Definition 2. *Let $\mathbb{C}[\![-]\!]_F$ be a categorical interpretation of Π. We call a predicate $P \subseteq_G F$* pre-logical for Π along $\mathbb{C}[\![-]\!]_F$ *if in diagram $(*)$ there exists a necessarily unique Π-**algebra** $(S_\Pi^P, \alpha : \Pi S_\Pi^P \to S_\Pi^P)$ such that the projection i induced by pullback is a Π-**algebra** morphism to the initial algebra (S_Π, ι_Π).*

An elementary description of P being pre-logical is that a) for all $\Gamma \vdash_{\Pi} x^{\tau} : \tau, \mathbb{C}[\![x^{\tau}]\!]_F : P^*\Gamma \to P\tau$ and b) for all operator $o \in O$ of arity $(\overrightarrow{\tau_1}, \sigma_1), \cdots, (\overrightarrow{\tau_m}, \sigma_m) \to \tau$ and well-formed terms $\Gamma, \overrightarrow{x_i : \tau_i} \vdash_{\Pi} M_i : \sigma_i \, (1 \leq i \leq n), \mathbb{C}[\![M_i]\!]_F : P^*(\Gamma, \overrightarrow{x_i : \tau_i}) \to P\sigma_i$ for all $1 \leq i \leq n$ implies $\mathbb{C}[\![o(\overrightarrow{x_1^{\tau_1}}.M_1, \cdots, \overrightarrow{x_n^{\tau_n}}.M_n)]\!]_F : P^*\Gamma \to P\tau$. Normally a) is satisfied as variables are interpreted by projections. For operators having no variable binding, if the interpretation of terms $\mathbb{C}[\![-]\!]_F$ satisfies a) and the semantic substitution lemma, i.e. $\mathbb{C}[\![M[\overrightarrow{M_i/x_i}]]\!]_F = \mathbb{C}[\![M]\!]_F \circ \langle \overrightarrow{\mathbb{C}[\![M_i]\!]_F} \rangle$, then the condition b) can be rewritten to $\mathbb{C}[\![s^{\tau_1, \cdots, \tau_n \to \tau}(x_1^{\tau_1}, \cdots, x_n^{\tau_n})]\!]_F \in P^*(\overrightarrow{x : \tau}) \to P\tau$.

Example 4. (Continued from example 2)

1. A predicate $P \subseteq_{\mathbb{C}(1,-)} F_{\lambda}$ is pre-logical for Π_{λ} along $\mathbb{C}[\![-]\!]_{F_{\lambda}}$ if for all $f \in P(\tau \Rightarrow \tau')$ and $g \in P\tau$, $@ \circ \langle f, g \rangle \in P\tau'$, and for all $\Gamma, x : \tau \vdash_{\Pi_{\lambda}} M : \tau'$, $\forall \rho \in P^*(\Gamma, x : \tau) \,.\, \mathbb{C}[\![M]\!]_{F_{\lambda}} \circ \rho \in P\tau'$ implies $\forall \rho \in P^*\Gamma \,.\, \mathbb{C}[\![\lambda x : \tau \,.\, M]\!]_{F_{\lambda}} \circ \rho \in P(\tau \Rightarrow \tau')$.
2. A predicate $P \subseteq_{\mathbf{Id_{Set}}} A$ is pre-logical for Σ along $\mathcal{A}[\![-]\!]$ if for all $c^{\tau_1, \cdots, \tau_n \to \tau} \in O_0$ and $x_i \in P^{\tau_i}$, $c_{\mathcal{A}}^{\tau_1, \cdots, \tau_n \to \tau}(x_1, \cdots, x_n) \in P^{\tau}$. An *algebraic predicate* [HS02] is just a pre-logical predicate for Σ_{CL} along $\mathcal{U}[\![-]\!]$ for a typed combinatory algebra $\mathcal{U}$ (i.e. a many-sorted Σ_{CL} algebra).
3. A predicate $P \subseteq_{\mathbf{Id_{Set}}} I_{\mathcal{A}}$ is pre-logical for $\Pi_{\Sigma\text{-fol}}$ along $\mathcal{I}_{\mathcal{A}}[\![-]\!]$ if for all first-order operator (including or, not) $s^{\tau_1, \cdots, \tau_n \to \tau}$ and $x_i \in P^{\tau_i}$, $s_{\mathcal{A}}^{\tau_1, \cdots, \tau_n \to \tau}(x_1, \cdots, x_n) \in P^{\tau}$ holds, and for all $\tau \in T_0$ and $\Gamma, x : \tau \vdash M : \mathbf{\Omega}$, $\forall \rho \in P^*(\Gamma, x : \tau) \,.\, \mathcal{I}_{\mathcal{A}}[\![M]\!]\rho \in P\mathbf{\Omega}$ implies $\forall \rho \in P^*\Gamma \,.\, \mathcal{I}_{\mathcal{A}}[\![\mathsf{exists}(x.M)]\!]\rho \in P\mathbf{\Omega}$.

Theorem 1 (The Basic Lemma of Pre-logical Predicates). *Let $\mathbb{C}[\![-]\!]_F$ be a categorical interpretation of Π. A predicate $P \subseteq_G F$ is pre-logical if and only if P satisfies the basic lemma.*

Proof. (if) If P satisfies the basic lemma, we have an isomorphism $f : S_{\Pi}^P \cong S_{\Pi}$. Then $f : (S_{\Pi}^P, f^{-1} \circ \iota_{\Pi} \circ (\Pi f)) \to (S_{\Pi}, \iota_{\Pi})$ is a Π-**algebra** morphism. Therefore P is pre-logical.

(only if) Suppose there exists a Π-**algebra** (S_{Π}^P, α). Let $! : (S_{\Pi}, \iota_{\Pi}) \to (S_{\Pi}^P, \alpha)$ be the unique morphism from the initial Π-**algebra.** From the universal property of initial Π-**algebra,** we have $i \circ ! = id$. Now we have $i \circ ! \circ i = i = i \circ id$, and since i is mono, $! \circ i = id$. Therefore (S_{Π}^P, α) and (S_{Π}, ι_{Π}) are isomorphic, thus S_{Π} and S_{Π}^P are so. □

This theorem is a categorical re-formulation of the inductive proof of the basic lemma for pre-logical relations in [HS02]. From now on we identify pre-logical predicates and predicates satisfying the basic lemma.

We give one sufficient condition for P being pre-logical. Below we identify a monomorphism in $\mathbf{M}_T$ and an object in $\mathbf{Sub}(\mathbf{M}_T)$. First, we can *lift* the endofunctor (of a typed binding signature) Π to the one over $\mathbf{Sub}(\mathbf{M}_T)$, namely $\tilde{\Pi}$. Here lifting means that $\tilde{\Pi}$ satisfies $p_{\mathbf{M}_T} \circ \tilde{\Pi} = \Pi \circ p_{\mathbf{M}_T}$ (see [Jac99], section 9.2). This is because all the constructs of Π have liftings over $\mathbf{Sub}(\mathbf{S}_T)$. Functor $p_{\mathbf{M}_T}$ is a subobject fibration, thus admits comprehension ([Jac99], example 4.6.3). It is easy to see that $\tilde{\Pi} \circ \top \cong \top \circ \Pi$, where $\top : \mathbf{M}_T \to \mathbf{Sub}(\mathbf{M}_T)$ is the right adjoint of $p_{\mathbf{M}_T}$ giving fibred terminal objects. Thus an initial Π-**algebra** is inductive ([Jac99], definition 9.2.6, proposition 9.2.7), i.e. $id_{S_{\Pi}}$ is an initial $\tilde{\Pi}$-**algebra.**

Proposition 1. *Let $P \subseteq_G F$ be a predicate and suppose that $H^{\pi_G} : H^P \rightarrowtail H^F$ has a $\tilde{\Pi}$-algebra structure. Then P satisfies the basic lemma for Π along the initial algebra semantics of Π in H^F.*

4 Composability of Pre-logical Relations

We move to the composability of binary pre-logical relations. Binary pre-logical relations are closed under relational composition, which is not enjoyed by logical relations [HS02]. We give here a categorical account of composability of pre-logical relations. In this section we fix a typed binding signature Π, a Cartesian category $\mathbb{C}$, a category $\mathbb{D}$ with finite limits, a finite-product preserving functor $G : \mathbb{C} \to \mathbb{D}$ and categorical interpretations $\mathbb{C}[\![-]\!]_{F_i} (1 \leq i \leq 3)$ of Π. We write fst, $\mathrm{snd} : \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ for projections.

First, we assume that a composition operator c over $\mathbf{Rel}_2(G)$ is available. This operator is partial, and defined over *composable pairs of relations,* i.e. a pair (R, S) of objects in $\mathbf{Rel}_2(G)$ such that $\mathrm{snd}(\pi_G(R)) = \mathrm{fst}(\pi_G(S))$. The composition operator yields an object $c(R, S)$ in $\mathbf{Rel}_2(G)$ such that $\mathrm{fst}(\pi_G(c(R, S))) = \mathrm{fst}(\pi_G(R))$ and $\mathrm{snd}(\pi_G(c(R, S))) = \mathrm{snd}(\pi_G(S))$, and a morphism $c(f, g) : c(R, S) \to c(R', S')$ for composable pairs of relations $(R, S), (R', S')$ and morphisms $f : R \to R', g : S \to S'$ in $\mathbf{Rel}_2(G)$. It is natural to assume that $\mathbf{Rel}_2(G)$ has identity relation, and the composition operator satisfies the laws of identity and associativity. To summarise, we assume that we have a *category object* in **Cat**:

$$\mathbf{Rel}_c(G) \xrightarrow{\ c\ } \mathbf{Rel}_2(G) \underset{\partial_2 = \mathrm{snd} \circ \pi_G}{\overset{\partial_1 = \mathrm{fst} \circ \pi_G}{\underset{\longleftarrow}{\overset{\longleftarrow}{id}}{\rightrightarrows}}} \mathbb{C} \qquad (**)$$

where $\mathbf{Rel}_c(G)$ is the category of composable pairs of relations obtained by pulling back ∂_2 along ∂_1. Using category objects in **Cat** to formulate the composition of relations is due to [KOPT97].

For $R \subseteq_G F_1, F_2$ and $S \subseteq_G F_2, F_3$, we define their composition $c(R, S)(\tau)$ to be $c(R\tau, S\tau)$. It is clear that $c(R, S) \subseteq_G F_1, F_3$.

Theorem 2 (Composability of Pre-logical Relations). *Let $R \subseteq_G F_1, F_2$ and $S \subseteq_G F_2, F_3$ be pre-logical binary relations for Π along $\mathbb{C}[\![-]\!]_{F_1} \times \mathbb{C}[\![-]\!]_{F_2}$ and $\mathbb{C}[\![-]\!]_{F_2} \times \mathbb{C}[\![-]\!]_{F_3}$ respectively. Then $c(R, S) \subseteq_G F_1, F_3$ is pre-logical for Π along $\mathbb{C}[\![-]\!]_{F_1} \times \mathbb{C}[\![-]\!]_{F_3}$.*

Proof. We find a morphism $h : S_\Pi \to H^{c(R,S)}$ such that $H^{\pi_G} \circ h = \mathbb{C}[\![-]\!]_{F_1} \times \mathbb{C}[\![-]\!]_{F_3}$ where $H^{\pi_G} : H^{c(R,S)} \to H^{\langle F_1, F_3 \rangle}$ is the morphism in $\mathbf{M}_T$. We give h by $h(\tau)(\Gamma)(M) = c(r(\tau)(\Gamma)(M), s(\tau)(\Gamma)(M))$ for all well-formed terms $\Gamma \vdash_\Pi M : \tau$, where $r : S_\Pi \to H^R$ and $s : S_\Pi \to H^S$ are morphisms which exist by definition of the basic lemma (see definition 1). □

When do we have a category object as $(**)$ above? Recall that composition of relations can be expressed by the $c(R, S)(x, z) = \exists y. R(x, y) \wedge S(y, z)$. The standard interpretation of this formula in set theory gives the composition of binary relations. Now we replace set theory with *regular fibration* [Jac99], which is a preordered fibration

$p : \mathbb{E} \to \mathbb{B}$ such that $\mathbb{B}$ is Cartesian and p has fibred finite products, fibred equality and simple coproducts satisfying Frobenius and Beck-Chevalley (for details, see [Jac99]). A regular fibration provides a categorical model of the $\exists\wedge\top$=-**fragment** of predicate logic. Interpreting the above formula in this model gives rise to a composition operation, which enjoys the identity and associativity laws.

Proposition 2. *Assume that* $p_{\mathbb{D}} : \mathbf{Sub}(\mathbb{D}) \to \mathbb{D}$ *is a regular fibration. Then we can construct a category object as* $(**)$ *above in* **Cat.**

5 Examples

Example 5. In this example, we examine the relationship between pre-logical predicates for combinatory algebras and pre-logical predicates for the simply typed lambda calculus in our framework. This is a revisit of proposition 3.3 in [HS02].

The standard abstraction mechanism $\lambda^* x.M$ in combinatory logic (see definition 7.1.5, [Bar84]) induces a $\Pi\lambda$-**algebra** structures over $S_{\Sigma_{CL}}$. From the universal property of initial $\Pi\lambda$-**algebra,** there is a unique $\Pi\lambda$-**algebra** morphism, namely $(-)_{CL} : S_{\Pi\lambda} \to S_{\Sigma_{CL}}$, which coincides with the standard lambda-to-CL translation (definition 7.3.1, [Bar84]). The composition $\mathcal{U}[\![(-)_{CL}]\!]$ gives an interpretation of the simply typed lambda calculus in a combinatory algebra $\mathcal{U}$. In general this is not a $\Pi\lambda$-**algebra** morphism. Conversely, giving the standard representation of *S, K* combinators in $S_{\Pi\lambda}$ equips it with a Σ_{CL} algebra structure. Then there exists a unique Σ_{CL}-**algebra** morphism from an initial Σ_{CL}-**algebra,** namely $(-)_\lambda : S_{\Sigma_{CL}} \to S_{\Pi\lambda}$.

Let $\mathcal{U}$ be a combinatory algebra and $P \subseteq_{\mathbf{Id_{Set}}} U$ be a pre-logical predicate for Σ_{CL} along $\mathcal{U}[\![-]\!]$. Then we have $H^{\pi_G} \circ p \circ (-)_{CL} = \mathcal{U}[\![(-)_{CL}]\!]$, thus P is pre-logical for $\Pi\lambda$ along $\mathcal{U}[\![(-)_{CL}]\!]$. This explains that an algebraic predicate relating combinators yields a pre-logical predicate ("if" part of proposition 3.3, [HS02]). Conversely, let $P \subseteq_{\mathbf{Id_{Set}}} U$ be a pre-logical predicate for $\Pi\lambda$ along $\mathcal{U}[\![(-)_{CL}]\!]$. It is a pre-logical predicate for Σ_{CL} along $\mathcal{U}[\![((-)_\lambda)_{CL}]\!]$—but not for $\mathcal{U}[\![-]\!]$ in general!

Theorem 3. *There exists a combinatory algebra* $\mathcal{U}_0$ *and a pre-logical predicate* $P \subseteq_{\mathbf{Id_{Set}}} U_0$ *for* $\Pi\lambda$ *along* $\mathcal{U}_0[\![(-)_{CL}]\!]$ *which is not not pre-logical for* Π_{CL} *along* $\mathcal{U}_0[\![-]\!]$.

The proof uses the fact that the image of the standard lambda-to-CL translation does not cover the entire set of combinatory logic terms, particularly *S* and *K*. To exploit this fact, we take $\mathcal{U}_0$ as the closed term algebra, and see that the definability predicate by $\mathcal{U}_0[\![(-)_{CL}]\!]$, which is pre-logical for $\Pi\lambda$, is not pre-logical for Π_{CL} along $\mathcal{U}_0[\![-]\!]$. This means that "only if" part of proposition 3.3, [HS02] is not precise enough. The subtle point is that "to which semantics" it satisfies the basic lemma, and it was missed in [HS02].

When is P a pre-logical predicate for Σ_{CL} along $\mathcal{U}[\![-]\!]$? One answer is to fix the lambda-to-CL translation $(-)_{CL}$ to make it surjective. To achieve this, we introduce another abstraction mechanism $\lambda' x.M$ defined to be $\lambda' x.x = SKK$, $\lambda' x.M = KM$ provided $x \notin \mathrm{FV}(M)$, $\lambda' x.\lambda^* y.x = K$, $\lambda' x.\lambda^* y.\lambda^* z.xz(yz) = S$, $\lambda' x.MN = S(\lambda' x.M)(\lambda' x.N)$. The lambda-to-CL translation constructed from this abstraction mechanism, say $(-)_{CL'}$, covers all the combinators, and moreover satisfies $((M)_\lambda)_{CL'}$

$= M$. Thus a pre-logical predicate for Π_λ along $\mathcal{U}[\![(-)_{CL'}]\!]$ is a pre-logical predicate for Σ_{CL} along $\mathcal{U}[\![-]\!]$. Another answer is to require $\mathcal{U}$ to be a *lambda algebra,* which always satisfies $\mathcal{U}[\![((-)_\lambda)_{CL}]\!] = \mathcal{U}[\![-]\!]$ (see lemma 5.2.3-2, [Bar84]).

Example 6. We examine the connection between lax logical predicates [PPST00] and pre-logical predicates as defined in here. For this, we fix a set of base types B and define the set of types including finite products by the BNF $\mathbf{Typ}^{\Rightarrow\times}(B) \ni \tau ::= b \mid 1 \mid \tau \times \tau \mid \tau \Rightarrow \tau$ where $b \in B$. The signature for the simply typed lambda calculus with finite products is defined by

$$\Pi_{\lambda\times} = (\mathbf{Typ}^{\Rightarrow\times}(B), \{\mathsf{lam}^{(\tau,\tau')\to\tau\Rightarrow\tau'}, \mathsf{app}^{\tau\Rightarrow\tau',\tau\to\tau'}, *^1, \mathsf{pair}^{\tau,\tau'\to\tau\times\tau'}, \mathsf{fst}^{\tau\times\tau'\to\tau}, \mathsf{snd}^{\tau\times\tau'\to\tau'}\}).$$

Let $\mathbf{L}$ be the free CCC generated from the set of base types B. An object of $\mathbf{L}$ is a type $\tau \in \mathbf{Typ}^{\Rightarrow\times}(B)$, and a morphism from τ to τ' in $\mathbf{L}$ is a $\beta\eta$-equivalence class of a well-formed terms $x : \tau \vdash_{\Pi_{\lambda\times}} M : \tau'$. We write $I : \mathbf{Typ}^{\Rightarrow\times}(B) \to \mathbf{L}$ for the inclusion functor. As we have seen in example 2, since $\mathbf{L}$ is a CCC, it provides a $\Pi_{\lambda\times}$-algebra structure, thus there exists a unique $\Pi_{\lambda\times}$-morphism $\mathbf{L}[\![-]\!]_I : S_{\Pi_{\lambda\times}} \to H^I$. We note that the mapping $\mathbf{L}[\![-]\!]_I$ is an epimorphism.

Let $\mathbb{C}$ be a CCC, $\mathbb{D}$ be a CCC with finite limits, $G : \mathbb{C} \to \mathbb{D}$ be a functor preserving finite products and $[\![-]\!] : \mathbf{L} \to \mathbb{C}$ be a strict Cartesian closed functor. *A lax logical predicate* q [PPST00] over $[\![-]\!]$ is a finite-product preserving functor $q : \mathbf{L} \to \mathbf{Pred}(G)$ such that $\pi_G \circ q = [\![-]\!]$.

Theorem 4. *A lax logical predicate* $p : \mathbf{L} \to \mathbf{Pred}(G)$ *determines a pre-logical predicate* $p \circ I \subseteq_G [\![-]\!] \circ I$ *for* $\Pi_{\lambda\times}$ *along* $H^{[\![-]\!]} \circ \mathbf{L}[\![-]\!]_I$. *Conversely, if* $P \subseteq_G [\![-]\!] \circ I$ *is a pre-logical predicate for* $\Pi_{\lambda\times}$ *along* $H^{[\![-]\!]} \circ \mathbf{L}[\![-]\!]_I$, *then there exists a lax logical predicate* q *such that for all* $\Gamma \vdash_{\Pi_{\lambda\times}} M : \tau$, $H^q \circ \mathbf{L}[\![M]\!]_I = p(\tau)(\Gamma)(M)$.

Example 7. In this example we see a characterisation of elementary submodels in terms of a binary pre-logical relation. Let $\Sigma = (T_0, O_0)$ be a typed first-order signature, $\mathcal{B}$ be a many-sorted Σ-algebra and $\mathcal{A}$ be a subalgebra of $\mathcal{B}$. For all $\Gamma \vdash_{\Pi_{\Sigma\text{-fol}}} M : \tau$ with $\tau \in T_0$, we have $\mathcal{A}[\![M]\!] = \mathcal{B}[\![M]\!]$ because $\mathcal{A}$ is a submodel of $\mathcal{B}$. However, this may not hold when $\tau = \Omega$ because of existential quantifier. Thus we say $\mathcal{A}$ is an *elementary submodel* of $\mathcal{B}$ (written $\mathcal{A} \preceq \mathcal{B}$) if the above holds for $\tau = \Omega$ as well.

Theorem 5. $\mathcal{A} \preceq \mathcal{B}$ *if and only if the inclusion relation* $R \subseteq_{\mathbf{Id}_{\mathbf{Set}}} I_\mathcal{A}, I_\mathcal{B}$, *i.e.* $R^\Omega = id_\Omega$ *and* $R^\tau = \{(x, x) \mid x \in A^\tau\}$, *is pre-logical for* $\Pi_{\Sigma\text{-fol}}$ *along* $\mathcal{I}_\mathcal{A}[\![-]\!] \times \mathcal{I}_\mathcal{B}[\![-]\!]$.

6 Conclusion

We have given a generalisation of pre-logical predicates to arbitrary typed formal systems, and shown that they are equivalent to predicates satisfying the basic lemma, and that binary pre-logical relations are closed under composition. We represent three underlying components of pre-logical predicates — syntax, semantics and predicates —

in the category of presentation models. Then we formulate pre-logical predicates and predicates satisfying the basic lemma, and show their equivalence.

It is interesting to extend our framework for defining formal systems. One direction is to allow type variables so that we can cover type systems such as System F or FPC [FP94]. The other direction is to modify the notion of contexts from the Cartesian one to linear one to cover linear logic. In both cases we also have to switch the notion of models from Cartesian categories to more elaborate categorical structures such as polymorphic fibrations, symmetric monoidal categories, etc.

Acknowledgments. I thank Donald Sannella, Daniel Turi and John Power for useful discussions. This work was supported by an LFCS studentship.

References

[Bar84] H. Barendregt. *The Lambda Calculus-Its Sytax and Semantics.* North Holland, 1984.

[Fio02] M. Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. In *Proc. PPDP 2002,* pages 26–37. ACM, 2002.

[FP94] M. Fiore and G. Plotkin. An axiomatization of computationally adequate domain theoretic models of FPC. In *Proc. LICS 1994,* pages 92–102. IEEE, 1994.

[FPT99] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proc. LICS 1999,* pages 193–202. IEEE Computer Society Press, 1999.

[Her93] C. Hermida. *Fibrations, Logical Predicates and Indeterminantes.* PhD thesis, The University of Edinburgh, 1993.

[HLST00] F. Honsell, J. Longley, D. Sannella, and A. Tarlecki. Constructive data refinement in typed lambda calculus. In *Proc. FoSSACS 2000,* volume 1784 of *LNCS,* pages 161–176. Springer, 2000.

[Hof99] M. Hoffman. Semantical analysis of higher-order abstract syntax. In *Proc. LICS 1999,* pages 204–213. IEEE Computer Society, 1999.

[HS02] F. Honsell and D. Sannella. Prelogical relations. *INFCTRL: Information and Computation (formerly Information and Control),* 178(1):23–43, 2002.

[Jac99] B. Jacobs. *Categorical Logic and Type Theory.* Elsevier, 1999.

[Kat03] S. Katsumata. Behavioural equivalence and indistinguishability in higher-order typed languages. In *WADT 2002, Revised Selected Papers,* volume 2755 of *LNCS,* pages 284–298. Springer, 2003.

[KOPT97] Y. Kinoshita, P. W. O'Hearn, A. J. Power, and M. Takeyama. An axiomatic approach to binary logical relations with applications to data refinement. In *Proc. TACS 1997,* volume 1281 of *LNCS,* pages 191–212. Springer, 1997.

[KP99] Y. Kinoshita and J. Power. Data-refinement for call-by-value programming languages. In *Proc. CSL 1999,* volume 1683 of *LNCS,* pages 562–576. Springr, 1999.

[Laf88] Y. Lafont. *Logiques, Categories et Machines.* PhD thesis, Université de Paris VII, 1988.

[Lei01] H. Leiß. Second-order pre-logical relations and representation independence. In *Proc. TLCA 2001,* volume 2044 of *LNCS,* pages 298–314. Springer, 2001.

[Mit96] J.Mitchell. *Foundations for Programming Languages.* MIT Press, 1996.

[MR92] Q. Ma and J. C. Reynolds. Types, abstractions, and parametric polymorphism, part 2. In *In Proc. MFPS 1991,* volume 598 of *LNCS,* pages 1–40. Springer, 1992.

[MS93] J. Mitchell and A. Scedrov. Notes on sconing and relators. In *Proc. CSL 1992,* volume 702 of *LNCS,* pages 352–378. Springer, 1993.

[MS03] M. Miculan and I. Scagnetto. A framework for typed HOAS and semantics. In *Proc. PPDP 2003,* pages 184–194. ACM, 2003.

[PPST00] G. Plotkin, J. Power, D. Sannella, and R. Tennent. Lax logical relations. In *Proc. ICALP 2000,* volume 1853 of *LNCS,* pages 85–102. Springer, 2000.

A Proofs

Proof of Proposition 1. The initiality of id_{S_Π} gives a morphism $f : S_\Pi \to H^P$ which is above the Π-**algebra** morphism $! : S_\Pi \to H^F$, i.e. $H^{\pi_G} \circ f = !$. Thus P satisfies the basic lemma by definition. □

Proof of Proposition 2. To prove this proposition, we use the internal logic of fibrations [Jac99]. From the assumption, the logic provides the $\exists \wedge \top$=-**fragment** of predicate logic. For details, see [Jac99]. The following reasoning is done in the internal logic of $p_{\mathbb{D}}$.

An object in $\mathbf{Rel}_2(G)$ is a triple (C, C', P) where C, C' are objects in $\mathbb{C}$ and P is a predicate $x : GC, y : GC' \vdash P(x, y)$ of the internal logic of the fibration. A morphism from (C, C', P) to (D, D', Q) is a pair of morphisms $f : C \to D, g : C' \to D'$ in $\mathbb{C}$ such that $x : GC, y : GC' \mid P(x, y) \vdash Q(Gf(x), Gg(y))$ holds.

An object in $\mathbf{Rel}_c(G)$ is a tuple (C, C', C'', P, P') such that (C, C', P) and (C', C'', P') are objects in $\mathbf{Rel}_2(G)$. A morphism from (C, C', C'', P, P') to (D, D', D'', Q, Q') in $\mathbf{Rel}_c(G)$ is a triple $(f : C \to D, g : C' \to D', h : C'' \to D'')$ such that $(f, g) : (C, C', P) \to (D, D', Q)$ and $(g, h) : (C', C'', P') \to (D', D'', Q')$ are morphisms in $\mathbf{Rel}_2(G)$.

For an object C in $\mathbb{C}$, we assign an object $id(C)$ in $\mathbf{Rel}_2(G)$ by $x : GC, y : GC \vdash x = y$. For all $f : C \to D$ in $\mathbb{C}$, we can derive a judgment $x : GC, y : GC \vdash x = y \vdash Gf(x) = Gf(y)$ in the internal logic of the fibration. We can extend this assignment to a functor $\mathbb{C} \to \mathbf{Rel}_2(G)$.

For an object (C, C', C'', P, P') in $\mathbf{Rel}_c(G)$, we define an object $c(C, C', C'', P, P')$ in $\mathbf{Rel}_2(G)$ by $C, C'', x : GC, z : GC'' \vdash \exists y : GC' \,.\, P(x, y) \wedge P'(y, z)$ (we omit re-indexing functors along projections for readability).

Let $(f, g, h) : (C, C', C'', P, P') \to (D, D', D'', Q, Q')$ be a morphism in $\mathbf{Rel}_c(G)$. In the internal logic of the fibration, we have the following derivation (annotation of objects are omitted for readability):

$$\dfrac{\dfrac{\dfrac{\dfrac{x, y \mid P(x, y) \vdash Q(Gf(x), Gg(y))}{x, y, z \mid P(x, y) \wedge P'(y, z) \vdash Q(Gf(x), Gg(y))} \quad \dfrac{y, z \mid P'(y, z) \vdash Q'(Gg(y), Gh(z))}{x, y, z \mid P(x, y) \wedge P'(y, z) \vdash Q'(Gg(y), Gh(z))}}{x, y, z \mid P(x, y) \wedge P'(y, z) \vdash Q(Gf(x), Gg(y)) \wedge Q'(Gg(y), Gh(z))}}{x, y, z \mid P(x, y) \wedge P'(y, z) \vdash \exists y \,.\, Q(Gf(x), y) \wedge Q'(y, Gh(z))}}{x, z \mid \exists y \,.\, P(x, y) \wedge P'(y, z) \vdash \exists y \,.\, Q(Gf(x), y) \wedge Q'(y, Gg(z))}$$

Thus c extends to a functor $c : \mathbf{Rel}_c(G) \to \mathbf{Rel}_2(G)$.

To see that *id* and *c* satisfy the laws of category object, such as $c(id(C), (C, C', P)) = (C, C', P)$, we show that the predicates on both sides are provably equal. The calculation is much like that in [Jac99], example 4.3.8. Since $p_{\mathbb{D}}$ is a fibred partial order, provable equality implies equality of objects. Thus the above equation strictly holds. □

Proof of Theorem 3. We use the fact that the image of the standard lambda-to-CL translation does not cover the entire set of combinatory logic terms.

First we write $[M]_w$ for the equivalence class of a combinatory logic term M by weak equivalence $=_w$ (see [Bar84], definition 7.2.1). We define the *closed term combinatory algebra* $\mathcal{U}_0$ by the tuple $(U_0, \bullet_w, [S]_w, [K]_w)$ where $U_0^\tau = \{[M]_w \mid M \in S_{\Sigma_{CL}}(\emptyset, \tau)\}$ and $\bullet_w^{\tau,\tau'}$ is the application operators defined by $[M]_w \bullet_w^{\tau,\tau'} [N]_w = [MN]_w$ for $[M]_w \in U_0^{\tau \Rightarrow \tau'}$ and $[N]_w \in U_0^\tau$. It is easy to see that the above choice of combinators satisfies the axioms of the combinatory algebra. As we have seen in example 2, we obtain an interpretation of combinatory logic terms in $\mathcal{U}_0$, namely $\mathcal{U}_0[\![-]\!]$. Explicitly, $\mathcal{U}_0[\![M]\!]\{[M_1]_w/x_1, \cdots, [M_n]_w/x_n\} = [M[M_1/x_1, \cdots, M_n/x_n]]_w$. We interpret simply typed lambda terms by $\mathcal{U}_0[\![(-)_{CL}]\!]$.

Now we define the definability predicate $D^\tau \subseteq U_0^\tau$ by $D^\tau = \{\mathcal{U}_0[\![M_{CL}]\!] \in U_0^\tau \mid M \in S_{\Pi_\lambda}(\emptyset, \tau)\}$. This is a pre-logical predicate for Π_λ along $\mathcal{U}_0[\![(-)_{CL}]\!]$. However, $D^{\tau \Rightarrow \tau' \Rightarrow \tau}$ does not include $[K^{\tau \Rightarrow \tau' \Rightarrow \tau}]_w$ for all τ and τ', thus is not a pre-logical predicate for Σ_{CL} along $\mathcal{U}_0[\![-]\!]$.

It is easy to see that there exists no closed term M such that $M_{CL} = K$ by induction on M. Next we prove the following lemma:

Lemma 1. *For all closed lambda term M and all combinatory term N, $M_{CL} \twoheadrightarrow_w N$ implies there exists a closed lambda term N' such that $N = N'_{CL}$.*

When M begins with a lambda abstraction, M_{CL} is always a normal form. Thus the claim clearly holds by taking $N' = M$. We do not consider the case when M is a variable, since we assume M is closed. So we think of the case when $M = M_0 M_1$ with two closed lambda terms M_0 and M_1. There are several possible causes of $M_{CL} \twoheadrightarrow_w N$.

- $N = M_{CL}$. We just take $N' = M$.
- There exists a combinatory term L such that $(M_0)_{CL} \twoheadrightarrow_w L$ and $L \twoheadrightarrow_w N$. From IH, there exists a combinatory term L' such that $L = L'_{CL}$. Again from IH, there exists a combinatory term N' such that $N = N'_{CL}$.
- $(M_0)_{CL} \twoheadrightarrow_w N_0$ and $N = N_0(M_1)_{CL}$. From IH, there exists a closed lambda term N'_0 such that $(N'_0)_{CL} = N_0$. Thus $N = (N'_0)_{CL}(M_1)_{CL} = (N'_0 M_1)_{CL}$.
- $(M_1)_{CL} \twoheadrightarrow_w N_1$ and $N = (M_0)_{CL} N_1$. The proof is similar to the above case.
- $(M_0)_{CL} = KN_0$ with a combinatory term N_0 and $N = N_0$. From the definition of lambda-to-CL translation, M_0 should be equal to $\lambda x\,.\, N'_0$ where N'_0 is a closed lambda term. Thus $N_0 = (N'_0)_{CL}$.
- $(M_0)_{CL} = SN_0N_1$ with some combinatory terms N_0, N_1 and $N = N_0(M_1)_{CL}(N_1(M_1)_{CL})$. From the definition of lambda-to-CL translation, M_0 should be equal to $\lambda x\,.\,(N'_0 N'_1)$. Then $\lambda x\,.\,(N'_0 N'_1) = S(N'_0)_{CL}(N'_1)_{CL}$, which implies $N_0 = (N'_0)_{CL}$ and $N_1 = (M'_1)_{CL}$. Thus we take $N' = N'_0 M_1 (N'_1 M_1)$.

Thus there exists no term reducing to K in the image of $(-)_{CL}$, otherwise K should be in the image of $(-)_{CL}$ (we assume the strong noramlisation of $\twoheadrightarrow_w$). Thus $[K]_w \notin D^{\tau \Rightarrow \tau' \Rightarrow \tau}$. □

Proof of Theorem 4. We only show the converse. The assumption says that there exists a morphism $p : S_{\Pi_{\lambda\times}} \to H^P$ such that $H^{\pi_G} \circ p = H^{[-]} \circ \mathbf{L}[\![-]\!]_I$. Recall that $\mathbf{L}[\![-]\!]_I$

is an epimorphism and H^{π_G} is a monomorphism. In category $\mathbf{M}_T$, any epimorphism is orthogonal to monomorphism, thus there exists a unique morphism h such that $H^{\pi_G} \circ h = H^{[\![-]\!]}$ and $h \circ \mathbf{L}[\![-]\!]_I = p$. Now we define the functor $q : \mathbf{L} \to \mathbf{Pred}(G)$ in question by $q\tau = P\tau$ and $qf = h(\tau')(\tau)(f)$ for a morphism $f : \tau \to \tau'$ in $\mathbf{L}$. We see q is indeed a functor. First q preserves identity, since $\pi_G(q(id_\tau)) = [\![id_\tau]\!] = id_{[\![\tau]\!]} = \pi_G(id_{P\tau})$ and π_G is faithful, we have $q(id_\tau) = id_{P\tau}$. Next we show $q(f \circ g) = qf \circ qg$ for all $f : \tau' \to \tau''$ and $g : \tau \to \tau'$. We have $\pi_G(q(f \circ g)) = [\![f \circ g]\!] = [\![f]\!] \circ [\![g]\!] = \pi_G(qf \circ qg)$, and since π_G is faithful, we have $q(f \circ g) = qf \circ qg$. It is routine to check $H^q \circ \mathbf{L}[\![M]\!]_I = p(\tau)(\Gamma)(M)$.

Next we show $P(\tau \times \tau') = P\tau \times P\tau'$. We consider well-formed terms $x : \tau \times \tau' \vdash_{\Pi_{\lambda\times}} \mathsf{fst}(x) : \tau$ and $x : \tau \times \tau' \vdash_{\Pi_{\lambda\times}} \mathsf{snd}(x) : \tau$. We define $j = \langle p(\tau)(x : \tau \times \tau')(\mathsf{fst}(x)), p(\tau')(x : \tau \times \tau')(\mathsf{snd}(x))\rangle$. Since $H^{\pi_G} \circ p = [\![-]\!] \circ \mathbf{L}[\![-]\!]_I$, we can show that $\pi_G j = \mathrm{id}_{[\![\tau\times\tau']\!]}$, which implies $j = \mathrm{id}_{P(\tau\times\tau')}$ since π_G is faithful. This means that the comparison map $P(\tau \times \tau') \to P\tau \times P\tau'$ is identity, thus $P(\tau \times \tau') = P\tau \times P\tau'$ holds. □

Proof of Theorem 5. Assume $\mathcal{A} \preceq \mathcal{B}$. We only have to show that for all $\Gamma \vdash_{\Pi_{\Sigma\text{-fol}}} M : \tau$ where $\tau \in T_0$ and $\rho \in A^*\Gamma$, $\mathcal{I}_\mathcal{A}[\![M]\!]\rho = \mathcal{I}_\mathcal{B}[\![M]\!]\rho$. This is clear, since M consists of operators in Σ, and $\mathcal{A}$ is a subalgebra of $\mathcal{B}$. Conversely, assume that the basic lemma holds. Then for all $\Gamma \vdash_{\Pi_{\Sigma\text{-fol}}} M : \Omega$ and $\rho \in A^*\Gamma$, we have $(\mathcal{I}_\mathcal{A}[\![M]\!]\rho, \mathcal{I}_\mathcal{B}[\![M]\!]\rho) \in R\Omega = \mathrm{id}_\Omega$. Thus $\mathcal{A} \preceq \mathcal{B}$ holds. □

A Faster Algorithm for Minimum Cycle Basis of Graphs

Telikepalli Kavitha[1*], Kurt Mehlhorn[1*], Dimitrios Michail[1*], and Katarzyna Paluch[2**]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany.
{kavitha,mehlhorn,michail}@mpi-sb.mpg.de
[2] Institute of Computer Science, University of Wroclaw, Poland.
abraka@ii.uni.wroc.pl

Abstract. In this paper we consider the problem of computing a minimum cycle basis in a graph G with m edges and n vertices. The edges of G have non-negative weights on them. The previous best result for this problem was an $O(m^\omega n)$ algorithm, where ω is the best exponent of matrix multiplication. It is presently known that $\omega < 2.376$. We obtain an $O(m^2 n + mn^2 \log n)$ algorithm for this problem. Our algorithm also uses fast matrix multiplication. When the edge weights are integers, we have an $O(m^2 n)$ algorithm. For unweighted graphs which are reasonably dense, our algorithm runs in $O(m^\omega)$ tune. For any $\epsilon > 0$, we also design a $1+\epsilon$ approximation algorithm to compute a cycle basis which is at most $1+\epsilon$ times the weight of a minimum cycle basis. The running time of this algorithm is $O(\frac{m^\omega}{\epsilon} \log(W/\epsilon))$ for reasonably dense graphs, where W is the largest edge weight.

1 Introduction

1.1 The Problem

Let $G = (V, E)$ be a graph. A *cycle* of G is any subgraph in which each vertex has even degree. Associated with each cycle is an *incidence vector* x, indexed on E, where $x_e = 1$ if e is an edge of C, $x_e = 0$ otherwise. The vector space over $GF(2)$ generated by the incidence vectors of cycles is called the *cycle space* of G. It is well-known that when G is connected, this vector space has dimension $N = m - n + 1$, where m is the number of edges of G and n is the number of vertices. A maximal set of linearly independent cycles is called a *cycle basis*.

The edges of G have non-negative weights. The weight of a cycle is the sum of the weights of its edges. The weight of a cycle basis is the sum of the weights of its cycles. We consider the problem of computing a cycle basis of minimum weight in a graph. (We use the abbreviation MCB to refer to a minimum cycle basis.)

* Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).
** Work done while the author was at MPII supported by Marie Curie Doctoral Fellowship.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 846–857, 2004.

1.2 Background

This problem has been extensively studied, both in its general setting and in special classes of graphs. Its importance lies in understanding the cyclic structure of a graph and its use as a preprocessing step in several algorithms. Such algorithms include algorithms for diverse applications like electrical circuit theory [2], structural engineering [1], and periodic event scheduling [5].

The oldest known references to the minimum cycle basis are Stepanec [13] and Zykov [17]. Though polynomial time algorithms for this problem were claimed, these algorithms were not correct [9,10]. The first polynomial time algorithm for the minimum cycle basis problem was given by Horton [8], and had running time $O(m^3n)$.

Horton's approach was to create a set M of mn cycles which he proved was a superset of an MCB and then extract the MCB as the shortest $m - n + 1$ linearly independent cycles from M using Gaussian elimination. Golynski and Horton [7] observed that the shortest $m-n+1$ linearly independent cycles could be obtained from M in $O(m^\omega n)$ time using fast matrix multiplication algorithms, where ω is the best exponent for matrix multiplication. It is presently known [4] that $\omega < 2.376$. The $O(m^\omega n)$ algorithm was the best known algorithm for the MCB problem.

De Pina [5] gave an $O(m^3 + mn^2 \log n)$ to compute an MCB in a graph. The approach in [5] is different from that of Horton; de Pina's algorithm is similar to the algorithm of Padberg and Rao [11] to solve the minimum weighted T-odd cut problem. Our new algorithm to compute an MCB is also based on the same approach.

1.3 New Results

In this paper we obtain the following new results.

For graphs with arbitrary non-negative weights on edges, we give an $O(m^2n+ mn^2 \log n)$ algorithm to compute an MCB, improving upon the current $O(m^\omega n)$ upper bound. In particular, whenever $m \geq n \log n$, we have an $O(m^2n)$ algorithm. We use an all pairs shortest paths (APSP) algorithm as a subroutine in our algorithm. We obtain better running times for integer edge weights and unweighted graphs by using faster all pairs shortest path algorithms for these cases [12,6,14,15]

We also look at approximation algorithms for computing a minimum cycle basis in a graph. Given any $\alpha > 1$, we have an α-approximation algorithm by relaxing the shortest paths subroutine to an α stretch paths[1] subroutine. We also show that a witness of a minimum cycle basis can be constructed in $O(m^\omega)$ time.

[1] An α stretch (s,t) path is a path which is at most α times the length of a shortest (s,t) path.

2 A Simple MCB Algorithm

De Pina [5] gave a combinatorial algorithm to compute a minimum cycle basis in a graph with non-negative weights on its edges. We feel that the intuition behind the algorithm is not clear from the combinatorial version of the algorithm. So, we interpret this algorithm algebraically. From the algebraic version of the algorithm, the scope for improvement is also clear.

2.1 An Algebraic Interpretation

Let $G = (V, E)$ be an undirected graph with m edges and n vertices and with non-negative weights on its edges. We assume that G is connected since a minimum cycle basis of a graph is the union of the minimum cycle bases of its connected components. Let T be any spanning tree of G. Let $e_1, \ldots, e_N$ be the edges of $G \setminus T$ in some arbitrary but fixed order.

A cycle in G can be viewed in terms of its incidence vector and so each cycle is a vector (with 0's and 1's in its coordinates) in the space spanned by all the edges. Here we will only look these vectors restricted to the coordinates indexed by $\{e_1, ..., e_N\}$.

In SIMPLE-MCB (see Fig. 1) we compute the cycles of a minimum cycle basis and their *witnesses*. A witness S of a cycle C is a subset of $\{e_1, ..., e_N\}$ which will prove that C belongs to our minimum cycle basis. We will view these witnesses or subsets in terms of their incidence vectors over $\{e_1, ..., e_N\}$.

Hence, both cycles and witnesses are vectors in the space $\{0,1\}^N$. $\langle C, S\rangle$ stands for the standard inner product of the vectors C and S. We say that a vector S is orthogonal to C if $\langle C, S\rangle = 0$. Since we are in the field $GF(2)$, observe that $\langle C, S\rangle = 1$ if and only if C contains an odd number of edges of S. We present in Fig. 1 a succinct description of the algorithm SIMPLE-MCB.

For $i = 1$ to N do the following:

1. Let S_i be any arbitrary non-zero vector in the subspace orthogonal to $\{C_1, C_2, ..., C_{i-1}\}$. That is, S_i is a non-trivial solution to the set of linear equations:
$$\langle C_k, x\rangle = 0 \text{ for } k = 1 \text{ to } i - 1.$$
(Initially, S_1 is any arbitrary non-zero vector in the space $\{0,1\}^N$.)
2. Compute a shortest cycle C_i such that $\langle C_i, S_i\rangle = 1$.

Fig. 1. SIMPLE-MCB: An algebraic framework for computing an MCB

Since each S_i is non-zero, it has to contain at least one edge e from $G\setminus T$. The cycle formed by edges of T and e has intersection of size exactly 1 with S_i. So, there is always at least one cycle with an odd number of edges of S_i.

Note that C_i is independent of $C_1, .., C_{i-1}$ because any vector v in the span of $\{C_1, ..., C_{i-1}\}$ satisfies $\langle v, S_i \rangle = 0$ (since $\langle C_j, S_i \rangle = 0$ for each $1 \leq j \leq i-1$), whereas $\langle C_i, S_i \rangle = 1$. Hence, it follows immediately that $\{C_1, ..., C_N\}$ is a basis.

We still have to describe how to compute a shortest cycle C_i such that $\langle C_i, S_i \rangle = 1$ and how to compute a non-zero vector S_i in the subspace orthogonal to $\{C_1, ..., C_{i-1}\}$. We will do that in Sections 2.2 and 2.3 respectively. We will first prove that $\{C_1, ..., C_N\}$ computed in SIMPLE-MCB forms an MCB.

Theorem 1. *The set $\{C_1, C_2, ..., C_N\}$ determined in SIMPLE-MCB is a minimum cycle basis.*

Proof. (from [5]) Suppose not. Then there exists an $0 \leq i < N$ such that there is a minimum cycle basis B that contains $\{C_1, ..., C_i\}$ but there is no minimum cycle basis that contains $\{C_1, ..., C_i, C_{i+1}\}$. Since the cycles in B form a spanning set, there exist cycles $D_1, ..., D_k$ in B such that

$$C_{i+1} = D_1 + D_2 + \cdots + D_k$$

Since $\langle C_{i+1}, S_{i+1} \rangle = 1$, there exists some D_j in the above sum such that $\langle D_j, S_{i+1} \rangle$ is 1. But C_{i+1} is a shortest cycle such that $\langle C_{i+1}, S_{i+1} \rangle = 1$. So the weight of $C_{i+1} \leq$ the weight of D_j.

Let $B' = B \cup \{C_{i+1}\} \setminus \{D_j\}$. It is easy to see that B' is also a basis. And the weight of B' is at most the weight of B which is a minimum cycle basis. So B' is also a minimum cycle basis. It is easy to show that $\{C_1, C_2, ..., C_{i+1}\} \subseteq B'$ because by assumption $\{C_1, ..., C_i\} \subseteq B$ and the cycle D_j that was omitted from B cannot be equal to any of $C_1, ..., C_i$ because $\langle D_j, S_{i+1} \rangle = 1$ whereas $\langle C_j, S_{i+1} \rangle = 0 \ \forall j \leq i$.

The existence of the basis B' contradicts that there is no minimum cycle basis containing $\{C_1, ..., C_i, C_{i+1}\}$. Hence, $\{C_1, C_2, ..., C_N\}$ is indeed a minimum cycle basis. □

2.2 Computing the Cycles

Given S_i, it is easy to compute a shortest cycle C_i such that $\langle C_i, S_i \rangle = 1$ by reducing it to n shortest path computations in an appropriate graph G_i. The following construction is well-known.

G_i has two copies v^+ and v^- of each vertex $v \in V$. For each edge $e = (u, v)$ in E do: if $e \notin S_i$, then add edges (u^+, v^+) and (u^-, v^-) to the edge set of G_i and assign their weights to be the same as e. If $e \in S_i$, then add edges (u^+, v^-) and (u^-, v^+) to the edge set of G_i and assign their weights to be the same as e. G_i can be visualised as 2 levels of G (the + level and the – level). Within each level, we have edges of $E \setminus S_i$. Between the levels we have the edges of S_i.

Given any v^+ to v^- path p in G_i, we can correspond to it a cycle in G by identifying the vertices and edges in G_i with their corresponding vertices and edges in G. Because we identify both v^+ and v^- with v, the path in G corresponding to p would be a cycle C.

More formally, take the incidence vector of the path p (over the edges of G_i) and obtain an incidence vector over the edges of G by identifying $(v^*, u^\dagger)$ with (v, u) where $*$ and $\dagger$ are $+$ or $-$. Suppose the path p contained more than one copy of some edge(s). (It could have contained both (v^+, u^-) and (v^-, u^+) for some (v, u).) Then add the number of occurrences of each such edge modulo 2 to obtain an incidence vector over the edges of G.

Let $p = min_{v \in V}$ shortest (v^+, v^-) path in G_i. The following lemma is simple to show.

Lemma 1. *The path p corresponds to a shortest cycle C in G that has odd intersection with S_i.*

The computation of the path p can be done by computing n shortest (v^+, v^-) paths (each by Dijkstra's algorithm) in G_i and taking their minimum or by one invocation of an all-pairs-shortest paths algorithm in G_i. This computation takes $O(n(m + n \log n))$ time. In the case when the edge weights are integers or the unweighted case it is better to use faster all-pairs-shortest paths algorithms than run Dijkstra's algorithm n times.

Since we have to compute totally N such cycles $C_1, C_2, ..., C_N$, we spend $O(mn(m + n \log n))$ time, since $N = m - n + 1$.

2.3 Computing the Subsets

We will now consider the problem of computing the subsets S_i, for $i = 1$ to N. S_i is a non-zero vector in the subspace orthogonal to $\{C_1, ..., C_{i-1}\}$. One way to find a non-zero vector in a subspace is to maintain the whole basis of the subspace. Any vector in that basis will then be a non-zero vector in the subspace.

Initially, $S_j = \{e_j\}$ for all j, $1 \leq j \leq N$. This corresponds to the standard basis of the space $\{0, 1\}^N$. At the beginning of phase i, we have $\{S_i, S_{i+1}, ..., S_N\}$ which is a basis of the space $\mathcal{C}^\perp$ orthogonal to the space $\mathcal{C}$ spanned by $\{C_1, ..., C_{i-1}\}$. We use S_i to compute C_i and update $\{S_{i+1}, ..., S_N\}$ to a basis $\{S'_{i+1}, ..., S'_N\}$ of the subspace of $\mathcal{C}^\perp$ which is orthogonal to C_i. The update step of phase i is as follows:

For $i + 1 \leq j \leq N$, let

$$S'_j = \begin{cases} S_j & \text{if } \langle C_i, S_j \rangle = 0 \\ S_j + S_i & \text{if } \langle C_i, S_j \rangle = 1 \end{cases}$$

The following lemma holds.

Lemma 2. *$S'_{i+1}, ...S'_N$ form a basis of the subspace orthogonal to $C_1, ..., C_i$.*

This completes the description of the algorithm SIMPLE-MCB.

Running Time of SIMPLE-MCB: During the update step of phase i, the cost of updating each $S_j, j > i$ is N and hence it is $N(N - i)$ for updating

$S_{i+1}, ..., S_N$. Since we have N phases, the total cost of maintaining this basis is about N^3, which is $O(m^3)$.

The total running time of the algorithm SIMPLE-MCB, by summing the costs of computing the cycles and witnesses, is $O(m^3 + mn^2 \log n)$. So, independent of which all-pairs-shortest-paths algorithm is used to compute the cycles, the cost of updating the witnesses is the bottleneck.

Note that in each phase we needed just one vector from the subspace orthogonal to $\{C_1, ..., C_i\}$. But the algorithm maintained $N-i$ such vectors: $S_{i+1}, ..., S_N$. This was the limiting factor in the running time of the algorithm.

3 Our Improvement

The maintenance of the basis of $\mathcal{C}^{\perp}$ costed us about m^2 in each iteration. In order to improve the running time of SIMPLE-MCB, we relax the invariant that $S_{i+1}, ..., S_N$ form a basis of the subspace orthogonal to $C_1, ..., C_i$. Since we need just one vector in this subspace, we can afford to relax this invariant and maintain the correctness of the algorithm. We will use a function *extend_cycle_basis* to compute the minimum cycle basis. This function works in a recursive manner.

The procedure *extend_cycle_basis*$(\{C_1, ..., C_i\}, \{S_{i+1}, \ldots, S_{i+k}\}, k)$ takes a partial basis $C_1, ..., C_i$ and k subsets $S_{i+1}, ..., S_{i+k}$ with the property that these subsets are all orthogonal to $C_1, ..., C_i$ and it recursively computes k new elements $C_{i+1}, ..., C_{i+k}$ of the minimum cycle basis. It first computes $C_{i+1}, ..., C_{i+\lfloor k/2 \rfloor}$ using $S_{i+1}, \ldots, S_{i+\lfloor k/2 \rfloor}$. Then it updates $S_{i+\lfloor k/2 \rfloor+1}, \ldots, S_{i+k}$ so that the updated sets are orthogonal to $C_{i+1}, ..., C_{i+\lfloor k/2 \rfloor}$ and they continue to be orthogonal to $C_1, ..., C_i$. Then it computes $C_{i+\lfloor k/2 \rfloor+1}, \ldots, C_{i+k}$. We present in Fig. 2 the overall algorithm FAST-MCB and the procedure *extend_cycle_basis*. Recall that the edges $e_1, ..., e_N$ are the edges of $G \setminus T$, where T is a spanning tree of G.

3.1 The Function *update:*

The function *update* is the key subroutine in our procedure *extend_cycle_basis*. After computing the cycles $C_{i+1}, ..., C_{i+\lfloor k/2 \rfloor}$, we call the function *update* with $\{S'_{i+1}, \ldots, S'_{i+\lfloor k/2 \rfloor}\}$ (the final versions of the subsets $S_{i+1}, \ldots, S_{i+\lfloor k/2 \rfloor}$) and $\{S_{i+\lfloor k/2 \rfloor+1}, ..., S_{i+k}\}$ as inputs. We want to update the sets $S_{i+\lfloor k/2 \rfloor+1}, ..., S_{i+k}$ so that the updated sets lie in the subspace orthogonal to the space spanned by $\mathcal{C} \cup \{C_{i+1}, ..., C_{i+\lfloor k/2 \rfloor}\}$. We know that $S_{i+\lfloor k/2 \rfloor+1}, ..., S_{i+k}$ are all orthogonal to $\mathcal{C}$ and now we need to ensure that the updated $S_{i+\lfloor k/2 \rfloor+1}, ..., S_{i+k}$ (call them $T_{i+\lfloor k/2 \rfloor+1}, \ldots, T_{i+k}$) are all orthogonal to $\mathcal{C} \cup \{C_{i+1}, ..., C_{i+\lfloor k/2 \rfloor}\}$.

We now want to update the sets $S_{i+\lfloor k/2 \rfloor+1}, ..., S_{i+k}$, i.e., we want to determine $T_{i+\lfloor k/2 \rfloor+1}, \ldots, T_{i+k}$ such that for each j in the range $i+\lfloor k/2 \rfloor+1 \leq j \leq i+k$ (i) T_j is orthogonal to $C_{i+1}, \ldots, C_{i+\lfloor k/2 \rfloor}$ and (ii) T_j continues to remain orthogonal to $C_1, ..., C_i$. So, we define T_j (for each $i + \lfloor k/2 \rfloor + 1 \leq j \leq i + k$) as follows:

$$T_j = S_j + \text{a linear combination of } S'_{i+1}, \ldots, S'_{i+\lfloor k/2 \rfloor}.$$

- Initialize the cycle basis with the empty set and initialize $S_j = \{e_j\}$ for $1 \leq j \leq N$.
- Call the procedure *extend_cycle_basis*$(\{\}, \{S_1, \ldots, S_N\}, N)$.

(A call to *extend_cycle_basis*$(\{C_1, ..., C_i\}, \{S_{i+1}, \ldots, S_{i+k}\}, k)$ extends the cycle basis by k cycles. $\mathcal{C}$ denotes the current partial cycle basis which is $\{C_1, ..., C_i\}$.)

The Procedure *extend_cycle_basis*$(\mathcal{C}, \{S_{i+1}, \ldots, S_{i+k}\}, k)$:

- if $k = 1$, compute a shortest cycle C_{i+1} such that $\langle C_{i+1}, S_{i+1} \rangle = 1$.
- if $k > 1$, use recursion.
  - call *extend_cycle_basis*$(\mathcal{C}, \{S_{i+1}, \ldots, S_{i+\lfloor k/2 \rfloor}\}, \lfloor k/2 \rfloor)$ to extend the current cycle basis by $\lfloor k/2 \rfloor$ elements. That is, the cycles $C_{i+1}, ..., C_{i+\lfloor k/2 \rfloor}$ are computed in a recursive manner. During the above recursive call, $S_{i+1}, ..., S_{i+\lfloor k/2 \rfloor}$ get updated. Call their final versions as $S'_{i+1}, ..., S'_{i+\lfloor k/2 \rfloor}$.
  - call *update*$(\{S'_{i+1}, \ldots, S'_{i+\lfloor k/2 \rfloor}\}, \{S_{i+\lfloor k/2 \rfloor+1}, ..., S_{i+k}\})$ to update $\{S_{i+\lfloor k/2 \rfloor+1}, ..., S_{i+k}\}$. Let $\{T_{i+\lfloor k/2 \rfloor+1}, ..., T_{i+k}\}$ be the output returned by *update*.
  - call *extend_cycle_basis*$(\mathcal{C} \cup \{C_{i+1}, ..., C_{i+\lfloor k/2 \rfloor}\}, \{T_{i+\lfloor k/2 \rfloor+1}, \ldots, T_{i+k}\}, \lceil k/2 \rceil)$ to extend the current cycle basis by $\lceil k/2 \rceil$ cycles. That is, the cycles $C_{i+\lfloor k/2 \rfloor+1}, ..., C_{i+k}$ will be computed recursively.

Fig. 2. FAST-MCB: A faster minimum cycle basis algorithm

This makes sure that T_j is orthogonal to the cycles $C_1, \ldots, C_i$ because S_j and all of $S'_{i+1}, ..., S'_{i+\lfloor k/2 \rfloor}$ are orthogonal to $C_1, \ldots, C_i$. Hence, T_j which is a linear combination of them will also be orthogonal to $C_1, \ldots, C_i$. The coefficients of the linear combination will be chosen such that T_j will be orthogonal to $C_{i+1}, \ldots, C_{i+\lfloor k/2 \rfloor}$.

Let

$$T_j = S_j + a_{j1}S'_{i+1} + a_{j2}S'_{i+2} + \cdots + a_{j\lfloor k/2 \rfloor}S'_{i+\lfloor k/2 \rfloor}.$$

We will determine the coefficients $a_{j1}, ..., a_{j\lfloor k/2 \rfloor}$ for all $i + \lfloor k/2 \rfloor + 1 \leq j \leq i + k$ simultaneously.

We want

$$\begin{pmatrix} T_{i+\lfloor k/2 \rfloor+1} \\ \vdots \\ \vdots \\ T_{i+k} \end{pmatrix} = (A\ I) \cdot \begin{pmatrix} S'_{i+1} \\ \cdots \\ S'_{i+\lfloor k/2 \rfloor} \\ S_{i+\lfloor k/2 \rfloor+1} \\ \cdots \\ S_{i+k} \end{pmatrix}$$

where A is a $\lceil k/2 \rceil \times \lfloor k/2 \rfloor$ matrix whose ℓth row has the unknowns $a_{j1}, ..., a_{j\lfloor k/2 \rfloor}$, where $j = i + \lfloor k/2 \rfloor + \ell$. And T_j represents a row with the coefficients of T_j as its row elements.

Let us multiply both sides of this equation with an $N \times \lfloor k/2 \rfloor$ matrix whose columns are the cycles $C_{i+1}, \ldots, C_{i+\lfloor k/2 \rfloor}$. That is,

$$\begin{pmatrix} T_{i+\lfloor k/2 \rfloor+1} \\ \vdots \\ \vdots \\ T_{i+k} \end{pmatrix} \cdot \left(C_{i+1}^T \ldots C_{i+\lfloor k/2 \rfloor}^T \right) = (A\ I) \cdot \begin{pmatrix} S'_{i+1} \\ \cdots \\ S'_{i+\lfloor k/2 \rfloor} \\ S_{i+\lfloor k/2 \rfloor+1} \\ \cdots \\ S_{i+k} \end{pmatrix} \cdot \left(C_{i+1}^T \ldots C_{i+\lfloor k/2 \rfloor}^T \right)$$

Then the left hand side is the 0 matrix since each of the vectors $T_{i+\lfloor k/2 \rfloor+1}, \ldots, T_{i+k}$ has to be orthogonal to each of $C_{i+1}, \ldots, C_{i+\lfloor k/2 \rfloor}$. Let

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} S'_{i+1} \\ \cdots \\ S'_{i+\lfloor k/2 \rfloor} \\ S_{i+\lfloor k/2 \rfloor+1} \\ \cdots \\ S_{i+k} \end{pmatrix} \cdot \left(C_{i+1}^T \ldots C_{i+\lfloor k/2 \rfloor}^T \right)$$

where

$$X = \begin{pmatrix} S'_{i+1} \\ \cdots \\ S'_{i+\lfloor k/2 \rfloor} \end{pmatrix} \cdot \left(C_{i+1}^T \ldots C_{i+\lfloor k/2 \rfloor}^T \right);\ Y = \begin{pmatrix} S_{i+\lfloor k/2 \rfloor+1} \\ \cdots \\ S_{i+k} \end{pmatrix} \cdot \left(C_{i+1}^T \ldots C_{i+\lfloor k/2 \rfloor}^T \right)$$

Then

$$0 = (A\ I) \cdot \begin{pmatrix} X \\ Y \end{pmatrix} = AX + Y$$

If X is invertible, then $A = -YX^{-1} = YX^{-1}$ since we are in $GF(2)$. We can then determine A in $O(k^\omega)$ time using fast matrix multiplication and inverse algorithms.

$$X = \begin{pmatrix} \langle S'_{i+1}, C_{i+1} \rangle & \ldots & \langle S'_{i+1}, C_{i+\lfloor k/2 \rfloor} \rangle \\ \langle S'_{i+2}, C_{i+1} \rangle & \ldots & \langle S'_{i+2}, C_{i+\lfloor k/2 \rfloor} \rangle \\ \vdots & \vdots & \vdots \\ \langle S'_{i+\lfloor k/2 \rfloor}, C_{i+1} \rangle & \ldots & \langle S'_{i+\lfloor k/2 \rfloor}, C_{i+\lfloor k/2 \rfloor} \rangle \end{pmatrix} = \begin{pmatrix} 1 * * \ldots * \\ 0\ 1 * \ldots * \\ 0\ 0\ 1 \ldots * \\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots \\ 0\ 0\ 0 \ldots 1 \end{pmatrix}$$

is an upper diagonal matrix with 1's on the diagonal, since each S'_j is the final version of the subset S_j using which C_j is computed, which means that $\langle S'_j, C_j \rangle = 1$ and $\langle S'_j, C_\ell \rangle = 0$ for all $\ell < j$. Hence, X is invertible. Thus $A = YX^{-1}$.

Lemma 3 follows from the implementation of the function *update.*

Lemma 3. *When $k = 1$, i.e., whenever we call extend_cycle_basis($\{C_1, \ldots, C_i\}$, $S_{i+1}, 1$), S_{i+1} is orthogonal to $\{C_1, \ldots, C_i\}$. And S_{i+1} always contains the edge e_{i+1}.*

Hence, just before we compute C_{i+1}, we always have a non-zero vector S_{i+1} orthogonal to $\{C_1, \ldots, C_i\}$. And C_{i+1} is a shortest cycle such that $\langle C_{i+1}, S_{i+1} \rangle$ is 1. Hence, the correctness of FAST-MCB follows then from Theorem 1.

3.2 The Running Time of FAST-MCB

The recurrence of our FAST-MCB algorithm is as follows:

$$T(k) = \begin{cases} \text{cost of computing a shortest odd cycle } C_i \text{ in } S_i & \text{if } k = 1 \\ 2T(k/2) + \text{cost of update} & \text{if } k > 1 \end{cases}$$

Cost of update: The computation of matrices X and Y takes time $O(mk^{\omega-1})$ using the fast matrix multiplication algorithm. We can also invert X in $O(k^{\omega})$ time and then we use fast matrix multiplication to multiply Y and X^{-1} to get the matrix A. Then we use fast matrix multiplication again to multiply the matrix $(A\ I)$ with the matrix whose rows are $S'_{i+1}, \ldots S_{i+k}$ to get the updated subsets $T_{i+\lfloor k/2 \rfloor+1}, \ldots T_{i+k}$. So the time required for all these computations is $O(mk^{\omega-1})$.

Using the algorithm described in Section 2.2 to compute a shortest cycle C_i that has odd intersection with S_i, the recurrence turns into

$$T(k) = \begin{cases} mn + n^2 \log n & \text{if } k = 1 \\ 2T(k/2) + O(k^{\omega-1}m) & \text{if } k > 1 \end{cases}$$

This solves to $T(k) = O(k(mn+n^2 \log n)+k^{\omega-1}m)$. Thus $T(m) = O(m^{\omega}+m^2n+mn^2 \log n)$. Since $m^{\omega} < m^2n$, this reduces to $T(m) = O(m^2n + mn^2 \log n)$. For $m > n \log n$, this is $T(m) = O(m^2n)$. For $m \leq n \log n$, this is $T(m) = O(mn^2 \log n)$.

Theorem 2. *A minimum cycle basis of an undirected weighted graph can be computed in time* $O(m^2n + mn^2 \log n)$.

Our algorithm has a running time of $O(m^{\omega} + m \cdot n(m + n \log n))$, where the $n(m + n \log n)$ term is the cost to compute all pairs shortest paths. This term can be replaced with a better term when the graph is unweighted or the edge weights are integers or when the graph is sparse. When the edges of G have integer weights, we can compute all pairs shortest paths in time $O(mn)$ [14,15], that is, we can bound $T(1)$ by $O(mn)$. When the graph is unweighted or the edge weights are small integers, we can compute all pairs shortest paths in time $\tilde{O}(n^{\omega})$ [12,6]. When such graphs are reasonably dense, say $m \geq n^{1+(1+\delta)/(\omega-1)}$ for some $\delta > 0$, then the m^{ω} term dominates the running time of our algorithm.

Theorem 3. *A minimum cycle basis in a graph with integer edge weights can be computed in time* $O(m^2n)$. *For unweighted graphs that satisfy* $m \geq n^{1+(1+\delta)/(\omega-1)}$ *for a constant* $\delta > 0$, *we have an* $O(m^{\omega})$ *algorithm to compute a minimum cycle basis.*

4 An Approximation Algorithm for Minimum Cycle Basis

The bottleneck in the running time of our minimum cycle basis algorithm is the computation of the shortest cycle C_i such that $\langle C_i, S_i \rangle = 1$. Suppose we relax

our constraint that our cycle basis should have minimum weight and ask for a cycle basis whose weight is at most α times the weight of an MCB. Then can we give a faster algorithm?

We show a positive answer to the above question. For any parameter $\alpha > 1$, we present below an approximation algorithm which computes a cycle basis whose weight is at most α times the weight of a minimum cycle basis. To the best of our knowledge, this is the first time that an approximation algorithm for the MCB problem is being given.

This algorithm is obtained by relaxing the base step ($k = 1$) in procedure *extend_cycle_basis* of our FAST-MCB algorithm (Fig. 2). In the original algorithm, we computed a shortest cycle C_i such that $\langle C_i, S_i \rangle = 1$. Here, we relax it to compute a cycle D_i such that $\langle D_i, S_i \rangle = 1$ and the weight of D_i is at most α times the weight of a shortest cycle that has odd intersection with S_i. The method of updating the subsets S_i would be identical to the way the updation is done in FAST-MCB.

A succinct description of our algorithm is given in Fig. 3.

For $i = 1$ to N do the following:

- Let S_i be any arbitrary non-zero vector in the subspace orthogonal to $\{D_1, D_2, ..., D_{i-1}\}$ i.e., S_i is a non-trivial solution to the set of equations: $\langle D_k, x \rangle = 0$ for $k = 1$ to $i - 1$.
- Compute a cycle D_i such that $\langle D_i, S_i \rangle = 1$ and the weight of $D_i \leq \alpha \cdot$ the weight of a shortest cycle that has odd intersection with S_i.

Fig. 3. APPROX-MCB: An α-approximate MCB

The linear independence of the D_i's follows from the existence of S_i's (by using S_i to show that D_i is linearly independent of $\{D_1, ..., D_{i-1}\}$). Similarly, note that the subsets $\{S_1, ..., S_N\}$ are linearly independent since each S_i is independent of $\{S_{i+1}, ..., S_N\}$ because $\langle S_i, D_i \rangle = 1$ whereas $\langle S_j, D_i \rangle = 0$ for each $j > i$.

4.1 Correctness of APPROX-MCB

Let $|C|$ denote the weight of cycle C. We need to show that $\sum_{i=1}^{N} |D_i| \leq \alpha \cdot$ weight of MCB. Let A_i be a shortest cycle that has odd intersection with S_i. The set $\{A_1, ..., A_N\}$ need not be linearly independent since the subsets S_i's were not updated according to the A_i's. The following lemma was originally shown in [5] in order to give an equivalent characterisation of the MCB problem as a maximisation problem. We present a simple proof of the lemma here.

Lemma 4. $\sum_{i=1}^{N} |A_i| \leq$ *weight of MCB.*

Proof. We will look at the A_i's in sorted order i.e., let π be a permutation on $[N]$ such that $|A_{\pi(1)}| \leq |A_{\pi(2)}| \leq ... \leq |A_{\pi(N)}|$. Let $\{C_1, ..., C_N\}$ be the cycles of an

MCB and let $|C_1| \leq |C_2| \leq ... \leq |C_N|$. We will show that for each i, $|A_{\pi(i)}| \leq |C_i|$. That will prove the lemma.

We will first show that $\langle C_k, S_{\pi(\ell)} \rangle = 1$ for some k and ℓ with $1 \leq k \leq i \leq \ell \leq N$. Otherwise, the $N-i+1$ linearly independent vectors $S_{\pi(i)}, S_{\pi(i+1)}, ..., S_{\pi(N)}$ belong to the subspace orthogonal to $C_1, ..., C_i$; however, this subspace has dimension only $N-i$. This means that $|A_{\pi(\ell)}| \leq |C_k|$ since $A_{\pi(\ell)}$ is a shortest cycle such that $\langle A_{\pi(\ell)}, S_{\pi(\ell)} \rangle = 1$. But by the sorted order, $|A_{\pi(i)}| \leq |A_{\pi(\ell)}|$ and $|C_k| \leq |C_i|$. This implies that $|A_{\pi(i)}| \leq |C_i|$. □

Since $|D_i| \leq \alpha \cdot |A_i|$ for each i, it follows from the above lemma that $\sum_{i=1}^{N} |D_i| \leq \alpha \cdot$ weight of MCB. Thus Theorem 4 follows.

Theorem 4. *The weight of the basis $\{D_1, ..., D_N\}$ computed by APPROX-MCB is at most α times the weight of a minimum cycle basis.*

4.2 The Running Time of APPROX-MCB

Since all the steps of APPROX-MCB, except the base step corresponding to computing a cycle, are identical to FAST-MCB, we have the following recurrence for APPROX-MCB:

$$T(k) = \begin{cases} \text{cost of computing an } \alpha \text{ stretch cycle } D_i \text{ that is odd in } S_i & \text{if } k = 1 \\ 2T(k/2) + O(k^{\omega-1}m) & \text{if } k > 1 \end{cases}$$

When $\alpha = 2$, we use the result in [3] to compute 2 stretch paths which would result in 2 stretch cycles. Then APPROX-MCB runs in time $\tilde{O}(m^{3/2}n^{3/2}) + O(m^{\omega})$. For reasonably dense graphs (say, $m \geq n^{(1.5+\delta)/(\omega-1.5)}$ for a constant $\delta > 0$), this is an $O(m^{\omega})$ algorithm.

For $1+\epsilon$ approximation, we use the all pairs $1+\epsilon$ stretch paths algorithm [16]. Then we have an $\tilde{O}(\frac{mn^{\omega}}{\epsilon} \log(W/\epsilon)) + O(m^{\omega})$ algorithm to compute a cycle basis which is at most $1+\epsilon$ times the weight of an MCB, where W is the largest edge weight in the graph. If $m \geq n^{1+(1+\delta)/(\omega-1)}$ for a constant $\delta > 0$ and all edge weights are polynomial in n, then APPROX-MCB is an $O(\frac{m^{\omega}}{\epsilon} \log(1/\epsilon))$ algorithm.

5 Computing a Certificate of Optimality

Given a set of cycles $\mathcal{C} = \{C_1, ..., C_N\}$ we would like to construct a certificate to verify the claim that $\mathcal{C}$ forms an MCB. A certificate is an "easy to verify" witness of the optimality of our answer. For example, the sets S_i, $1 \leq i \leq N$ in our algorithm from which we calculate the cycles $\mathcal{C} = \{C_1, ..., C_N\}$ of the minimum cycle basis, are a certificate of the optimality of $\mathcal{C}$. The verification algorithm would then consist of verifying that the cycles in $\mathcal{C}$ are linearly independent and that each C_i is a shortest cycle such that $\langle C_i, S_i \rangle = 1$.

Theorem 5. *Given a set of cycles $\mathcal{C} = \{C_1, ..., C_N\}$ we can construct a certificate $\{S_1, ..., S_N\}$ in $O(m^{\omega})$ time.*

The above theorem follows from a simple algorithm that inverts an $N \times N$ matrix whose rows are the incidence vectors of $C_1, ..., C_N$ over the edges *of* $G \backslash T$, where T is a spanning tree of G.

Acknowledgment. We wish to thank Jaikumar Radhakrishnan for his helpful comments.

References

1. A. C. Cassell, J. C. Henderson, and K. Ramachandran. Cycle bases of minimal measure for the structural analysis of skeletal structures by the flexibility method. In *Proc. Royal Society of London Series A,* volume 350, pages 61–70, 1976.
2. L. O. Chua and L. Chen. On optimally sparse cycle and coboundary basis for a linear graph. In *IEEE Trans. Circuit Theory,* volume CT-20, pages 495–503, 1973.
3. E. Cohen and U. Zwick. All-pairs small-stretch paths. *Journal of Algorithms,* 38:335–353, 2001.
4. D. Coppersmith and S. Winograd. Matrix multiplications via arithmetic progressions. *Journal of Symb. Comput.,* 9:251–280, 1990.
5. J.C. de Pina. *Applications of Shortest Path Methods.* PhD thesis, University of Amsterdam, Netherlands, 1995.
6. Z. Galil and O. Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computing Systems and Sciences,* 54:243–254, 1997.
7. Alexander Golynski and Joseph D. Horton. A polynomial time algorithm to find the minimum cycle basis of a regular matroid. In *8th Scandinavian Workshop on Algorithm Theory,* 2002.
8. J. D. Horton. A polynomial-time algorithm to find a shortest cycle basis of a graph. *SIAM Journal of Computing,* 16:359–366, 1987.
9. E. Hubicka and M. M. Syslo. Minimal bases of cycles of a graph. In M. Fiedler, editor, *Recent Advances in Graph Theory,* pages 283–293, 1975.
10. E. Kolasinska. On a minimum cycle basis of a graph. *Zastos. Mat.,* 16:631–639, 1980.
11. Padberg and Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research,* 7:67–80, 1982.
12. R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computing Systems and Sciences,* 51:400–403, 1995.
13. G. F. Stepanec. Basis systems of vector cycles with extremal properties in graphs. *Uspekhi Mat. Nauk,* 19:171–175, 1964.
14. M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM,* 46:362–394, 1999.
15. M. Thorup. Floats, integers, and single source shortest paths. *Journal of Algorithms,* 35:189–201, 2000.
16. U. Zwick. All pairs shortest paths in weighted directed graphs - exact and approximate algorithms. In *Proc. of the 39th Annual IEEE FOCS,* pages 310–319, 1998.
17. A. A. Zykov. *Theory of Finite Graphs.* Nauka, Novosibirsk, 1969.

The Black-Box Complexity of Nearest Neighbor Search

Robert Krauthgamer[1]* and James R. Lee[2]**

[1] IBM Almaden Research Center, 650 Harry Road, San Jose CA 95120, USA.
robi@almaden.ibm.com
[2] Computer Science Division, U.C. Berkeley, Berkeley, CA 94720, USA.
jrl@cs.berkeley.edu

Abstract. We define a natural notion of efficiency for approximate nearest-neighbor (ANN) search in general n-point metric spaces, namely the existence of a randomized algorithm which answers $(1+\varepsilon)$-approximate nearest neighbor queries in $\mathrm{polylog}(n)$ time using only polynomial space. We then study which families of metric spaces admit efficient ANN schemes in the black-box model, where only oracle access to the distance function is given, and any query consistent with the triangle inequality may be asked.

For $\varepsilon < \frac{2}{5}$, we offer a complete answer to this problem. Using the notion of metric dimension defined in [GKL03] (à la [Ass83]), we show that a metric space X admits an efficient $(1+\varepsilon)$-ANN scheme for any $\varepsilon < \frac{2}{5}$ *if and only if* $\dim(X) = O(\log \log n)$. For coarser approximations, clearly the upper bound continues to hold, but there is a threshold at which our lower bound breaks down—this is precisely when points in the "ambient space" may begin to affect the complexity of "hard" subspaces $S \subseteq X$. Indeed, we give examples which show that $\dim(X)$ does not characterize the black-box complexity of ANN above the threshold.

Our scheme for ANN in low-dimensional metric spaces is the first to yield efficient algorithms without relying on any additional assumptions on the input. In previous approaches (e.g., [Cla99,KR02,KL04]), even spaces with $\dim(X) = O(1)$ sometimes required $\Omega(n)$ query times.

1 Introduction

Nearest-neighbor search. Nearest-neighbor search (NNS) is the problem of preprocessing a set X of n points lying in a huge (possibly infinite) metric space (M, d) so that given a query $q \in M$, one can efficiently locate the nearest point to q among the points in X. Computing such nearest neighbors efficiently is a classical and fundamental problem with numerous practical applications. These include data compression, database queries, machine learning, computational biology, data mining, pattern recognition, and ad-hoc networks. A common feature

* Part of this work was done while this author was with the International Computer Science Institute and with the Computer Science Division of U.C. Berkeley.

** Supported by NSF grant CCR-0121555 and an NSF Graduate Research Fellowship.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 858–869, 2004.

of many of these examples is that comparing two elements is costly, hence the number of distance computations should be made as small as possible.

Most previous research has focused on the important special case when $M = \mathbb{R}^d$ and distances are computed according to some ℓ_p norm. While many types of data can be naturally represented in such a form, this is certainly not true for a significant number of applications, and it is therefore desirable to address NNS in general metric spaces. On the other hand, data structures for general metrics might perform a nearest neighbor query in time as poorly as $\Omega(n)$ which is unacceptable in practice. Such a dependence is inherent even when only approximate solutions are required. A well-known example is where X forms a uniform metric, so that the interpoint distances in X are all equal, providing essentially no information.

Metric dimension. Given this state of affairs, an increasing amount of recent attention has focused on understanding the complexity of NNS in terms of a metric's implicit structure. In Euclidean spaces, an obvious and common measure for a metric's complexity is the dimension of the Euclidean host space. Thus it is natural that to characterize the complexity of general metric spaces, one ought to define an analogous notion of *metric dimension,* and indeed this approach has been pursued to great success in recent papers [Cla99,KR02,KL04,HKMR04], where significant progress on solving exact and approximate versions of the NNS problem in general metrics has been made.

Unfortunately, each of these works falls short of offering the sort of generality that one should desire from such an approach. In [Cla99], to achieve efficient algorithms (for *exact* NNS), it is necessary to make strong assumptions about the distribution of queries. In [KR02,HKMR04], the notion of dimension is too restrictive, eliminating large classes of metric spaces which should be considered low-dimensional, and for which efficient algorithms should exist (see [KL04] for a more detailed explanation).

Finally, in [KL04], a more satisfying notion of dimension (taken from [GKL03], and independently used in a different form by [Cla99]) is proposed, but the algorithms in both [KL04] and [Cla99] are efficient only under the additional assumption that the *aspect ratio* Φ (i.e. the ratio of the largest to smallest distance in X) is at most polynomial in $n = |X|$. In particular, the algorithm presented in [KL04] answers approximate nearest neighbor queries in time $2^{O(\dim(X))} \log \Phi$. Thus even when the set of points is $X = \{1, 2, 4, \ldots, 2^n\} \subseteq \mathbb{R}$ with the line metric $d(x, y) = |x - y|$, as in Figure 1, the algorithms of [KL04], as well as those of [Cla99,KR02,HKMR04], require $\Omega(n)$ time to answer some queries (i.e. they are no better than the trivial algorithm which tests every point). Despite the fact that (X, d) is clearly "low-dimensional" (being a subset of the real line), previous approaches perform dismally. Besides being theoretically disappointing, these algorithms are incapable of searching for (even approximate) nearest neighbors in highly clustered data (e.g. 1).

Efficient algorithms in the black-box model. In the present work, we are concerned with *approximate nearest neighbor search* (ANN). The $(1 + \varepsilon)$-ANN

problem is defined as follows: Given a query $q \in M$, we are required to return an element $a \in X$ for which $d(q,a) \leq (1+\varepsilon)\, d(q,X)$, where $d(q,X)$ is the distance from q to the closest point in X. (This is after an initial preprocessing stage.) We resolve the aforementioned shortcomings by presenting an ANN data structure for general metric spaces which is efficient whenever $\dim(X)$ (defined formally in Section 1.2) is small, and under no additional assumptions.

We will discuss our definition of "efficient" momentarily. Beforehand, let us describe another question that arises in the above framework: Is "dimension" the right notion to characterize the complexity of ANN in general metric spaces? Certainly one may motivate the study of algorithms for "low-dimensional" metrics by their abundance in practical settings (see [KL04]), but one should also consider how tightly $\dim(X)$ captures the difficulty of nearest neighbor search in general metrics. To this end, we consider a black-box model of nearest neighbor search in arbitrary metric spaces, where the query is accessed as an oracle via the distance function. We say that a metric space X admits an efficient $(1+\varepsilon)$-ANN scheme if there exists a (possibly randomized) algorithm which answers any possible $(1+\varepsilon)$-approximate nearest neighbor query in $\mathrm{polylog}(n)$ time using only polynomial space (although quite a natural definition, we offer additional motivation in Section 1.2, where the model is specified more precisely).

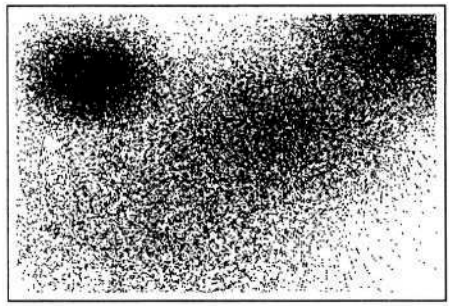
Under this complexity regime, we show that for every $\varepsilon < \frac{2}{5}$, a metric space X admits an efficient $(1+\varepsilon)$-ANN scheme *if and only if* $\dim(X) = O(\log\log n)$, and thus below a certain approximation threshold, $\dim(X)$ captures precisely the complexity of the problem. The behavior above the threshold is quite different, and we demonstrate in Section 2 that for sufficiently coarse approximations, the "ambient space" begins to play a crucial role, at which point even metrics of very high dimension may become tractable. We note that the constants of these constructions are not optimized; our primary goal is simply to show the existence of an approximation threshold, on either side of which drastically different behaviors are exhibited.

1.1 Results and Techniques

Upper bounds. We give the first efficient (i.e. $\mathrm{polylog}(n)$ query time and $\mathrm{poly}(n)$ space) $(1+\varepsilon)$-ANN scheme for metrics whose doubling dimension, denoted by $\dim(X)$ (and defined in Section 1.2), is small. In particular, these bounds hold for any fixed $\varepsilon > 0$ and whenever $\dim(X) = O(\log\log n)$. For instance, in the case where $\dim(X) = O(1)$, our algorithm answers queries in $O(\log^2 n)$ time and $O(n^2)$ space, while making only $O(\log n)$ calls to the distance function. We note that the space bound we achieve for arbitrary metrics—obtained in Section 3.2—is a polynomial whose degree is *independent* of $\dim(X)$ and the desired approximation. Indeed, our data structure can be built without knowledge of ε (which can be passed as a parameter at query-time). When $\dim(X)$ is small, a general reduction from $O(1)$-ANN to $(1+\varepsilon)$-ANN follows easily from the techniques of [KL04], which we review in Section 1.2.

Our data structure is based on two new techniques. The first is a structural theorem about the existence of "dense," "well-separated" clusters of points in

(a) The doubling line



(b) Mixture of gaussians in the plane

Fig. 1. Clustered point sets

low-dimensional metrics. These sets manifest themselves in the form of *ring separators*— "thick" annuli whose inside and outside each contain a large fraction of the points. (A similar object is used in the construction of the ring-cover trees of [IM98] which are used to solve ANN in $\mathbb{R}^d$. Our setting is quite different, since we are not reducing to the problem of point location in equal balls. Hence we must show that for low-dimensional metrics, ring separators exist unconditionally.) Using these separators, we build a binary decision tree of height $2^{O(\dim(X))} \log n$ which can be used to answer $O(1)$-ANN queries in time $2^{O(\dim(X))} \log n$. Unfortunately, the natural implementation of this tree requires space $n^{2^{2\dim(X)}}$, which is $n^{\Omega(\text{polylog}(n))}$ even when $\dim(X) = \Theta(\log \log n)$.

This exponential blow-up in space is a typical problem encountered in NNS algorithms based on metric decompositions, and is the most difficult technical challenge faced by the present work. In Section 3.2, we overcome this problem for low-dimensional metrics, obtaining a data structure that uses $O(n^2 \text{ polylog}(n))$ space whenever $\dim(X) = O(\log \log n)$. In addition, even for arbitrary spaces (with no bound on the dimension), the space consumption is only polynomial. This improvement requires a second new technique which amounts to "folding" the decision tree back onto itself, often merging many apparently distinct branches into one. The difficulties and solutions are discussed more thoroughly in Section 3.2. This folding allows us to obtain a very compact "approximate" representation of the previously huge decision tree, while incurring only a small additional overhead at every node.

We note that since the doubling dimension was introduced in [GKL03], and the premise that "low-dimensional" general metrics should be algorithmically tractable was set forth, an increasing number of works have found applications of this idea to optimization problems; we mention, in particular, the predecessor to this work [KL04] and the new results of [Tal04] for approximating problems like TSP in low-dimensional metrics. We also mention the related work

of [KKL03] in machine learning. We believe that the development and discovery of the rich properties of low-dimensional metrics continued herein will find additional application elsewhere.

Lower bounds. Our lower bounds are entirely information theoretic. Given a metric space X, there exists a "nearly-uniform" submetric $S \subseteq X$ whose size is roughly $k \geq 2^{\Omega(\dim(X))}$. Nearly uniform means that the aspect ratio (the ratio of the largest pairwise distance in S to the smallest) is bounded by a small constant. In Section 2, we then prove that, for every $\delta > 0$, this "large" subset S must contain within itself a subset $T \subseteq S$ with very small aspect ratio, i.e. $2 + \delta$, and yet which satisfies $|T| \geq k^{\delta'}$ (where δ' depends on δ, of course). This is a very simple (yet interesting) Ramsey-like property of metric spaces.

Now, if an algorithm were not allowed to compute distances from the query to $X \setminus T$ (i.e. the "ambient space"), then a lower bound of $\Omega(k^{\delta'})$ queries for $(1+\delta)$-ANN would follow fairly easily for T. And indeed, by a slightly technical extension argument, we can prove that any algorithm solving the $(1+\varepsilon)$-ANN problem must make at last $2^{\Omega(\dim(X))}$ queries to the distance oracle for $\varepsilon < \frac{2}{5}$. This shows that in the black-box model, querying against the ambient space cannot help too much when one requires a sufficiently fine approximation.

But our lower bound breaks down for coarser approximations, and we show that this is for good reason: When only a 3-approximation is desired, there are n-point metrics X with $\dim(X) = \Omega(\log n)$ for which every query against X can be decided in $O(\log n)$ time in the black-box model. Thus above a certain approximation threshold, $\dim(X)$ no longer characterizes the complexity of ANN.

1.2 Preliminaries

Metric spaces. Let (X, d) be an n-point metric space, and let $S \subseteq X$ be a subset. We denote by

$$B_S(x, r) = \{y \in S : d(x, y) < r\}$$

the open ball of radius r about x in S. When $S = X$, we omit the subscript S. We write $d(x, S) = \inf_{y \in S} d(x, y)$. Define $\operatorname{diam}(S) = \sup_{x,y \in S} d(x, y)$, and let the *aspect ratio* of S be the quantity

$$\Phi(S) = \frac{\operatorname{diam}(S)}{\inf_{x,y \in S} d(x, y)}.$$

Finally, we say that a subset Y of X is a *β-net* if it satisfies (1) For every $x, y \in Y, d(x, y) \geq \beta$ and (2) $X \subseteq \bigcup_{y \in Y} B(y, \beta)$. Such nets always exist for any $\beta > 0$. For finite metrics, they can be constructed greedily. For arbitrary metrics, proof of their existence is an easy application of Zorn's lemma.

The doubling dimension. We recall that the *doubling constant* $\lambda(X)$ is the least value λ such that every ball in X can be covered by λ balls of half the radius. The *doubling dimension* [GKL03] is then defined by $\dim(X) = \log_2 \lambda(X)$. Here are some simple properties which demonstrate that $\dim(X)$ is a robust and meaningful notion.

1. For $X = \mathbb{R}^k$ equipped with any norm, $\dim(X) = \Theta(k)$.
2. If $S \subseteq X$, then $\dim(S) \leq 2 \cdot \dim(X)$.
   (Using a slightly different definition of $\dim(X)$ which is equivalent up to a constant, one can ensure that $\dim(S) \leq \dim(X)$.)
3. $\dim(X_1 \cup \cdots \cup X_m) \leq \max_i \dim(X_i) + \log m$.
   (In particular, $\dim(X) \leq \log |X|$.)

The following simple lemma is important.

Lemma 1 (Nearly-uniform metrics). *Let (X, d) be a metric space, and let $S \subseteq X$. If the aspect ratio of the metric induced on S is at most $\Phi \geq 2$, then $|S| \leq \Phi^{O(\dim(X))}$.*

Proof. Let $d_{\min} = \inf\{d(x,y) : x, y \in S\}$ and $d_{\max} = \sup\{d(x,y) : x, y \in S\}$ be the minimum and maximum interpoint distance in S, respectively, and assume that $\Phi = \frac{d_{\max}}{d_{\min}} < \infty$. Notice that S is contained in a ball of radius $2\,d_{\max} \leq 2\Phi\, d_{\min}$ in X (centered at any point of S). Applying the definition of doubling dimension iteratively several times we get that this ball, and in particular S, can be covered by $2^{\dim(X) \cdot O(\log \Phi)}$ balls of radius $d_{\min}/3$. Each of these balls can cover at most one point of S (by definition of $d_{\min}$) and thus $|S| \leq 2^{\dim(X) \cdot O(\log \Phi)} \leq \Phi^{O(\dim(X))}$.

In particular, we observe that the above lemma provides a bound on the cardinality of a δR-net intersected with a ball of radius R. Namely, such an intersection contains at most $(\frac{1}{\delta})^{O(\dim(X))}$ points.

The black-box model and efficiency. Our model is quite simple. Suppose that (X, d) is a metric space. We assume that the only thing known about the query (and thus the only constraint on the query) is that the space $(X \cup \{q\}, d)$ is again a metric space, i.e. that the query does not violate the triangle inequality. The only access that an algorithm has to the query is through oracle calls to the distance function, i.e. the values $d(q, x)$ for $x \in X$. We assume that $d(q, \cdot)$ can be evaluated in unit time (although this is without loss of generality, since our upper bounds scale linearly with the time needed to evaluate the distance function, and our lower bounds are in terms of the number of calls to $d(q, \cdot)$).

We are defining an algorithm as "efficient" if, after the preprocessing phase, it can answer any query in $\mathrm{polylog}(n)$ time using only $\mathrm{poly}(n)$ space. We don't make any restriction on preprocessing time or space, but we note that in all of our upper bounds, both are linear in the space used by the algorithm for answering a query.

As for the running time, we note that all of the algorithms in [Cla99,KR02, KL04] strive for $\mathrm{polylog}(n)$ query times, thus it is *the* natural candidate for "efficiency." We also note that the best algorithms for ANN in high-dimensional Euclidean spaces answer queries in $\mathrm{polylog}(n)$ time [IM98,KOR98,H01]. As for space, $\mathrm{poly}(n)$ is again the natural choice, but this assumption should not be abused. Straightforward implementations of the algorithms of [IM98] and [KOR98], although outstanding theoretical achievements, are hampered due to

their extremely high space complexity (the degree of the polynomial grows with $\frac{1}{\varepsilon}$ for $(1+\varepsilon)$-ANN).

Even in the worst case (i.e. when $\dim(X) = \Omega(\log n)$), the algorithms of Section 3.2 use only $\mathrm{poly}(n)$ space (independent of the approximation factor desired). When $\dim(X) = O(\log\log n)$, the space consumption is $O(n^2\,\mathrm{polylog}(n))$. This factor has not been optimized, and we hope that eventually a near-linear space algorithm can be obtained, at least for the case when $\dim(X) = O(1)$.

The [KL04] reduction to $O(1)$-ANN. In [KL04], it is shown that, using only $2^{O(\dim(X))} \cdot n$ space, one can maintain a data structure which, given a query q, converts any $(1+\alpha)$-ANN to q to a $(1+\varepsilon)$-ANN using time $(\alpha/\varepsilon)^{O(\dim(X))}$. (In essence, one can do "brute-force" search around the $O(1)$-ANN. The number of nearby points in a $(1+\varepsilon)$-net is only exponential in the doubling dimension.) This term is only $\mathrm{polylog}(n)$ whenever $\dim(X) = O(\log\log n)$ and $\alpha/\varepsilon = O(1)$, thus we content ourselves with finding $O(1)$-ANNs in everything that follows. One actually needs to maintain pointers from the data structure in the current paper, to that borrowed from [KL04], but this is a minor issue which we ignore in the current version.

2 Lower Bounds

In this section, we show that for any metric space X and any fixed $\varepsilon < \frac{2}{5}$, solving the $(1+\varepsilon)$-ANN problem on X is as hard as unordered search in a k-element database with $k = 2^{\Omega(\dim(X))}$. It will follow that any algorithm (deterministic or randomized) which solves the $(1+\varepsilon)$-ANN problem on X must make at least $2^{\Omega(\dim(X))}$ calls to the distance oracle for some query q. We note that the constructions of this section are not optimized; our goal is simply to show the existence of an approximation threshold, on either side of which drastically different behaviors are exhibited.

Theorem 1. *For any metric space X and any fixed $\varepsilon < \frac{2}{5}$, any algorithm solving the $(1+\varepsilon)$-ANN problem on X must make at least $2^{\Omega(\dim(X))}$ calls to the distance oracle for some query q. For randomized algorithms, this bound holds in expectation.*

First, we require a partial converse to Lemma 1.

Lemma 2. *For any n-point metric space X and any $2 < \Phi_0 \le 4$, there exists a subset $S \subseteq X$ with $\Phi(S) \le \Phi_0$ and $|S| \ge 2^{\dim(X)(\log \Phi_0 - 1)}$.*

The full proof is deferred to the full version. The basic idea is that, if every subset of small aspect ratio is small, this yields a recipe for covering every large ball by smaller balls (and hence the dimension must be small).

Theorem 2. *Let (X,d) be any metric space which contains a submetric $S \subseteq X$ with $\Phi(S) \le \Phi_0$ and $|S| \ge k$. Then for every $\varepsilon < (\Phi_0 + \frac{1}{2})^{-1}$, any algorithm for $(1+\varepsilon)$-ANN on X must make at least $\Omega(k)$ calls to the distance oracle for some query q. For randomized algorithms, this holds in expectation.*

Proof (Sketch). Let $S = \{x_1, x_2, \ldots, x_k\}$. Let $d_{\max} = \max_{x,y \in S} d(x,y)$ and $d_{\min} = \min_{x,y \in S} d(x,y)$. To each index $i \in \{1, \ldots, k\}$, we associate a query q_i which satisfies:

$$d(q_i, y) = \begin{cases} \frac{1}{2}d_{\max} + \frac{1}{4}d_{\min} + d(y, x_i) & \text{if } d(y, x_i) < \frac{1}{2}d_{\min} \\ \frac{1}{2}d_{\max} + \frac{3}{4}d_{\min} & \text{if } d(y, x_i) \geq \frac{1}{2}d_{\min} \text{ and } d(y, S) \leq \frac{1}{2}d_{\min} \\ \frac{1}{2}d_{\max} + \frac{1}{4}d_{\min} + d(y, S) & \text{otherwise} \end{cases}$$

First, one must assure that the space $(X \cup \{q_i\}, d)$ satisfies the triangle inequality for every $1 \leq i \leq k$. Then, one shows that for $\varepsilon = \varepsilon(\Phi_0)$ small enough, finding a $(1+\varepsilon)$-ANN to q_i is equivalent to guessing the value of i. The lower bound of $\Omega(k)$ follows. These proofs are deferred to the full version.

Now we prove the main theorem of this section.

Proof (of Theorem 1). Let (X, d) be any metric space. As $\Phi_0 \to 2$ in Lemma 2, the lower bound value of ε to which the preceding theorem applies behaves like $\varepsilon < (\Phi_0 + \frac{1}{2})^{-1} \to \frac{2}{5}$. Thus for any fixed $\varepsilon < \frac{2}{5}$, there is a lower bound of $2^{\Omega(\dim(X))}$ on the number of calls to the distance function which are needed to answer some $(1+\varepsilon)$-ANN query.

We obtain the following corollary.

Corollary 1. *If* $\dim(X) = \omega(\log \log n)$ *and* $\varepsilon < \frac{2}{5}$, *then there is no efficient* $(1+\varepsilon)$*-ANN scheme for X in the black-box model, since* $2^{\Omega(\dim(X))}$ *is bigger than any* $\text{polylog}(n)$.

2.1 Above the Threshold

In this section, we show that when coarser approximations are desired, there are metrics of high dimension which nevertheless admit very efficient ANN algorithms, and thus the lower bounds of the previous section cannot be pushed too much further. Again, we do not seek to optimize constants.

Let $A = \{e_1, \ldots, e_n\}$ where $e_i \in \mathbb{R}^n$ is an n-dimensional vector with a 1 in the ith coordinate and zeros elsewhere. Additionally, let B consist of $k = O(\log n)$ vectors chosen at random by setting each of their coordinates to be -1 or $+1$ with equal probabilities (independent of all other coordinates). We endow these points with the ℓ_∞ metric, i.e., for any two vectors $u, v \in \mathbb{R}^n$, let $d(u,v) = \|u - v\|_\infty = \max_{1 \leq i \leq n} |u_i - v_i|$ (where v_i is the ith coordinate of v).

Let $X = A \cup B$ be the set of points to be preprocessed. Clearly, X contains the n-point uniform metric A, and thus $\dim(X) \geq \dim(A) = \log n$. On the other hand, $|X| \leq 2n$ and thus $\dim(X) \leq 1 + \log n$. However, it is not difficult to verify that, with high probability (over the choice of B), there exists a 3-ANN algorithm for X. We omit the description from this version.

3 Efficient Algorithms

We provide two algorithms for $(1+\varepsilon)$ approximate nearest neighbor search in a general metric space; the two have similar query time, but they differ in their space requirement. By the general reduction discussed in Section 1.2, it suffices to exhibit an $O(1)$-ANN algorithm. Our first algorithm (Section 3.1) *is* based on the existence of a certain ring-separator, which naturally yields a binary decision tree that can be used to solve 3-ANN. The decision tree's depth is $2^{O(\dim(X))}\log n$, so this algorithm has an excellent (actually optimal) query time. However, it's space requirement grows rapidly with $\dim(X)$. The second algorithm, which achieves space that is polynomial in n (independently of $\dim(X)$) is significantly more complex, and we refer the reader to Section 3.2 for a discussion of the subtle issues which arise.

3.1 The Ring-Separator Tree

The basic notion introduced in this subsection is that of a ring-separator; this naturally yields a ring-separator tree, which can be used as a binary decision tree for 3-ANN. Throughout this section, we shall use the following definition. For $x \in S \subseteq X$ and $R_1, R_2 \geq 0$, define the *annulus about* x as

$$A_S(x, R_1, R_2) \stackrel{\text{def}}{=} B_S(x, R_2) \setminus B_S(x, R_1).$$

The ring-separator. Let (X,d) be an n-point metric space. A *δ-ring-separator* of a subset $S \subseteq X$ is a pair (x, R) consisting of a point $x \in S$ and a real $R \geq 0$, that satisfies the following condition: $|B_S(x,R)| \geq \delta|S|$ and yet $|B_S(x,2R)| \leq (1-\delta)|S|$. We now prove the main lemma of this subsection.

Lemma 3 (Ring separators). *For any metric space (X,d) and any subset $S \subseteq X$ with $|S| \geq 2$, there exists a δ-ring-separator of S with $\delta \geq (\frac{1}{2})^{O(\dim(X))}$.*

Proof. We proceed by contradiction. Fix some $0 < \delta < 1$ and assume that S does not have a δ-ring-separator; we will show that for a sufficiently large constant $c > 0$, $\delta > (\frac{1}{2})^{c\dim(X)}$, thus proving the lemma.

Let $\bar{B}_S(x,r) = \{y \in S : d(x,y) \leq r\}$ be the *closed ball* of radius r around x (in S). For every point $x \in S$, let $R(x) \stackrel{\text{def}}{=} \inf\{R \geq 0 : |\bar{B}_S(x,R)| \geq \delta|S|\}$. Since $|S| \geq 2$ is finite, $R(x)$ is defined and furthermore $|B_S(x,R(x))| < \delta|S|$. By our assumption, for all $x \in X$, $|B_S(x,2R(x))| > (1-\delta)|S|$, and hence each annulus $A_S(x_i, R(x_i), 2R(x_i))$ contains at least $(1-2\delta)|S|$ points.

Let $x_0 \in S$ be the point for which $R(x_0)$ is minimal, and iteratively for $t = 1, 2, \ldots$ choose $x_t \in S$ to be an arbitrary point of

$$\bigcap_{i=0}^{t-1} A_S(x_i, R(x_i), 2R(x_i)).$$

Clearly we can continue this process as long as the above intersection remains non-empty. Suppose we are forced to stop after selecting k points

$x_0, x_1, \ldots, x_{k-1}$. On the one hand, we threw away at most $2\delta|S|$ points at every step, and thus $k \geq 1/2\delta$. On the other hand, the set $U = \{x_0, x_1, \ldots, x_{k-1}\}$ is contained in $B(x_0, 2R(x_0))$. Furthermore, for any pair x_i, x_j with $i < j$, we see that $d(x_i, x_j) \geq R(x_i)$ since $x_j \notin B_S(x_i, R(x_i))$. But by construction, $R(x_i) \geq R(x_0)$ for all $i \geq 0$. It follows that the set U has aspect ratio at most 4, and thus by Lemma 1, $k \leq 2^{O(\dim(X))}$. We conclude that $\delta \geq 1/2k \geq (\frac{1}{2})^{O(\dim(X))}$.

The ring-separator tree. Given the above lemma, it is natural to define a *δ-ring-separator tree* for a metric space (X, d). This is a binary tree where each node has a label $S \subseteq X$, constructed recursively as follows. The root of the tree has the label $S = X$. A node labelled by S is a leaf if $|S| = 1$, and has two children if $|S| \geq 2$. In the latter case, we take (x, R) to be a δ-ring-separator of S, and add under the node an *inside child,* whose label is $S_I = B_S(x, 2R)$, and an *outside child,* whose label is $S_O = S \setminus B_S(x, R)$. Note that S_I and S_O are not a partition of S, as their intersection is generally non-empty. Let us add the ring-separator into the node's label and say that the non-leaf node's label is $\langle S, (x, R)\rangle$(where $S \subseteq X$, $x \in S$ and $R > 0$). Lemma 3 shows that if $|S| \geq 2$ then S admits a δ-ring-separator with $\delta \geq (\frac{1}{2})^{O(\dim(X))}$. Since every step decreases the size of S by a factor of at least $1 - \delta$, the height of the tree is at most $2^{O(\dim(X))} \log n$.

The 3-ANN algorithm. We now show how to use ring-separator trees to solve the 3-ANN problem on X in time $2^{O(\dim(X))} \log n$. Unfortunately, a bound of $2^{O(\dim(X))} \log n$ on the height of the ring-separator tree implies a possibly huge space requirement of $n^{2^{O(\dim(X))}}$. This problem will be remedied in Section 3.2.

Let q be the query against X. The algorithm proceeds along a root to leaf path, i.e., starts at the root and recursively goes down the tree until a leaf node is met. Suppose that we are at a node $N = \langle S, (x, R)\rangle$. If $d(q, x) \leq 3R/2$, the algorithm proceeds to the inside child of N; otherwise, it proceeds to the outside child. Eventually, a leaf node $N = \langle\{x\}\rangle$ is met. Let x_i be the point x seen in the ith node along this root to leaf path (either the point from the ring-separator or the only point in S). Then the algorithm outputs the point which is closest to q among the points x_i.

This algorithm clearly runs in time linear in the height of the tree, i.e. $2^{O(\dim(X))} \log n$. We now proceed to show that the point x_i which is output is indeed a 3-approximate nearest neighbor to q.

Proposition 1. *The above algorithm outputs a 3-approximate nearest neighbor to q.*

Proof. Let $a^* \in X$ be the real nearest neighbor to q, i.e. $d(q, X) = d(q, a^*)$. Let $N_1, N_2, \ldots, N_k$ be the sequence of tree nodes seen by the algorithm on input q. For $i < k$ let $N_i = \langle S_i, (x_i, R_i)\rangle$, and let $N_k = \langle\{x_k\}\rangle$. Clearly $a^* \in S_1$ since $S_1 = X$. If $a^* \in S_k$, then $x_k = a^*$, and in this case the algorithm returns the exact nearest neighbor. Otherwise, there exists some j for which $a^* \in S_j$ but $a^* \notin S_{j+1}$. We claim that in this case, x_j is a 3-approximate nearest neighbor to q.

If N_{j+1} is the inside child of N_j, then $d(q, x_j) \leq 3R_j/2$, yet $d(a^*, x_j) \geq 2R_j$, so by the triangle inequality,

$$d(q, a^*) \geq d(a^*, x_j) - d(q, x_j) \geq 2R_j - 3R_j/2 = R_j/2 \geq d(q, x_j)/3.$$

If N_{j+1} is the outside child of N_j, then $d(q, x_j) \geq 3R_j/2$, yet $d(a^*, x_j) \leq R_j$. By the triangle inequality $d(q, a^*) \geq R_j/2$, and we conclude that

$$d(x_j, q) \leq d(x_j, a^*) + d(a^*, q) \leq R_j + d(a^*, q) \leq 3\, d(a^*, q).$$

The proof follows by recalling that the algorithm outputs the closest point to q among $x_1, \ldots, x_k$.

3.2 Polynomial Storage

We now discuss how to achieve a space requirement that is polynomial in n, regardless of $\dim(X)$, by modifying the ring-separator tree algorithm of Section 3.1. In a nutshell, we introduce three techniques that, when applied together, "massage" this decision tree into a polynomial size directed acyclic graph (DAG) that can be used for $O(1)$-ANN. First, we "canonicalize" the decision tree by snapping every δ-ring-separator to a suitable net of the metric. This step limits the number of distinct radii that are used by the ring-separators in the data structure. Second, we eliminate the need for outside children in the decision tree by using several (actually $2^{O(\dim(X))} \log n$) inside children. This step opens the possibility to take a path in the tree (sequence of inside children) that corresponds to properly nested balls and represent the information in the entire sequence by a single ball (namely, the last one). This modification is crucial for the third step, in which we "fold" the decision tree onto itself, by merging nodes that have the same role (i.e., correspond to the same ball). A crucial detail in this step is a subtle invariant that is maintained when going into smaller and smaller balls (inside children). Specifically, whenever we go into a ball $B_X(y, 2R)$, we know that $d(y, q) \leq \gamma R$ for a suitable constant $1 < \gamma < 2$. This guarantees that every sequence of balls that we generate is indeed properly nested.

We sketch the resulting ring-separator DAG and algorithm, deferring details to the full version of the paper. Fix, for every R which is a power of 2, an R-net Y_R of X such that $Y_R \subseteq Y_{R/2}$. Each vertex of the DAG is a tuple $\langle y, R \rangle$, where $y \in Y_R$ and R is a power of 2, and represents the ball $S = B_X(y, 2R)$. (It is easy to bound the number of non-trivial values R per node y by $O(n)$.) If $S = B_X(y, 2R)$ contains only one point of X then the vertex has no outgoing edges. Otherwise, let (x, t) be a ring-separator for S. The first technique above guarantees that $x \in Y_{t/2}$. We now add an outgoing edge to every vertex $\langle y', R' \rangle$ where $R' \geq t$ is a power of 2 and $y' \in Y_{R'/32}$ such that $\frac{3}{4}R' \leq d(x, y') \leq 3R'$ and $d(y, y') \leq \frac{3}{2}R$. (The case $R' \leq t$ is slightly different.) Given a query point q, the $O(1)$-ANN algorithm traverses the DAG, starting from a fixed vertex corresponding to a ball that contains all of X, and ending when a sink is reached. When the algorithm is at the vertex $\langle y, R \rangle$ (assuming it is not a sink), we compute the value R' which is

a power of 2 and $d(x,q) \in (R', 2R']$. (The case $d(x,q) \leq 2t$ is slightly different.) If there is an edge to a vertex $\langle y', R'/32 \rangle$ with $d(y',q) \leq \frac{9}{8}R'/32$, the traversal proceeds along any such edge. Otherwise, the traversal stops. Eventually, the algorithm reports the closest point to the query among the points y and x seen along the traversal.

Acknowledgements. The authors would like to thank the Weizmann Institute, and in particular their host Uriel Feige.

References

[Ass83] P. Assouad. Plongements lipschitziens dans $\mathbf{R}^n$. *Bull. Soc. Math. France,* 111(4):429–448, 1983.

[Cla99] K. L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete Comput. Geom.,* 22(1):63–93, 1999.

[GKL03] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Proceedings of the 44th annual Symposium on the Foundations of Computer Science,* 2003.

[Gro99] M. Gromov. *Metric structures for Riemannian and non-Riemannian spaces.* Birkhäuser, Boston, 1999.

[HKMR04] K. Hildrum, J. Kubiatowicz, S. Ma, and S. Rao. A note on finding nearest neighbors in growth-restricted metrics. In *Proceedings of the 15th annual ACM-SIAM Symposium on Discrete Algorithms,* 2004.

[H01] S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV, 2001),* pages 94–103. IEEE Computer Soc., Los Alamitos, CA, 2001.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *30th Annual ACM Symposium on Theory of Computing,* pages 604–613, May 1998.

[KKL03] S. Kakade, M. Kearns, and J. Langford. Exploration in metric state spaces. In *Proc. of the 20th International Conference on Machine Learning,* 2003.

[KL04] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the 15th annual ACM-SIAM Symposium on Discrete Algorithms,* 2004.

[KOR98] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *30th Annual ACM Symposium on the Theory of Computing,* pages 614–623, 1998.

[KR02] D. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *34th Annual ACM Symposium on the Theory of Computing,* pages 63–66, 2002.

[Tal04] K. Talwar. Bypassing the embedding: Approximation schemes and distance labeling schemes for growth restricted metrics. To appear in the procedings of the *36th annual Symposium on the Theory of Computing,* 2004.

Regular Solutions of Language Inequalities and Well Quasi-orders

Michal Kunc*

Department of Mathematics, Masaryk University,
Janáčkovo nám. 2a, 662 95 Brno, Czech Republic,
kunc@math.muni.cz,http://www.math.muni.cz/~kunc

Abstract. By means of constructing suitable well quasi-orders of free monoids we prove that all maximal solutions of certain systems of language inequalities are regular. This way we deal with a wide class of systems of inequalities where all constants are languages recognized by finite simple semigroups. In a similar manner we also demonstrate that the largest solution of the inequality $XK \subseteq LX$ is regular provided the language L is regular.

1 Introduction

Systems of language equations and inequalities were intensively studied especially in connection with context-free languages since these languages can be elegantly described as components of least solutions of systems of explicit polynomial equations. Much less attention was devoted to implicit language equations and to equations employing other operations than union and concatenation. Only little research has been done also on maximal solutions of language equations. Such issues were first addressed by Conway [3], who observed that inequalities of the form $E \subseteq L$, where E is a regular function of variables and L is a regular language, possess only finitely many maximal solutions, all of them are regular and computable. More precisely, every component of a maximal solution of such an equation is a union of certain classes of the syntactic congruence of L. In particular, this leads to an algorithm for calculating best approximations of a given regular language by other given languages.

In his book Conway also formulated several conjectures concerning for instance maximal solutions of commutation equations $XL = LX$ and so-called semi-linear inequalities. Problems of commutation of languages were revisited in the past few years in a series of articles (e.g. [2,7]), where it was proved that in certain special cases the largest language commuting with a given regular language is again regular (see [8] for a survey and simplified proofs). On the other hand, recently the author demonstrated that the largest language commuting with a given finite language even need not be recursively enumerable [11].

Regular solutions of systems of inequalities generalizing regular grammars were studied for example by Leiss [13]. Baader and Küsters [1] used largest solutions of systems of linear equations, i.e. equations of the form

* Supported by the grant 201/01/0323 of the Grant Agency of the Czech Republic.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 870–881, 2004.

$$K_0 + K_1X_1 + \cdots + K_nX_n = L_0 + L_1X_1 + \cdots + L_nX_n \ ,$$

where $K_0, \ldots, K_n, L_0, \ldots, L_n$ are regular languages, for dealing with unification of concept descriptions; they proved that the largest solution of each such system is regular and its computation is an ExpTime-complete problem. Maximal solutions were also considered in the case of standard systems of equations defining context-free languages and related classes [15]. An attempt to initiate development of a unified theory of general language equations has been made by Okhotin [14]; in particular, he describes classes of languages definable as unique, smallest and largest solutions of systems of language inequalities using all Boolean operations.

In this paper we introduce a new method of demonstrating regularity of maximal solutions of language inequalities based on the concept of well quasi-orders of free monoids. Well quasi-orders already proved to be a very useful tool in many areas of mathematics and computer science [10]. In the theory of formal languages well quasi-orders are frequently applied to obtain regularity conditions; the most important result of this kind is a generalization of Myhill-Nerode theorem due to Ehrenfeucht et al. [4] stating that a language is regular if and only if it is upward closed with respect to a monotone well quasi-order.

This article deals with two different classes of inequalities. First we consider systems of inequalities of a very general form (involving even infinitary union and intersection operations) and show that regularity of maximal solutions of such systems is guaranteed when only constant languages recognized by simple semigroups are allowed (this in particular covers the case of group languages). In the second part of the paper we show that the largest solution of the inequality $XK \subseteq LX$ is regular provided the language L is regular. This contrasts with the fact that the largest solution of the equation $XL = LX$, where L is a regular language, is not always recursively enumerable. In both situations studied in this paper the result is achieved by constructing a suitable well quasi-order of the free monoid and demonstrating that every solution of our system is in fact contained in some solution upward closed with respect to this quasi-order. This extended abstract contains complete descriptions of these quasi-orders, but most of the proofs are omitted due to space constraints; they can be found in the full version of the paper [12].

Basic notions employed in our considerations are recalled in the following section. For a more comprehensive introduction to formal languages and to semigroup theory the reader is referred to [16] and [6], respectively.

2 Preliminaries

We denote the sets of positive and non-negative integers by $\mathbb{N}$ and $\mathbb{N}_0$, respectively. Throughout the paper we consider a finite alphabet A and an infinite set of variables $\mathcal{X}$. As usual, we write A^+ for the set of all non-empty finite words over A, and A^* for the set obtained from A^+ by adding the empty word ε. We use the same symbols A^+ and A^* to denote the free semigroup and the free monoid, respectively, which arise from these sets when we equip them with the

operation of concatenation. Languages over A are arbitrary subsets of A^* and a language $L \subseteq A^*$ is called ε-free if $\varepsilon \notin L$.

Let E be an arbitrary expression built from languages over A (called constants) and variables from $\mathcal{X}$ using some symbols for language operations and let $\alpha : \mathcal{X} \to 2^{A^*}$ be a mapping assigning to each variable a language over A. Then $\alpha(E)$ denotes the language obtained by replacing each occurrence of every variable $X \in \mathcal{X}$ in E with the language $\alpha(X)$ and evaluating the resulting expression. A language inequality is a formal inequality $E \subseteq F$ of two expressions over constant languages and variables. A solution of $E \subseteq F$ is any mapping $\alpha : \mathcal{X} \to 2^{A^*}$ satisfying $\alpha(E) \subseteq \alpha(F)$. We call a solution α regular if all the languages $\alpha(X)$ are regular. Solutions of a given system of language inequalities are partially ordered by componentwise inclusion

$$\alpha \leq \beta \iff \forall X \in \mathcal{X} : \alpha(X) \subseteq \beta(X)$$

and we are mainly interested in solutions maximal with respect to this ordering.

Let $\mathfrak{S} = (S, *)$ be a semigroup and let $\sigma : A^+ \to \mathfrak{S}$ be a semigroup homomorphism. We say that a language $L \subseteq A^+$ is *recognized* by the homomorphism σ if $\sigma^{-1}\sigma(L) = L$. The *syntactic congruence* $\equiv_L$ of a language $L \subseteq A^+$ is the congruence of the free semigroup A^+ defined by the condition

$$u \equiv_L v \iff (\forall x, y \in A^*)(xuy \in L \iff xvy \in L) .$$

In other words, the relation $\equiv_L$ is the largest congruence of A^+ such that the corresponding projection homomorphism recognizes L. The factor semigroup $A^+/\equiv_L$ is called the *syntactic semigroup* of L and denoted $\mathcal{S}(L)$; the projection homomorphism $\sigma_L : A^+ \to \mathcal{S}(L)$ is referred to as the *syntactic homomorphism* of L. It is well-known that a language is regular if and only if its syntactic semigroup is finite.

A commutative and idempotent semigroup is called a *semilattice*. A *null semigroup* is a semigroup $\mathfrak{S} = (S, *)$ containing a zero element 0 such that $s * t = 0$ for every $s, t \in S$. An *ideal* of a semigroup $\mathfrak{S} = (S, *)$ is a non-empty subset $I \subseteq S$ such that for all $s \in I$ and $t \in S$ we have $s * t \in I$ and $t * s \in I$. A semigroup is called *simple* if it has no proper ideal. A complete classification of finite simple semigroups is known, namely every finite simple semigroup is isomorphic to a so-called Rees matrix semigroup over some group (see e.g. [6]).

If a semigroup $\mathfrak{S} = (S, *)$ possesses a congruence relation $\equiv$ such that the factor-semigroup $\mathfrak{S}/\equiv$ is a chain (i.e. for all $s, t \in S$ either $s * t \equiv t * s \equiv s$ or $s * t \equiv t * s \equiv t$), then every congruence class $(s \equiv)$, for $s \in S$, is a subsemigroup of $\mathfrak{S}$ and the semigroup $\mathfrak{S}$ is called a *chain of semigroups* $(s \equiv)$.

We conclude this section by recalling the definition of well quasi-orders and Higman's Lemma—one of the fundamental results of the theory of well quasi-orders, which is often applied in connection with formal languages.

A *quasi-order* $\leq$ on a set S is a reflexive and transitive relation. We say that a subset T of S is *upward closed* with respect to $\leq$ if for every $t \in T$ and $s \in S$, $t \leq s$ implies $s \in T$. A quasi-order $\leq$ on S is called a *well quasi-order* if the following equivalent conditions are satisfied:

(i) There exists neither an infinite strictly descending sequence in S nor an infinite sequence of mutually incomparable elements of S.
(ii) If $(s_n)_{n\in\mathbb{N}}$ is an infinite sequence of elements of S, then there exist $i, j \in \mathbb{N}$ such that $i < j$ and $s_i \leq s_j$.
(iii) Every infinite sequence of elements of S has an infinite ascending subsequence.
(iv) For every subset $T \subseteq S$ there exists a finite subset U of T such that for each $t \in T$ there exists some element $s \in U$ satisfying $s \leq t$.
(v) There does not exist an infinite sequence of upward closed subsets of S strictly ascending with respect to inclusion.

The following useful fact can be easily verified using condition (ii).

Lemma 1. *Let $(S, \leq)$ and $(T, \sqsubseteq)$ be quasi-ordered sets and let $\varphi : S \to T$ be an arbitrary mapping satisfying the condition*

$$(\forall s_1, s_2 \in S)(\varphi(s_1) \sqsubseteq \varphi(s_2) \implies s_1 \leq s_2)\ . \tag{1}$$

If the relation $\sqsubseteq$ is a well quasi-order on T, then $\leq$ is a well quasi-order on S.

For an arbitrary set A, we denote by $\preceq$ the subword partial order on A^*, which is defined, for $a_1, \ldots, a_n, b_1, \ldots, b_m \in A$, by setting $a_1 \cdots a_n \preceq b_1 \cdots b_m$ if and only if there exist $1 \leq i_1 < i_2 < \cdots < i_n \leq m$ such that $b_{i_1} \cdots b_{i_n} = a_1 \cdots a_n$.

Proposition 1 (Higman [5]). *For every finite set A, the relation $\preceq$ is a well partial order on A^*.*

3 Decomposition Quasi-orders

Let us start this section by describing systems of inequalities which will be considered here. Let $\mathfrak{L}$ be a finite set of ε-free languages over A. We say that an inequality $E \subseteq F$ is an *$\mathfrak{L}$-inequality* if the expression E is a product of variables and arbitrary constants and the expression F is built from variables and languages belonging to $\mathfrak{L} \cup \{\{\varepsilon\}\}$ using symbols for the operations of concatenation, arbitrary (possibly infinite) union and arbitrary (possibly infinite) intersection.

Let $\sigma : A^+ \to \mathfrak{S}$ be an arbitrary homomorphism onto a finite semigroup $\mathfrak{S}$. We define a quasi-order $\leq_\sigma$ on A^* by setting $v \leq_\sigma u$ if and only if $v = a_1 \cdots a_n$, where $a_1, \ldots, a_n \in A$, and $u = u_1 \cdots u_n$, where $u_j \in A^+$ and $\sigma(u_j) = \sigma(a_j)$ for $j = 1, \ldots, n$. This quasi-order is monotone, i.e. from $v_1 \leq_\sigma u_1$ and $v_2 \leq_\sigma u_2$ it follows that $v_1 v_2 \leq_\sigma u_1 u_2$. Notice that if $v \leq_\sigma u$ then either $u = v = \varepsilon$ or $u, v \in A^+$ and $\sigma(v) = \sigma(u)$.

The following result states that all maximal solutions of arbitrary systems of $\mathfrak{L}$-inequalities are regular provided there exists a homomorphism σ recognizing all languages from $\mathfrak{L}$ for which the relation $\leq_\sigma$ is a well quasi-order.

Theorem 1. *Let $\mathfrak{L}$ be a finite set of ε-free languages over A and let $\sigma : A^+ \to \mathfrak{S}$ be a homomorphism onto a finite semigroup $\mathfrak{S}$ recognizing all languages in $\mathfrak{L}$ such*

that A^* is well quasi-ordered by $\leq_\sigma$. Let I be an arbitrary (possibly infinite) set and let $\Sigma = \{E_i \subseteq F_i \mid i \in I\}$ be a system of $\mathfrak{L}$-inequalities. Then every solution of Σ is contained in a regular solution of Σ; in particular, every maximal solution of the system Σ is regular. If only finitely many variables occur in Σ, then every solution of Σ is contained in a maximal one. The same conclusions hold true if only ε-free solutions of Σ are considered.

Proof. Let α be a solution of Σ. For every $X \in \mathcal{X}$ define the language

$$\beta(X) = \{u \in A^* \mid \exists v \in \alpha(X) : v \leq_\sigma u\} .$$

It is clear that $\alpha(X) \subseteq \beta(X)$ and that $\varepsilon \in \beta(X)$ if and only if $\varepsilon \in \alpha(X)$ since the empty word is incomparable with the other elements of A^*. We are going to show that β is a regular solution of Σ.

First observe that because the quasi-order $\leq_\sigma$ is monotone, if a word u belongs to $\beta(E_i)$, there exists $v \in \alpha(E_i)$ such that $v \leq_\sigma u$. We prove by induction with respect to the structure of the expression F_i that if $v \in \alpha(F_i)$ and $v \leq_\sigma u$ for some words u and v, then $u \in \beta(F_i)$, which is enough to conclude that β is a solution of Σ. So assume a word v belongs to $\alpha(e)$ for some subexpression e of F_i and $v \leq_\sigma u$. If e is a variable, we have $u \in \beta(e)$ by the definition of β. In the case e is a language from $\mathfrak{L}$, one obtains $u \in \beta(e)$ from the fact $\sigma(u) = \sigma(v)$. For $e = \{\varepsilon\}$, the only possibility is $u = v = \varepsilon \in \beta(e)$. If the expression e is of the form $\bigcup_{k\in K} e_k$ or $\bigcap_{k\in K} e_k$ for some set K, then $u \in \beta(e)$ is clear from the induction hypothesis. Finally, consider $e = e_1 \cdot e_2$. Then $v = v_1 \cdot v_2$, where $v_1 \in \alpha(e_1)$ and $v_2 \in \alpha(e_2)$. From $v \leq_\sigma u$ we deduce $v_1 = a_1 \cdots a_m$ and $v_2 = a_{m+1} \cdots a_n$, where $0 \leq m \leq n$ and $a_j \in A$, and $u = u_1 \cdots u_n$ for some words $u_1, \ldots, u_n \in A^+$ satisfying $\sigma(u_j) = \sigma(a_j)$ for $j = 1, \ldots, n$. Therefore $v_1 \leq_\sigma u_1 \cdots u_m$ and $v_2 \leq_\sigma u_{m+1} \cdots u_n$ and we can apply the induction hypothesis to these words. Hence $u \in \beta(e)$.

In order to prove that $\beta(X)$ is a regular language, observe that $\beta(X)$ is upward closed with respect to the well quasi-order $\leq_\sigma$, therefore it can be generated by finitely many elements of A^*, i.e. $\beta(X)$ is a union of finitely many languages of the form $\langle v \rangle = \{u \in A^* \mid v \leq_\sigma u\}$ for a word $v \in A^*$. And it is easy to see that for arbitrary letters $a_1, \ldots, a_n \in A$ we have

$$\langle a_1 \cdots a_n \rangle = (\sigma^{-1}\sigma(a_1)) \cdots (\sigma^{-1}\sigma(a_n)) ,$$

which shows that each language $\langle v \rangle$ is regular.

We have already proved that every solution of Σ is contained in a regular solution whose every component is a language upward closed with respect to the well quasi-order $\leq_\sigma$. Because there is no infinite strictly ascending sequence of such upward closed sets, this immediately implies that if there are only finitely many variables, every solution is in fact contained in a maximal solution. □

Remark 1. Observe that existence of a maximal solution above every solution follows immediately from Zorn's Lemma (even if there are infinitely many variables) since all operations in our inequalities are monotone and left-hand sides

employ only finitary operations. In contrast, our proof of this fact in the case of finitely many variables avoids the Axiom of Choice, although even for regular solutions of simple inequalities it does not provide us with an algorithm for computing such a maximal solution.

Further notice that the relation $\leq_\sigma$ in the proof is a monotone well quasi-order on A^* and therefore the languages $\beta(X)$ are regular due to the result of Ehrenfeucht et al. [4]; we give a direct proof of their regularity because it also provides us with some information on how maximal solutions are related to constant languages occurring in the system.

Now we prove that when the semigroup used to recognize languages from the set $\mathfrak{L}$ is a group, the relation $\leq_\sigma$ is a well quasi-order on A^* and therefore Theorem 1 can be applied.

Lemma 2. *Let $\sigma : A^+ \to \mathfrak{G}$ be a homomorphism onto a finite group $\mathfrak{G} = (G, *)$. Then $\leq_\sigma$ is a well quasi-order on A^*.*

Proof. Clearly, it is sufficient to deal with the restriction of $\leq_\sigma$ to A^+. We consider the free monoid over the alphabet $\{0,1\} \times G$ ordered by the subword partial order $\preceq$ and define a mapping $\varphi : A^+ \to (\{0,1\} \times G)^*$ by the formula:

$$\varphi(a_1 \cdots a_n) = (0, \sigma(a_1))(0, \sigma(a_1 a_2)) \cdots (0, \sigma(a_1 \cdots a_{n-1}))(1, \sigma(a_1 \cdots a_n)) ,$$

for every $a_1, \ldots, a_n \in A$. We are going to prove that this mapping satisfies (1). Let us consider arbitrary non-empty words $v = a_1 \cdots a_n$ and $u = b_1 \cdots b_m$ such that $\varphi(v) \preceq \varphi(u)$, where $a_1, \ldots, a_n, b_1, \ldots, b_m \in A$. We have to show that $v \leq_\sigma u$ holds. Let $1 \leq i_1 < i_2 < \cdots < i_n \leq m$ be indices determining the word $\varphi(v)$ as a subsequence of $\varphi(u)$. Then in particular $i_n = m$ due to the number 1 on the first component of the last pair in $\varphi(v)$. Consequently we can define a decomposition $u = u_1 \cdots u_n$ by the rule $u_j = b_{i_{j-1}+1} \cdots b_{i_j}$ for $j = 1, \ldots, n$, where $i_0 = 0$, and verify the required property of the decomposition using the fact that $\mathfrak{G}$ is a group:

$$\begin{aligned} \sigma(u_j) &= (\sigma(b_1 \cdots b_{i_{j-1}}))^{-1} * \sigma(b_1 \cdots b_{i_j}) \\ &= (\sigma(a_1 \cdots a_{j-1}))^{-1} * \sigma(a_1 \cdots a_j) = \sigma(a_j) . \end{aligned}$$

Altogether, we have proved that (1) is valid for φ and because $\preceq$ is a well partial order on $(\{0,1\} \times G)^*$ due to Proposition 1, the relation $\leq_\sigma$ is a well quasi-order on A^+ by Lemma 1. □

Using a more involved direct proof we can precisely characterize those semigroups $\mathfrak{S}$ which satisfy that $\leq_\sigma$ is a well quasi-order on A^* for every homomorphism $\sigma : A^+ \to \mathfrak{S}$.

Theorem 2. *Let $\mathfrak{S}$ be a finite semigroup. Then the relation $\leq_\sigma$ is a well quasi-order on A^* for every alphabet A and every homomorphism $\sigma : A^+ \to \mathfrak{S}$ if and only if $\mathfrak{S}$ is a chain of simple semigroups.*

When we are concerned with regular languages, the most interesting inequalities are those built using regular operations. Systems of such inequalities are in fact a special case of systems considered in this section since the star operation is constructed from the operations of concatenation and infinite union and as our systems are allowed to be infinite, we can actually use the operation of infinite union (and consequently the star operation) also on left-hand sides of inequalities. Because all languages from a given set $\mathfrak{L}$ can be recognized by the product of semigroups recognizing individual languages and any product of simple semigroups is again simple, the following result is an immediate consequence of Theorems 1 and 2.

Corollary 1. *Let Σ be a finite system of inequalities of the form $E_i \subseteq F_i$, where E_i and F_i are regular expressions over variables, the language $\{\varepsilon\}$ and regular languages recognized by simple semigroups. Then every solution of Σ is contained in a maximal solution and every maximal solution of Σ is regular.*

Remark 2. In the system Σ of Corollary 1 one can prescribe whether a given variable X contains the empty word or not since the inequalities $\{\varepsilon\} \subseteq X$ and $X \subseteq A^+$ are of the required form.

If only one constant language occurs in a system of inequalities, then in order to apply Theorems 1 and 2 it is sufficient to know that the language is recognized by a chain of finite simple semigroups. Notice that unlike for languages recognized by groups or simple semigroups, recognizability of a regular language by a chain of simple semigroups is independent of the underlying alphabet since additional letters not employed by the language form a zero element in the syntactic semigroup, which becomes the least element of the chain.

Corollary 2. *Let $L \subseteq A^+$ be a regular language recognized by a chain of simple semigroups. Let Σ be a finite system of inequalities of the form $E_i \subseteq F_i$, where E_i and F_i are regular expressions over variables and the languages L and $\{\varepsilon\}$. Then every solution of Σ is contained in a maximal solution and every maximal solution of Σ is regular.*

Before we proceed to demonstrate results of this section on examples, let us describe a characterization of regular languages recognized by simple semigroups and chains of simple semigroups by means of minimal automata. Recall that languages recognizable by groups are precisely those languages whose minimal automaton is codeterministic, i.e. contains no distinct states p and q such that $\delta(p, a) = \delta(q, a)$ for some $a \in A$. This condition can be transformed into a condition corresponding to the case of simple semigroups by considering codeterminism for two-letter words instead of single letters.

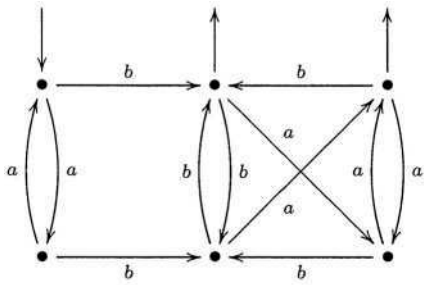
Lemma 3. *A regular ε-free language L over A can be recognized by a simple semigroup if and only if its minimal automaton contains no states p and q which for some letters $a, b \in A$ satisfy $\delta(p, a) \neq \delta(q, a)$ and $\delta(p, ab) = \delta(q, ab)$.*

A similar condition can be formulated also for chains of simple semigroups:

Lemma 4. *A regular ε-free language L over A can be recognized by a chain of simple semigroups if and only if there exists a total ordering $\leq$ of A such that the minimal automaton of L does not contain states p and q for which there exist a word $u \in A^*$ and letters $a, b \in A$ satisfying $a \leq b$, $\delta(p, au) \neq \delta(q, au)$ and $\delta(p, aub) = \delta(q, aub)$.*

Now we illustrate Theorem 1 on a concrete non-trivial inequality with one constant language recognized by a chain of simple semigroups.

Example 1. Let L be the ε-free language over the alphabet $A = \{a, b\}$ whose minimal automaton is the following:



The language L consists exactly of those words $u \in L^+$ which contain some occurrence of b and where the difference between the length of u and the number of blocks of occurrences of b in u is even. The syntactic semigroup of L is defined by the relations $a^3 = a$, $b^3 = b$, $ab^2 = a^2b$, $ba^2 = b^2a$ and $bab = b^2$; it is a chain of two simple semigroups whose elements are represented by the words a, a^2 and b, b^2, ab, ab^2, ba, b^2a, aba, ab^2a, respectively.

Let us consider the inequality $aXaXa \subseteq LXL$ with one variable X. It is easy to verify that this inequality possesses a largest solution, namely the language $(a^2)^*ab^2a(a^2)^* \cup A^*bA^+bA^*$. In the proof of Theorem 1 we have seen that this solution is upward closed with respect to the well partial order $\leq_{\sigma_L}$. In fact, there are precisely 87 minimal elements in this solution with respect to $\leq_{\sigma_L}$:

$$\{ab^2a\} \cup \big(\{\varepsilon, a, a^2\} \cdot (\{b, b^2\}\{a, a^2\}\{b, b^2\} \cup \{b^3, b^4, babab\}) \cdot \{\varepsilon, a, a^2\}\big) \setminus$$
$$\setminus (\{\varepsilon, a, a^2\}b^2a^2b^2\{\varepsilon, a, a^2\} \cup \{ab^4a, ab^2ab^2a, aba^2b^2a, ab^2a^2ba\}) .$$

Let us now give a few simple examples showing that if languages in the set $\mathfrak{L}$ cannot be recognized by a chain of simple semigroups, then the conclusion of Theorem 1 often does not hold. In our examples we deal with the simplest semigroups which are not of this form, namely with null semigroups and semilattices. First we look at what happens in the presence of infinite unions.

Example 2. Let $a \in A$ and let $\mathfrak{L}$ contain only the language $\{a\}$, whose syntactic semigroup is a two-element null semigroup. Then for any non-regular set $N \subseteq \mathbb{N}$, the largest solution of the inequality $X \subseteq \bigcup_{n \in N} a^n$ is not regular.

A similar situation arises for $\mathfrak{L} = \{a^+, b^+\}$, where $a, b \in A$. Both languages a^+ and b^+ are recognized by a homomorphism to a three-element semilattice with a zero element and two incomparable elements corresponding to letters a and b. In this case, the largest solution of the inequality $X \subseteq \bigcup_{n \in N} (a^+b^+)^n$ is not regular provided $N \subseteq \mathbb{N}$ is a non-regular set of positive integers.

The following examples demonstrate that even if no infinitary operations are allowed to occur in our inequalities, the restriction to chains of simple semigroups is essential.

Example 3. Let $\mathfrak{L} = \{\{a\}, \{b\}\}$, where $a, b \in A$. To recognize these languages we need a three-element null semigroup and the largest solution of the inequality $X \subseteq aXa \cup \{b\}$ is a non-regular language $\{a^n b a^n \mid n \in \mathbb{N}_0\}$.

Analogously, for the set $\mathfrak{L} = \{a^+, b^+, c^+\}$, where $a, b, c \in A$, the inequality $X \subseteq a^+b^+Xa^+b^+ \cup c^+$ has the largest solution non-regular, namely equal to $\bigcup_{n \in \mathbb{N}_0} (a^+b^+)^n c^+ (a^+b^+)^n$. And to recognize the languages of $\mathfrak{L}$, one can use a four-element semilattice with a zero element and three incomparable elements.

4 Semi-commutation

Let K and L be languages over the alphabet A and consider the inequality $XK \subseteq LX$. It is easy to see that the union of arbitrarily many solutions of this inequality is again its solution. In particular, this means that this inequality possesses the largest solution, namely the union of all solutions. In this section we explain why the largest solution of the inequality $XK \subseteq LX$ is always regular provided L is a regular language. With this aim we introduce another well quasi-order on A^*. But this time we have to consider more involved structures than just plain sequences as we did in Section 3.

The basic idea of the proof is to think of the inequality $XK \subseteq LX$ as a game of two players, the attacker and the defender. The language K determines possible actions of the attacker and the language L determines possible actions of the defender. A position of the game is an arbitrary word w from A^*. At each step of the game, both players successively modify the word according to the following rules. When the game is in a position w, the attacker chooses any element v of K and appends it to w. If no word from L is a prefix of wv, the attacker wins. Otherwise the defender removes any word belonging to L from the beginning of wv. The resulting word is a new position of the game. The defender wins the game if and only if he manages to continue playing forever.

Observe that if the defender has a winning strategy for a given initial position $w \in A^*$, then the set of all positions reachable from w in some scenario corresponding to a chosen winning strategy forms a solution of the inequality $XK \subseteq LX$ containing w. Conversely, given any solution $M \subseteq A^*$ of $XK \subseteq LX$, one can easily construct winning strategies of the defender for all elements of M. Therefore the largest solution of the inequality $XK \subseteq LX$ is exactly the set of all positions of the game where the defender has a winning strategy. The main result of this section can then be reformulated as follows: If the set of possible

actions of the defender is regular, then the set of all winning positions of the defender is regular no matter what actions are available to the attacker.

Given an initial position $w \in A^*$, we consider the actions of the defender which can be performed without removing any letters previously added by the attacker. In other words, we deal with all sequences $(w_1, \ldots, w_n)$ of elements of L whose concatenation $w_1 \cdots w_n$ is a prefix of w. We arrange these sequences into the form of a tree expressing the order of actions, i.e. the node $(w_1, \ldots, w_n)$ will be a successor of the node $(w_1, \ldots, w_{n-1})$. In addition, we have to consider for each sequence $(w_1, \ldots, w_n)$ the suffix u of w satisfying $w = w_1 \cdots w_n u$. This word u can be removed by the defender in the following turn together with several letters previously added by the attacker. The only information the defender needs to know is which words can be appended to u in order to get a word from L. This is uniquely determined by the $\equiv_L$-class of u, and therefore it is sufficient to label the node $(w_1, \ldots, w_n)$ by the element $\sigma_L(u) \in \mathcal{S}(L)$. Actually, even less information about the word w is needed: we capture properties of the game more accurately by only indicating for each node which elements of $\mathcal{S}(L)$ occur as labels of its successors, i.e. we assign to each node a set of elements of $\mathcal{S}(L)$.

In this way, we construct a labelled tree for every $w \in A^*$. Then we introduce a well quasi-order on the set of such trees expressing possibility of using winning strategies for one initial position also for another one and prove that the largest solution of the inequality $XK \subseteq LX$ is upward closed with respect to the quasi-order induced on A^*.

Let us now describe the construction in detail. Let $L \subseteq A^+$ be a regular language and let $\sigma_L : A^+ \to \mathcal{S}(L)$ be its syntactic homomorphism. We consider the monoid $\mathcal{S}(L)^1$ obtained from $\mathcal{S}(L)$ by adding a new neutral element 1 and extend the syntactic homomorphism to $\sigma_L : A^* \to \mathcal{S}(L)^1$ by defining $\sigma_L(\varepsilon) = 1$. By an *L-tree* we mean a quadruple $\tau = (N_\tau, r_\tau, \pi_\tau, \ell_\tau)$, where

- N_τ is a finite set of nodes of τ,
- $r_\tau \in N_\tau$ is a distinguished node called the root of τ,
- the mapping $\pi_\tau : N_\tau \setminus \{r_\tau\} \to N_\tau$ maps each node to its predecessor,
- for every $\nu \in N_\tau$ there exists $n \in \mathbb{N}_0$ such that $\pi_\tau^n(\nu) = r_\tau$,
- the mapping $\ell_\tau : N_\tau \to 2^{\mathcal{S}(L)^1}$ is a labelling of nodes with sets of elements of $\mathcal{S}(L)^1$ satisfying $\ell_\tau(\nu) \subseteq \ell_\tau(\pi_\tau(\nu))$ for all $\nu \in N_\tau \setminus \{r_\tau\}$.

We denote by $\mathcal{T}(L)$ the set of all L-trees.

Now we define a quasi-order $\sqsubseteq$ on $\mathcal{T}(L)$. For $\tau, \vartheta \in \mathcal{T}(L)$ we set $\tau \sqsubseteq \vartheta$ if and only if there exists a mapping $H : N_\tau \to N_\vartheta$ which satisfies:

$$\forall \nu \in N_\tau : \ell_\tau(\nu) \subseteq \ell_\vartheta(H(\nu)) \,,$$

$$\forall \nu \in N_\tau \setminus \{r_\tau\} \; \exists k \in \mathbb{N} : H(\pi_\tau(\nu)) = \pi_\vartheta^k(H(\nu)) \,.$$

The relation $\sqsubseteq$ is in fact a well quasi-order due to Kruskal's Tree Theorem [9].

In order to define a quasi-order on A^*, we construct a mapping φ from A^* to $\mathcal{T}(L)$ as follows. For $w \in A^*$ let $N_{\varphi(w)}$ be the set of all finite sequences $(w_1, \ldots, w_n)$, where $n \in \mathbb{N}_0$ and $w_1, \ldots, w_n \in L$, such that the word $w_1 \cdots w_n$ is a prefix of w. The root $r_{\varphi(w)}$ of $\varphi(w)$ is the empty sequence and the predecessor

mapping is given by the rule $\pi_{\varphi(w)}(w_1,\ldots,w_n) = (w_1,\ldots,w_{n-1})$. Finally, we put an element $s \in \mathcal{S}(L)^1$ into the set $\ell_{\varphi(w)}(w_1,\ldots,w_n)$ if and only if there exist words $\overline{w} \in L^*$ and $\widetilde{w} \in A^*$ such that $w_1 \cdots w_n \overline{w} \widetilde{w} = w$ and $\sigma_L(\widetilde{w}) = s$.

Now for every $v, w \in A^*$ we set $v \leq_L w$ if and only if $\varphi(v) \sqsubseteq \varphi(w)$. Because $\sqsubseteq$ is a well quasi-order, by Lemma 1 this rule defines a well quasi-order on A^*. Then for any solution $M \subseteq A^*$ of the inequality $XK \subseteq LX$ it can be proved that the language $\{w \in A^* \mid \exists v \in M : v \leq_L w\}$ is a regular solution of $XK \subseteq LX$ containing M. This immediately gives the desired result:

Theorem 3. *If $K \subseteq A^*$ is an arbitrary language and $L \subseteq A^*$ is a regular language, then the largest solution and the largest ε-free solution of the inequality $XK \subseteq LX$ are regular.*

The following example in particular shows that it is essential to consider the whole tree structure associated with each word, not only the corresponding elements of the syntactic semigroup and the lengths of paths in the tree.

Example 4. Assume $A = \{a,b,c,d,e,f,g,h,i\}$ and let $K = \{e,i\}$ and

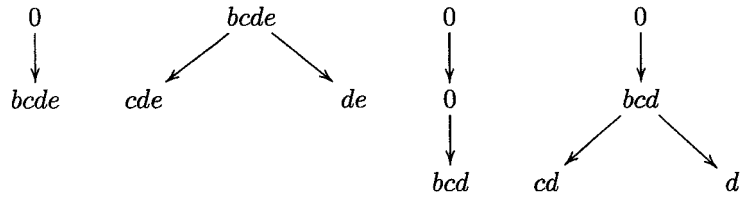
$$L = \{a,b,bc,f,fg,gh,hc,e,i\} \cup cdKiK \cup dKeK \cup g^*\{b,h\}cdK^3 .$$

The largest solution of the inequality $XK \subseteq LX$ is the language

$$L^* \cup L^*cdKi \cup L^*dKe \cup L^*g^*\{b,h\}cdK^2 \cup \\ \cup L^+g^*\{b,h\}cdK \cup bcdK \cup LL^+g^*\{b,h\}cd \cup Lbcd .$$

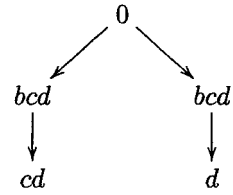
In order to calculate this solution, observe that if u belongs to a solution M of the inequality, then for every $n \in \mathbb{N}$ we have $ue^n \in MK^n \subseteq L^nM$. Because $\varepsilon \notin L$, if we take n sufficiently large, we deduce that the word ue^3 is a prefix of a word from L^*. Since this is possible only if $ue^3 \in L^*$, there remain only few cases to deal with.

We know that the largest solution is upward closed with respect to the well quasi-order $\leq_L$, therefore one can find finitely many elements of $\mathcal{T}(L)$ characterizing the solution, i.e. minimal elements of the image of the solution under the mapping φ. In our case, there are four one-node trees corresponding to words ε, cdi^2, de^2, $gbcde^2$ and four trees with more than one node corresponding to words $agbcde$, $bcde$, a^2gbcd and $abcd$, respectively:



(in the trees depicted here, each word u stands for the corresponding label $\sigma_L(u) \in \mathcal{S}(L)$ and by the symbol 0 we mean the zero element of $\mathcal{S}(L)$).

Finally, let us point out that the word *fghcd* does not belong to the solution even though the tree corresponding to it is very similar to the one of *abcd:*



This is a consequence of the facts $abcd \equiv_L fghcd$ and $bcd \equiv_L hcd \equiv_L ghcd$. Moreover, the same equivalences hold and therefore labels of these two trees are equal even if we consider elements of the syntactic semigroup of L^+ instead of L. In fact, when the inequality is viewed as a game, the difference between these two trees is that for the word *fghcd* the defender has to make his decision immediately after the first turn of the attacker whereas for the word *abcd* he can decide according to the opponent's second move.

Acknowledgement. I am very grateful to Ondřej Klíma for carefully reading the manuscript and for providing me with numerous helpful suggestions.

References

1. Baader, F., Küsters, R.: Unification in a description logic with transitive closure of roles. In *Proc. LPAR 2001,* LNCS 2250, Springer (2001) 217–232.
2. Choffrut, C., Karhumäki, J., Ollinger, N.: The commutation of finite sets: A challenging problem. *Theor. Comput. Sci.* **273** (2002) 69–79.
3. Conway, J.H.: *Regular Algebra and Finite Machines.* Chapman and Hall (1971).
4. Ehrenfeucht, A., Haussler, D., Rosenberg, G.: On regularity of context-free languages. *Theor. Comput. Sci.* **27** (1983) 311–332.
5. Higman, G.: Ordering by divisibility in abstract algebras. *Proc. Lond. Math. Soc.* **2** (1952) 326–336.
6. Howie, J.M.: *Fundamentals of Semigroup Theory.* Oxford University Press (1995).
7. Karhumäki, J., Petre, I.: Conway's problem for three-word sets. *Theor. Comput. Sci.* **289** (2002) 705–725.
8. Karhumäki, J., Petre, I.: Two problems on commutation of languages. In *Current Trends in Theoretical Computer Science,* World Scientific (to appear).
9. Kruskal, J.B.: Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Trans. Amer. Math. Soc.* **95** (1960) 210–225.
10. Kruskal, J.B.: The theory of well-quasi-ordering: a frequently discovered concept. *J. Comb. Theory, Ser. A,* **13** (1972) 297–305.
11. Kunc, M.: The power of commuting with finite sets of words. manuscript (2004), available at `http://www.math.muni.cz/~kunc`
12. Kunc, M.: Regular solutions of language inequalities and well quasi-orders. manuscript (2004), available at `http://www.math.muni.cz/~kunc`
13. Leiss, E.L.: *Language Equations.* Springer (1999).
14. Okhotin, A.: Decision problems for language equations. submitted (2003).
15. Okhotin, A.: Greatest solutions of language equations. submitted (2003).
16. Rozenberg, G., Salomaa, A. (eds.): *Handbook of Formal Languages.* Springer (1997).

A Calculus of Coroutines

J. Laird*

Dept. of Informatics, University of Sussex, UK
jiml@sussex.ac.uk

Abstract. We describe a simple but expressive calculus of sequential processes, which are represented as coroutines. This calculus can be used to express a variety of programming language features; we give simple macros for procedure calls, labelled jumps, integer references and stacks. We describe the operational properties of the calculus using reduction rules and equational axioms.

We describe a notion of categorical model for our calculus, and give a simple example of such a model based on a category of games and strategies. We prove full abstraction results showing that equivalence in the categorical model corresponds to observational equivalence in the calculus, and also to equivalence of *evaluation trees,* which are infinitary normal forms for the calculus.

We show that our categorical model can be used to interpret the untyped λ-calculus and use this fact to extract a sound translation of the λ-calculus into our calculus of coroutines.

1 Introduction

The object of this paper is to describe a simple calculus of control flow, based on *coroutines,* which can be used to give concise and precise descriptions of features such as state, non-local control, concurrent threads, and higher-order procedures[1]. In this sense, our work follows an established strand of research, studying higher-order sequential languages by translation into process calculi such as the π-calculus [5,3]. A crucial difference is that whereas in [3], sequentiality is isolated from a complicated universe of concurrent behaviour using a subtle notion of typing, in our calculus it is automatically a property of all terms. Thus, for example, we are able to give a new category-theoretic account of sequential control flow by describing a notion of categorical model for the calculus.

Underlying our attempts to describe and reason about sequential computational processes is *game semantics* [1,2,6], which has provided a precise and

* Supported by EU FET-GC 'MyThS: Models and Types for Security in Mobile Distributed Systems' IST-2001-32617.

[1] We will concentrate on sequential features here, although as noted in the conclusion, the use of coroutines to schedule multiple threads of control can be expressed by extending the calculus with non-determinstic choice.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 882–893, 2004.

wide-ranging account of such processes based upon representing them as strategies for a two-player game. A correspondence between coroutines and strategies is part of the folklore of the subject. In particular, although composition of strategies has been characterized as "parallel composition of processes plus hiding" [1], composition of *coroutines* might be considered to provide a more precise analogy, as strategies typically interact by passing *control* back and forth.

One objective of the work reported here is to formalise the correspondence between coroutines and strategies. We give a simple games model of our calculus (an instance of our class of categorical models), with a direct correspondence between terms in normal form (evaluation trees) and strategies. Thus our calculus provides a new way of representing (and hence reasoning about) strategies, potentially allowing new ways of investigating their algorithmic properties.

Although our games model is structurally very simple, we show that it has sufficient expressive power to model higher-order functions (and this corresponds to a translation of λ-terms into our calculus). This development has been inspired by the work of Longley, who has shown [7] that terms of the untyped λ-calculus can be interpreted as strategies on a simple "universal game" in which Opponent and Player alternately choose natural number indices. In essence, we give a formal construction of such a model.

2 A Calculus of Coroutines

We assume a set B of basic datatypes, writing $\mathbf{n}$ for a type with n distinct values — we shall generally take $B = \{\mathbf{0}, \mathbf{1}, \omega\}$ — i.e. an empty type, a type of commands, and a type of natural numbers. Terms are assigned *function types* of the form $\mathbf{m}^{\mathbf{n}}$, where $\mathbf{m}$ and $\mathbf{n}$ are in B. A term of type $\mathbf{m}^{\mathbf{n}}$ represents a program which receives a value in $\mathbf{n}$ as input on the left, performs some computation, and (if it terminates) produces a value in $\mathbf{m}$ as output. The typable terms are given in contexts[2] of variables, according to the judgements in Table 1. Variables (or *coroutine names*) come with a complementation operation $\overline{(\_)}$ such that $\overline{\overline{k}} = k$, and if $k : \mathbf{n}^{\mathbf{m}}$, then $\overline{k} : \mathbf{m}^{\mathbf{n}}$.

The key operations for combining terms are sequential composition — $M \cdot N$ evaluates M and then supplies the result as an input to N — and coroutine composition — in $M \mid_j N$, control is initially given to M, calling j transfers it to N, calling $\overline{j}$ transfers it back, and so on, until a value has been computed. The other constants and operations may be described informally as follows:

Discard (or skip) $\star$ discards its input and returns the value $\star$,
Copy $?M$ evaluates M, and then returns its original input value,
Replication $!(j).M$ creates a chain of coroutines $M \mid_j M \mid_j \ldots$,
Zero and Successor 0 and succ are the corresponding functions,
Pairing (M, N) evaluates $\star \cdot M$ if its input is 0, and $n \cdot N$ if its input is $n+1$.

[2] We have made the structural rules of exchange and contraction explicit in order to simplify the definition of the categorical semantics. They may be made implicit by assuming that contexts are sets rather than sequences.

Table 1. Term formation rules

$$\frac{}{\Gamma,k{:}T\vdash k{:}T} \qquad \frac{\Gamma,k{:}R,j{:}S,\Delta\vdash M{:}T}{\Gamma,j{:}S,k{:}R,\Delta\vdash M{:}T} \qquad \frac{\Gamma,i{:}S,j{:}S\vdash M{:}T}{\Gamma,k{:}S\vdash M[k/i,k/j]{:}T}$$

$$\frac{\Gamma\vdash M{:}b^a \quad \Delta\vdash N{:}c^b}{\Gamma,\Delta\vdash M\cdot N{:}c^a} \qquad \frac{\Gamma,j{:}a^b\vdash M{:}c^d \quad \Delta,\overline{j}{:}b^a\vdash N{:}c^b}{\Gamma,\Delta\vdash M\mid_j N{:}c^d}$$

$$\frac{\Gamma\vdash M{:}\mathbf{1}^b}{\Gamma\vdash ?M{:}b^b} \qquad \frac{}{\Gamma\vdash\star{:}1^b} \qquad \frac{\Gamma,j{:}a^b,\overline{j}{:}b^a\vdash M{:}c^b}{\Gamma,\overline{j}{:}b^a\vdash !(j).M{:}c^b}$$

$$\frac{\Gamma\vdash M{:}b^{\mathbf{1}} \quad \Delta\vdash N{:}b^\omega}{\Gamma,\Delta\vdash(M,N){:}b^\omega} \qquad \frac{}{\Gamma\vdash 0{:}\omega^{\mathbf{1}}} \qquad \frac{}{\Gamma\vdash \mathtt{succ}{:}\omega^\omega}$$

The "non-returning fragment" (for which the denotational semantics is simpler) is obtained by constraining coroutine composition so that it cannot terminate by returning a value. We may express this restriction by requiring that the return type for coroutine composition and replication is empty — i.e. $M\mid_k N$ and $!(j).M$ must have types of the form $\mathbf{0}^b$.

2.1 Operational Semantics

We will now give a system of rewriting rules which allow us to reduce terms (possibly with free identifiers) to *head-normal forms.* A term is in head-normal form if it is a *value* — either $*$, or a numeral n represented as $(0\cdot\mathtt{succ})\cdot\ldots\cdot\mathtt{succ}$ — or has the form $(v\cdot k)\cdot M$. The rewrite rules for reducing *programs* (terms of the form $v\cdot M$) to head normal forms use the notion of an *evaluation context* $E[\_]$[3] which is defined by the grammar:

$$E[\_] ::= [\_] \mid E[\_\cdot M] \mid E[\_\mid_k M]$$

For each context of the form $C[\_] = M\cdot E[\_]$ or $C[\_] = M\mid_k E[\_]$, we obtain a well-typed term $C[\Box]$ as follows:
$M\cdot\Box = M$ and $M\mid_k\Box = M$, $C[\Box\cdot M] = C[M]$ and $C[\Box\mid_k M] = C[\Box]$.

The small-step reduction rules for programs are as follows:

$$\star\cdot E[0]\longrightarrow 0\cdot E[\Box] \qquad v\cdot E[\mathtt{succ}]\longrightarrow(v\cdot\mathtt{succ})\cdot E[\Box]$$
$$v\cdot E[\star]\longrightarrow\star\cdot E[\Box] \qquad v\cdot E[?M]\longrightarrow v\cdot E[M\cdot v]$$
$$0\cdot E[(M,N)]\longrightarrow\star\cdot E[M] \qquad (v\cdot\mathtt{succ})\cdot E[(M,N)]\longrightarrow v\cdot E[N]$$
$$v\cdot E_k[k]\longrightarrow(v\cdot k)\cdot E_k[?\star] \qquad v\cdot E[(E'_k[k]\mid_k M)]\longrightarrow v\cdot E[M\mid_{\overline{k}} E'[\Box]]$$
$$v\cdot E[!(j).M]\longrightarrow v\cdot E[M\mid_j !(j).M]$$

Given $M : a^{\mathbf{1}}$, we write $M\Downarrow$ if reduction of $\star\cdot M$ terminates. Since every program reduces, either to another program or to a head-normal form, $M\Downarrow$ entails that there exists a head-normal form H such that $\star\cdot M\twoheadrightarrow H$, and we write

[3] For any coroutine name k, the evaluation contexts which *do not* bind k are written $E_k[\_]$.

$M \Downarrow H$. We may now define standard notions of observational approximation and equivalence.

Definition 1. *Given terms $M, N : T$, we write $M \lesssim N$ if for any context $C[\_]$, $C[M] \Downarrow$ implies $C[N] \Downarrow$, and $M \simeq N$ if $M \lesssim N$ and $N \lesssim M$.*

3 Expressiveness

We give macros for a variety of programming constructs, both to demonstrate the expressive power of the calculus, and so that they can subsequently be used to express more complex features. Simple examples include the identity function — $I : a^a \equiv ?\star$ — and a divergent term at every type — $\Omega \equiv !(j).j$ — from which we may derive the predecessor function — $\mathtt{pred} : \omega^\omega \equiv (\Omega, I)$.

Labelled Jumps. Using a single coroutine swap we can express a form of GOTO — from $\Gamma, k : a^b \vdash M : c^d$, form $\Gamma \vdash \mathtt{label}\ k.M : b^d \equiv M \cdot \Omega \mid_k I$, for which we may derive the reduction:

$$v \cdot E[\mathtt{label}\ k.E_k[k]] \twoheadrightarrow v \cdot E[\Box]$$

Note that if we add labelled jumps to the non-returning fragment then we regain the power of the full calculus. However, the restriction to the non-returning fragment does not represent a significant restriction on expressive power; we may *CPS* translate each term $\Gamma \vdash M : a^b$ in the full calculus to a term $\Gamma, k : \mathbf{0}^a \vdash M^k : \mathbf{0}^b$ in the non-returning fragment, such that $M \Downarrow v$ if and only if $M^k \Downarrow v \cdot k$:

Loops. Replication is a powerful form of recursion; for instance, we may use it to define stacks (see below). We obtain a simpler form of recursion if we require that $\overline{j}$ is not free in M when we form $!(j).M$. For example, we may define *while* loops as follows:

$$\mathtt{while}\ M = 0\ \mathtt{do}\ N \equiv !(k).(M \cdot (N \cdot k, \star))$$

The *regular* calculus is the finitary calculus in which replication is restricted in this way. In fact, every regular term can be expressed using *iteration* — from $\Gamma \vdash M : a^a$ derive $\Gamma \vdash M^* : \mathbf{0}^a \equiv !(x).(M \cdot x)$ — instead of replication.

Procedure Calls. Given terms in-context $\Gamma \vdash M : a^b$ and $\Gamma, x : a^b \vdash N : C$, we define $[x \mapsto M].N \equiv N \mid_x (M \cdot \overline{x})^*$, with the intention that $[x \mapsto M].E_x[x]$ is equivalent to $[x \mapsto M].E_x[M]$ (and more generally, $[x \mapsto M].N \simeq N[M/x]$). The recursive version of this operation uses replication — if $x : b^b$ may occur free in $M : b^b$ then we define $\mathtt{rec}\ [x \mapsto M].N \equiv N \mid_x !(x).(M \cdot \overline{x})^*$.

Parameterization. It is useful to be able to abstract variables of value type — from $\Gamma, x : \mathbf{n}^1 \vdash M : \mathbf{m}^1$, form $\Gamma \vdash \delta x.M : \mathbf{m}^n$, the intended equivalences being $v \cdot \delta x.E_x[x] \simeq v \cdot \delta x.E_x[v]$, and $v \cdot \delta x.M \simeq M[v/x]$. We may represent parameterization by composition with a coroutine, defined using iteration and copying, which always returns the value with which it was initialized:

$$\delta x.M \equiv (?\overline{x})^* \mid_{\overline{x}} \star \cdot M$$

Store. We can express locally bound integer references by setting up a coroutine which behaves as a reference cell:

$$\mathsf{cell}(k) \equiv (\delta u.(u \cdot \overline{k} \cdot (u, \mathtt{succ})))^*$$

when k is read (called with 0) it returns the last non-zero value written to it. Thus if $k := N \equiv N \cdot \mathtt{succ} \cdot k \cdot \star$, $\mathsf{read}(k) \equiv 0 \cdot k \cdot \mathtt{pred}$ and $\mathtt{new}\, k := v.M \equiv M \mid_k v \cdot \mathsf{cell}(k)$, the intended equivalences are:

$$\star \cdot E[\mathtt{new}\, k := v.E_k[\mathsf{read}(k)]] \simeq v \cdot E[\mathtt{new}\, k := v.E_k[\Box]]$$
$$v \cdot E[\mathtt{new}\, k := v.E_k[k := u]] \simeq \star \cdot E[\mathtt{new}\, k := u.E_k[\Box]]$$

Similarly, we can define a coroutine which behaves as a stack (for which we require replication):

$$\mathsf{stack}(k) \equiv (0 \cdot \overline{k}, ?(\star \cdot x))^* \mid_x !(x).(\mathtt{succ} \cdot \overline{k} \cdot (\overline{x}, ?(\star \cdot x)))^*$$

popping the stack (with $0 \cdot k$) returns the last non-zero value pushed onto it (with $v \cdot k$) which has not yet been popped, or 0 if it is empty. Turing completeness of the calculus with finitary datatypes follows, since we may simulate a Turing machine using two stacks.

4 Equivalence

We define an equational theory of program equivalence from the following axioms:

$$(j \cdot M) \mid_j N = N \mid_{\overline{j}} M \qquad M \mid_k N = M \ (k \notin FV(M))$$
$$(L \cdot M) \cdot N = L \cdot (M \cdot N) \qquad (L \cdot M) \mid_k N = L \cdot (M \mid_k N) \ (k \notin FV(L))$$
$$0 \cdot ?M = 0 \cdot M \cdot 0 \qquad \mathtt{succ} \cdot ?M = \mathtt{succ} \cdot ?M \cdot \mathtt{succ}$$
$$\star \cdot M = M \qquad (0 \cdot M, \mathtt{succ} \cdot M) = M$$
$$0 \cdot (M, N) = M \qquad \mathtt{succ} \cdot (M, N) = N$$
$$0 \cdot \star = \star \qquad \mathtt{succ} \cdot \star = \star$$
$$!(j).M = M \mid_j !(j).M$$

Note that if $E[\_]$ is an evaluation context which does not bind any of the variables in M, then $E[M] = M \cdot E[\Box]$ is derivable. Using this fact it is straightforward to verify that the operational semantics is sound with respect to the equational theory — i.e. if $M \longrightarrow N$ then $M = N$. We can also derive the equivalences mentioned in the previous section, except for $[x \mapsto M].N \simeq N[M/x]$ and $v \cdot \delta x.M \simeq M[v/x]$. To extend our theory to equivalences of this kind, and to divergence, we define a natural notion of infinitary normal form, or *evaluation tree* (analogous to a Böhm tree in the λ-calculus), with the intention that terms are equivalent if they have the same evaluation tree.

Definition 2. *The evaluation tree of a term $M : a^b$ is a set of approximants $\{M_{ij} : a^b \mid i, j \in \mathbb{N}\}$ defined as follows:*
If $b = \mathbf{0}$, then $M_{ij} = \Omega$, and if $b = \mathbf{1}$, then $M_{0j} = \Omega$ and:

- *if* $M \Downarrow v$, *then* $M_{i+1j} = v$,
- *if* $M \Downarrow (v \cdot k) \cdot N$, *then* $M_{i+1j} = v \cdot k \cdot N_{ij}$,
- *if* $M \not\Downarrow$, *then* $M_{i+1j} = \Omega$,

If $b = \omega$, *then* $M_{ij} = ((0 \cdot M)_{ij}, \ldots, (j \cdot M)_{ij}, \Omega)$.

We will now show that the evaluation tree theory is *complete* — i.e. terms with distinct evaluation trees can be distinguished operationally.

Proposition 1. $M \lesssim N$ *implies* $E(M) \subseteq E(M')$.

Proof. Given terms $x_1, \ldots, x_n \vdash M : a^b$ and $x_1, \ldots, x_n \vdash M' : a^b$, suppose $E(M) \not\subseteq E(M')$. Then there exists a least i such that $M_{ij} \neq M'_{ij}$ for some j. We prove by induction on i that there exist terms $v, N_1(\overline{x_1}), \ldots, N_n(\overline{x_1}), P$ such that $v \cdot (\ldots (M \cdot P|_{x_1} N_1) \ldots |_{x_n} N_n) \Downarrow$ and $v \cdot (\ldots (M \cdot P|_{x_1} N_1) \ldots |_{x_n} N_n) \not\Downarrow$.

The converse of this proposition is also true; we will use the denotational semantics of the calculus to prove this in the next section.

5 Denotational Semantics

We will now give a description of the denotational semantics for our calculus. This takes the form of a notion of categorical model, based on symmetric monoidal categories, and a concrete example of a such a model in a category of games. Thus we obtain a new form of categorical description of sequential processes which also connects coroutines to the categorical structures used to model linear logic and higher-order functions; in the next section we will use this structure to extract a translation of the untyped λ-calculus. The games model establishes the consistency of the categorical and equational axioms, as well as formalizing the "folklore" correspondence between coroutines and strategies.

The first requirement for the categorical semantics is an affine category of comonoids — a SMC $(\mathcal{C}, I, \otimes)$ such that I is a terminal object, and for each object A in $\mathcal{C}$ there is a map $\delta : A \to A \otimes A$ such that (A, δ_A, t_A) is a comonoid (where $t : A \to I$ is the terminal map). Thus we have natural transformations $\pi_i : \_1 \otimes \_2 \to \_i$ for each i. We will interpret terms-in-context $x_1 : S_1, \ldots, x_n : S_n \vdash M : T$ in an affine category of comnoids as morphisms from $[\![S_1]\!] \otimes \ldots \otimes [\![S_n]\!]$ to $[\![T]\!]$. The projections, diagonal map, and twist maps $\theta_{A,B} : A \otimes B \to B \otimes A$ for the symmetric monoid yield obvious interpretations of the structural rules of weakening, contraction and exchange.

Our concrete example will be a simple category $\mathcal{G}$ of Abramsky-Jagadeesan-style games [1,2] (in fact, our category is a full subcategory of the category of AJM games and history sensitive strategies). A *game* A is simply a set-indexed set of sets $\{R_q \mid q \in Q_A\}$. We refer to the elements of the indexing set Q_A as *queries* and to the elements of each set R_q as *responses* (to q). The set of *moves* M_A is the disjoint union of all queries and reponses. The set L_A of *legal sequences* of A consists of all alternating sequence of queries and responses such that each response from R_q is preceded by the query q.

A morphisms from A to B in in $\mathcal{G}$ (a *strategy*) is a non-empty subset of $(M_A + M_B)^*$ satisfying the following conditions:

Projection and Switching. If $s \in \sigma$, then $s\restriction A \in L_A$ and $s\restriction B \in L_B$, if $s\restriction B = \varepsilon$ then $s\restriction A = \varepsilon$, and $s\restriction A$ and $s\restriction B$ have equal parity.

Determinacy and Even-prefix Closure. If $sab, sac \in \sigma$ then $s \in \sigma \wedge b = c$.

We form a symmetric monoidal category $\mathcal{G}$ with games as objects and strategies from A to B as morphisms from A to B using essentially the same definitions as [2]. In particular, *composition* of strategies $\sigma : A \to B$ and $\tau : B \to C$ is by "parallel composition plus hiding" — i.e. $\sigma;\tau =$

$$\{s \in (M_A + M_C)^* \mid \exists t \in (M_A + M_B + M_C)^*.s = t\restriction A, C \wedge t\restriction A, B \in \sigma \wedge t\restriction B, C \in \tau\}.$$

On objects, the tensor product is the disjoint sum of indexed sets, and on strategies, it is defined via restriction as in [1,2]. The unit for $\otimes$ is the empty game, which is also a terminal object. For each game A we have a copycat strategy $\delta_A : A \to A \otimes A$.

Proposition 2. $(\mathcal{G}, \otimes)$ *is an affine category of comonoids.*

To interpret types in our categorical model, we require interpretations of each basic type $\mathbf{n}$ as an object $\underline{n}$. We interpret the type $\mathbf{m^n}$ as a n-fold tensor product of copies of $\underline{m}$. Thus for each countable cardinal m we require a functor $(\_)^m : \mathcal{C} \to \mathcal{C}$ such that $(\_)^{m+1} \cong Id_{\mathcal{C}} \otimes (\_)^m$[4], and $((\_) \otimes (\_))^m \cong (\_)^m \otimes (\_)^m$. (Thus for finite m, we may define $A^0 = I$, $A^{n+1} = A \otimes A^n$.)

In $\mathcal{G}$, $\underline{n}$ is the game with a single query and n distinct responses, and $\underline{n}^m$ is the game with a m queries, each of which has n responses. Thus the legal sequences of $\underline{n}^m$ correspond to sequences of natural numbers $q_1 r_1 q_2 r_2 \ldots$, where $q_i < m$ and $r_i < n$ for all i.

We now describe the categorical structure required to soundly interpret the term formation rules.

- To interpret $\cdot$, an associative "internal composition" operation; for each triple l, m, n, a morphism $\gamma_{l,m,n} : \underline{m}^l \otimes \underline{n}^m \to \underline{n}^l$ such that:

$$(\gamma_{a,b,c} \otimes \mathsf{id}_{\underline{d}^c});\gamma_{a,c,d} = (\mathsf{id}_{\underline{b}^a} \otimes \gamma_{b,c,d});\gamma_{a,b,d}$$
$$(\mathsf{id}_{\underline{b}^{a+1}} \otimes \delta_{\underline{c}^b});(\mathsf{id}_{\underline{b}} \otimes \theta_{c,\underline{b}^a} \otimes \mathsf{id}_{\underline{c}^b});(\gamma_{1,b,c} \otimes \gamma_{a,b,c}) = \gamma_{(a+1),b,c}$$

  In $\mathcal{G}$, $\gamma_{l,m,n}$ is the obvious copycat strategy which copies a query in $\underline{n}^l$ to $\underline{m}^l$, uses the response as a query in $\underline{n}^m$, and then copies the response to answer the original query.
- For every m, n, a natural isomorphism $\Lambda(\_) : \mathcal{C}(A \otimes \underline{m}^n, \underline{0}^m) \to \mathcal{C}(A, \underline{n}^m)$ (i.e. $\underline{n}^m$ is the *exponential* of $\underline{m}^n$ by 0^m) such that:

$$\epsilon_{m,n} = (\delta_{\underline{m}^n} \otimes \mathsf{id}_{\underline{n}^m});(\mathsf{id}_{\underline{m}^n} \otimes (\theta_{\underline{n}^m,\underline{m}^n};\epsilon_{n,m}));\gamma_{n,m,0} \qquad (*)$$

[4] To simplify notation, we assume in the following that this isomorphism, together with the associativity and unit isomorphisms for the tensor product are all identities — i.e. we assume $\mathcal{C}$ is strict monoidal.

Table 2. Interpretation of terms in the categorical model

$$\begin{gathered}
[\![\Gamma, x : T \vdash x : T]\!] = \pi_r \\
[\![\Gamma, x : R, y : S, \Delta \vdash M : T]\!] = (\mathsf{id}_\Gamma \otimes \theta_{[\![R]\!],[\![S]\!]} \otimes \mathsf{id}_\Delta); [\![\Gamma, y : S, x : R, \Delta \vdash M : T]\!] \\
[\![\Gamma, z : S \vdash M[z/x, z/y] : T]\!] = (\mathsf{id}_\Gamma \otimes \delta_{[\![S]\!]}); [\![\Gamma, y : S, x : R, \Delta \vdash M : T]\!] \\
[\![\Gamma \vdash \star : \mathbf{1^n}]\!] = t_\Gamma; \mathsf{skip^n}, \\
[\![\Gamma \vdash 0 : \omega^1]\!] = t_\Gamma; \mathsf{z} \\
[\![\Gamma \vdash \mathsf{succ} : \omega^\omega]\!] = t_\Gamma; \mathsf{s} \\
[\![\Gamma, \Delta \vdash M \mid_k N : \mathbf{0^l}]\!] = (\mathsf{id}_{[\![\Gamma]\!]} \otimes \Lambda([\![\Delta, k : \mathbf{m^n} \vdash N : \mathbf{0^m}]\!])); [\![\Gamma, j : \mathbf{n^m} \vdash \mathbf{0^l}]\!] \\
[\![\Gamma, \Delta \vdash (M, N) : b^\omega]\!] = ([\![\Gamma \vdash M : b^1]\!] \otimes [\![\Delta \vdash N : b^\omega]\!]) \\
[\![\Gamma \vdash M \cdot N : \mathbf{n^l}]\!] = ([\![\Gamma \vdash M : \mathbf{m^l}]\!] \otimes [\![\Gamma \vdash N : \mathbf{m^n}]\!]); \gamma_{l,m,n} \\
[\![\Gamma \vdash ?M : \omega^\omega]\!] = [\![\Gamma \vdash M : \mathbf{1}^\omega]\!]; \mathsf{copy} \\
[\![\Gamma, \vec{j} : \mathbf{n^m} \vdash !(j).M : \mathbf{0^n}]\!] = \Lambda^{-1}((\Lambda([\![\Gamma, j : \mathbf{m^n}, \vec{j} : \mathbf{n^m} \vdash M : \mathbf{0^n}]\!]))^\dagger)
\end{gathered}$$

where $\epsilon_{n,m} : \underline{n}^m \otimes \underline{m}^n \to \underline{0}^m = \Lambda^{-1}(\mathsf{id}_{\underline{n}^m})$, and thus $\Lambda^{-1}(f) = (\mathsf{id} \otimes f); \epsilon$. This is really the key equation for our categorical model, as it allows us to unfold coroutine composition in terms of the twist isomorphism and internal composition. In $\mathcal{G}$, we have an isomorphism between strategies in $\mathcal{G}(A \otimes \underline{m}^n, \underline{0}^m)$ and $\mathcal{G}(A, \underline{n}^m)$ arising from an isomorphism between the corresponding sets of sequences.

- To interpret $\star$, 0 and succ, we require distinguished morphisms $\mathsf{skip} : I \to \underline{1}$, $\mathsf{z} : I \to \underline{\omega}$ and $\mathsf{s} : I \to (\underline{\omega})^\omega$ such that:

$$\mathsf{skip} \otimes \mathsf{id}_{\underline{n}}; \gamma_{1,1,n} = \mathsf{id}_{\underline{n}} \ (\mathsf{z} \otimes \mathsf{id}_{\underline{n}^\omega}); \gamma_{1,\omega,n} = \pi_l \text{ and } (\mathsf{s} \otimes \mathsf{id}_{\underline{n}^\omega}); \gamma_{\omega,\omega,n} = \pi_r$$

  In our category of games, z is the strategy which responds to every query with zero, and s responds to the query i with the response $i + 1$.
- To interpret replication, a *parameterised fixpoint operator* — i.e. an operation $(\_)^\dagger : \mathcal{C}(A \otimes B, A) \to \mathcal{C}(B, A)$ such that $f^\dagger = \delta_B; (f^\dagger \otimes \mathsf{id}_B); f$.
  In a cpo-enriched model (such as $\mathcal{G}$, where the order is set-theoretic inclusion of strategies), we may obtain $f^\dagger$ as the least fixpoint of the operation sending $g : B \to A$ to $\delta_B; (g \otimes \mathsf{id}_B); f$. We will say that such a model is *continuous.*

To interpret the copy operation, we derive a morphism $\mathsf{copy} : \underline{1}^\omega \to \underline{\omega}^\omega$ using the monoidal isomorphism $\mathsf{mon}_{A,B} : A^\omega \otimes B^\omega \cong (A \otimes B)^\omega$:

$$\mathsf{copy} : \underline{\omega} \to \underline{\omega}^\omega = (\mathsf{z} \otimes \mathsf{succ}); \mathsf{mon}_{\underline{1},\underline{\omega}}; \gamma^\omega_{1,1,\omega}.$$

The interpretation of terms from the non-returning fragment using the above structure is given in Table 2. Note that composition of coroutines is interpreted simply as composition of the corresponding morphisms (and hence, in $\mathcal{G}$, as the "parallel composition plus hiding" of strategies). We may give the semantics of the general form of coroutine composition either by CPS translation into the non-returning fragment, or by interpreting the `label` binding operation as a natural map from $\mathcal{C}(A \otimes \underline{0}^m, \underline{0}^n)$ to $\mathcal{C}(A, \underline{m}^n)$. The soundness of the model then follows straightforwardly from the above definitions.

Lemma 1. *The equational axioms hold in the categorical model.*

To prove that any continuous model is *computationally adequate* we use approximants from the "replication free" fragment (in which we still allow Ω).

Lemma 2. *For any replication-free term M, either $M \Downarrow$, or $\star \cdot M \twoheadrightarrow v \cdot E[\Omega]$.*

Proof. We observe that all reduction rules except replication strictly reduce the size of terms with respect to the following order: $M < N$ if either M has fewer occurrences of copying than N, or M has no more occurrences of copying, and is shorter.

By replacing instances of replication $!(j).N$ with approximants of the form $N \mid_j \ldots \mid_j N \mid_j \Omega$ we then prove adequacy.

Proposition 3. *In any (non-trivial) continuous model, $\mathcal{C}$, $M \Downarrow$ iff $[\![M]\!]_{\mathcal{C}} \neq \bot$.*

To characterize the *fully abstract* categorical models, we require a further condition on the functor $(\_)^\omega$. Recall that a *minimal invariant* [8] for an endofunctor $F : \mathcal{C} \to \mathcal{C}$ on a cpo-enriched category is an object $\Delta(F)$ such that there is an isomorphism $\mathsf{out} : \Delta(F) \cong F(\Delta(F)) : \mathsf{in}$, and $\mathsf{id}_{\Delta(F)}$ is the least fixpoint of the operation which takes $f : \Delta(F) \to \Delta(F)$ to $\mathsf{out}; F(f); \mathsf{in} : \Delta(F) \to \Delta(F)$. In the following, we will say that $(\_)^\omega$ is a minimal invariant if for any A, A^ω is a minimal invariant for the functor $A \otimes \_$ — i.e. if id_{A^ω} is the least upper bound of the chain of morphisms $\{\mathsf{id}_A^i \otimes \bot_{A^\omega} \mid i \in \omega\}$. Clearly for any game A, $A^\omega = \coprod_{j \in \omega} A$ is a minimal invariant.

Lemma 3. *If $\mathcal{C}$ is continuous and $(\_)^\omega$ is a minimal invariant, then for any term M, $[\![M]\!]_{\mathcal{C}} = \bigsqcup_{i \in \omega} \bigsqcup_{j \in \omega} [\![M_{ij}]\!]_{\mathcal{C}}$.*

Proof. We prove this first for replication-free terms, by induction on the ordering defined in Lemma 2. If $M : b^{\mathbf{1}}$, then either $\star \cdot M \longrightarrow v \cdot E[\Omega]$, and $[\![M]\!] = \bot$, or else $M \Downarrow v \cdot k \cdot M'$, where $M' < M$ and hence we can apply the induction hypothesis to M'. If $M : b^\omega$, then we use the fact that $M = \bigsqcup_{j \in \omega} [\![(0 \cdot M, \ldots, j \cdot M, \Omega)]\!]$ by minimal invariance. Now for any M, $[\![M]\!] = \bigsqcup_{n \in \omega} [\![M^n]\!] = \bigsqcup_{n \in \omega} \bigsqcup_{i \in \omega} \bigsqcup_{j \in \omega} [\![M^n_{ij}]\!] = \bigsqcup_{i \in \omega} \bigsqcup_{j \in \omega} [\![M_{ij}]\!]$.

Theorem 1. *For any terms M, N the following are equivalent:*

i $[\![M]\!] \leq [\![N]\!]$ *in all continuous categorical models with minimal invariants.*
ii $[\![M]\!] \subseteq [\![N]\!]$ *in $\mathcal{G}$,*
iii $M \lesssim N$,
iv $E(M) \subseteq E(N)$.

Proof. (*i*) $\Longrightarrow$ (*ii*) holds because $\mathcal{G}$ is a continuous model. (*ii*) $\Longrightarrow$ (*iii*) by the standard argument showing that a computationally adequate model is inequationally sound. (*iii*) $\Longrightarrow$ (*iv*) by Proposition 1. (*iv*) $\Longrightarrow$ (*i*) since $E(M) \subseteq E(N)$ implies that $\forall i, j \in \omega. [\![M_{ij}]\!] \sqsubseteq [\![N]\!]$ and hence $[\![M]\!] = \bigsqcup_{i \in \omega} \bigsqcup_{j \in \omega} [\![M_{ij}]\!] \sqsubseteq [\![N]\!]$.

Note that there is a direct correspondence between deterministic strategies and evaluation trees: the denotation of $\Gamma \vdash (v_1 \cdot k_1 \cdot M_1, \ldots, v_i \cdot k_i \cdot M_i, \ldots) : 0^{\mathbf{n}}$ is a strategy which responds to the query i on the right with the query v_i in the i^{th} component on the left, and then plays as M_i. In other words we can easily establish the following *definability property.*

Lemma 4. *For any compact strategy $\sigma : [\![\Gamma]\!] \to [\![T]\!]$ there is a term $\Gamma \vdash M_\sigma : T$ such that $\sigma = [\![M_\sigma]\!]$.*

6 Higher-Order Procedures

We will now show that our categorical model also has enough of the structure of a model of linear logic to interpret the untyped (call-by-name) λ-calculus, λ_N.

Lemma 5. *If A^ω is a minimal invariant then $A^\omega \cong A^\omega \otimes A^\omega$.*

Proof. We define an isomorphism $\phi : A^\omega \to A^\omega \otimes A^\omega$ by taking the least fixed point of the continuous function $\Phi : \mathcal{C}(A^\omega, A^\omega \otimes A^\omega) \to \mathcal{C}(A^\omega, A^\omega \otimes A^\omega)$ which sends $f : A^\omega \to A^\omega \otimes A^\omega$ to $(\mathsf{id}_A \otimes (\mathsf{id}_A \otimes f); \theta_{A^\omega, A^\omega}); \theta_{A^\omega, A^\omega}$. We obtain ϕ^{-1} as the least fixed point of the analogous operation $\widehat{\Phi} : \mathcal{C}(A^\omega \otimes A^\omega, A^\omega) \to \mathcal{C}(A^\omega \otimes A^\omega, A^\omega)$. Then by minimal invariance, $\psi; \psi^{-1} = (\bigsqcup_{i \in \omega} \Phi_A^n(\bot)); (\bigsqcup_{i \in \omega} \widehat{\Phi}_A^n(\bot)) = \bigsqcup_{i \in \omega} (\mathsf{id}_A^n \otimes \bot) = \mathsf{id}_{A^\omega}$, and vice-versa.

Lemma 6. *If $\underline{\omega}^\omega$ is a minimal invariant then $\underline{\omega}^\omega$ is an exponential of $\underline{\omega}^\omega$ by $\underline{\omega}^\omega$.*

Proof. We have a natural isomorphism $\mathcal{C}(A \otimes \underline{\omega}^\omega, \underline{\omega}^\omega) \cong \mathcal{C}(A \otimes \underline{\omega}^\omega \otimes \underline{\omega}^\omega, \underline{0}^\omega) \cong \mathcal{C}(A \otimes \underline{\omega}^\omega, \underline{0}^\omega) \cong \mathcal{C}(A, \underline{\omega}^\omega)$, as required.

Thus we can construct a model of the untyped *affine* λ-calculus in $\mathcal{C}$, interpreting terms-in-context $x_1, \ldots, x_n \vdash M$ as morphisms from $(\underline{\omega}^\omega)^n$ to $\underline{\omega}^\omega$. To extend this interpretation to non-affine terms we use the following observation

Lemma 7. *If A^ω is a minimal invariant then $A^\omega \cong (A^\omega)^\omega$.*

Proof. We use minimal invariance as in Lemma 5 to define $\psi_A : A^\omega \cong (A^\omega)^\omega$.

Lemmas 5 and 7 allow us to infer that the functor $(\_)^\omega$ is in fact a *monoidal comonad* on $\mathcal{C}$. The co-Kleisli category of this co-monad is thus a model of the λ-calculus, based on the object $\underline{\omega}^\omega$, which is a *reflexive object* (i.e. $(\underline{\omega}^\omega)^{\underline{\omega}^\omega} \trianglelefteq \underline{\omega}^\omega$). A corresponding games model of λ_N has been described by Longley [7].

We will not, however, use such an interpretation of λ_N, principally because the requirement to represent the *promotion* rule of linear logic introduces a heavy syntactic overhead into the associated translation. We will use an alternative notion of promotion, which leads to a simpler translation of application (although it is not sound with respect to η-equivalence).

Definition 3. *We define a map $\delta(\omega) : A \to A^\omega$ as the least fixpoint of the continuous map from $\mathcal{C}(A, A^\omega)$ to itself which sends f to $\delta_A; (\mathsf{id}_A \otimes f)$. For any morphism $f : A \to B^\omega$, we define $f^\ddagger : A \to B^\omega = \delta(\omega); f^\omega; \psi_B^{-1}$.*

We also define maps $\mathsf{der}_A : A^\omega \to A^\omega = \psi_A; \pi_l^\omega$ and $\mathsf{con}_A : A^\omega \to A^\omega \otimes A^\omega = \psi_A; \mathsf{mon}_A^\omega; (\psi_A^{-1} \otimes \psi_A^{-1})$. The following lemma follows from minimal invariance.

Lemma 8. *For any* $f : A \to B^\omega$, $f^\ddagger; \mathsf{der}_B = f$ *and* $f^\ddagger; \mathsf{con}_B = \delta_B; (f^\ddagger \otimes f^\ddagger)$.

We may thus give a semantics of λ_N in $\mathcal{C}$, interpreting a term $x_1, \ldots, x_n \vdash M$ as a morphism from $(\underline{\omega}^\omega)^n$ to $\underline{\omega}^\omega$ as follows:

- $[\![\Gamma, x \vdash x]\!] = \pi_l; \mathsf{der}$
- $[\![\Gamma \vdash \lambda x.M]\!] = \Lambda([\![\Gamma, x \vdash M]\!])$
- $[\![\Gamma \vdash M\ N]\!] = \mathsf{con}_\Gamma; ([\![\Gamma \vdash M]\!]^\ddagger \otimes [\![\Gamma \vdash N]\!]); \mathsf{app}$

Proposition 4. *This interpretation of* λ_N *is sound with respect to* β*-equivalence.*

Proof. $[\![\Gamma, \Delta \vdash (\lambda x.M)\ N]\!] = (\mathsf{id}_\Gamma \otimes [\![\Delta \vdash N]\!]^\ddagger); [\![\Gamma, x \vdash M]\!]$. We show by structural induction using Lemma 8 that this is equal to $[\![\Gamma, \Delta \vdash M[N/x]]\!]$.

We will now give a translation of λ_N into our calculus of coroutines and show that it corresponds to the categorical interpretation. Our first requirement is a representation of the encodings of natural number co-products and products implicit in the isomorphisms $A^\omega \otimes A^\omega \cong A^\omega$ and $(A^\omega)^\omega$. We can represent these primitive recursive functions as operations in our calculus, for which we give the defining properties and omit the details. The co-pairing operation — from $M : b^\omega$ and $N : b^\omega$, form $[M, N] : b^\omega$ — is supplied with closed terms $\iota_i : \omega^\omega$ for $i \in \{1, 2\}$ such that $\iota_i \cdot [M_1, M_2] \equiv M_i$, and $[\iota_1 \cdot M, \iota_2 \cdot M] \equiv M$. The pairing operation — from $M : \omega^b$ and $N : \omega^b$, form $\langle M, N\rangle : \omega^b$ — comes with closed terms $\pi_i : \omega^\omega$ for $i \in \{1, 2\}$ such that $\langle v_1, v_2\rangle \cdot \pi_i \equiv v_i$ and $\langle M \cdot \pi_1, M \cdot \pi_2\rangle \equiv M$. We also assume an equality test taking terms $M, N : \omega^{\mathbf{1}}$ to a term $(M = N) : \omega^{\mathbf{1}}$ which evaluates M and N and produces output 0 if they are equal, and 1 otherwise.

The translation is given in a continuation-passing style: for each λ-term $M(x_1, \ldots, x_n)$, we define $x_1 : \omega^\omega, \ldots, x_n : \omega^\omega, k : \omega^\omega \vdash M^k : \mathbf{0}^\omega$ as follows:

- $x^k = (\langle 0, I\rangle \cdot x \cdot k)^*$
- $(\lambda x.M)^k = \mathtt{Lambda}_{x,j,k}(M^j)$
- $(M(x_1, \ldots, x_n)\ N(x_1, \ldots, x_n))^k =$
  $\mathtt{App}_{j,l,k}(\mathtt{Prom}_{i,j}(M^i[(\langle \boldsymbol{\pi_1} \cdot \boldsymbol{\iota_1}, \boldsymbol{\pi_2}\rangle \cdot \boldsymbol{x})/\boldsymbol{x}]), N^l[(\langle \boldsymbol{\pi_1} \cdot \boldsymbol{\iota_2}, \boldsymbol{\pi_2}\rangle \cdot \boldsymbol{x})/\boldsymbol{x}])$

where Lambda, App and Prom are operations on coroutine terms defined as follows:

- $\mathtt{Lambda}_{x,j,k}(M) = [j \mapsto \iota_1 \cdot k].[x \mapsto \iota_2 \cdot k].M$
- $\mathtt{App}_{i,j,k}(M, N) = ([i \mapsto [k, \bar{j}]].M^i) \mid_{\bar{j}} N^j.$
- $\mathtt{Prom}_{i,k}(M) = j^* \mid_j !(j).(\delta a.a \cdot (\delta b.(a \cdot \pi_1 = b \cdot \pi_1) \cdot ((b \cdot \pi_2 \cdot \bar{i} \cdot k), b \cdot j) \cdot \bar{j})^* \mid_{\bar{i}} M)$

We prove the following Proposition using the fact that Lambda, $\mathtt{App}_{i,j,k}$ and Prom are interpreted as the corresponding operations in the categorical model.

Proposition 5. *For any term* $\Gamma \vdash M$ *of* λ_N, $[\![\Gamma \vdash M]\!]_{\mathcal{C}} = \Lambda([\![\Gamma, k \vdash M^k]\!])$.

Corollary 1. *For any* λ*-terms* $\lambda x.M, N$ *we have* $(\lambda x.M\ N)^k \simeq M[x/N]$.

7 Further Directions

In this paper we have restricted our attention to the study of deterministic sequential processes, although coroutines are more usually considered in the context of implementing concurrent computation. In fact, we can represent concurrent threads in our calculus by adding an erratic choice operator; from $M_1 : a^b$ and $M_2 : a^b$ form $(M_1 + M_2) : a^b$, with the reduction rule:

$$v \cdot E[M_1 + M_2] \longrightarrow v \cdot E[M_i] \quad (i \in \{1, 2\}).$$

The following translation of parallel composition of processes and CSP-style message passing and channel restriction uses an additional "resumption" variable k, and implements asynchronous communication using a stack for each channel:

$$\begin{array}{cc} (\mathsf{snd}\, c\, v)^k \equiv \mathsf{push}(v, c) \cdot \ k^* & (\mathsf{rcv}\, c)^k = !(j).\mathsf{pop}(c) \cdot (k \cdot j, I) \\ (M \| N)^k \equiv ((\overline{i} + \overline{j}) \cdot k)^* \mid_{\overline{i}} M^i \mid_{\overline{j}} N^j & (\nu c.M)^k \equiv M^k \mid_c \mathsf{stack}(c) \end{array}$$

This is just one representation of concurrency primitives amongst many. Comparison of these, and study of the operational and denotational properties of coroutines combined with non-determinism is left as future work.

Another possible extension, suggested by the π-calculus, would be to allow names to be passed between coroutines. The current restriction to "first-order" value-passing imposes a limit on the expressiveness of the calculus — we lack a straightforward representation of higher-order references, for example. However, extending the calculus to higher-order radically alters (and complicates) the nature of interaction, as names may be called outside their original scope. The study of an appropriate calculus for describing this kind of higher-order sequential processes is ongoing.

References

1. S. Abramsky, R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic,* 59:543–574, 1994.
2. S. Abramsky, R. Jagadeesan and P. Malacaria. Full abstraction for PCF. *Information and Computation,* 163:409–470, 2000.
3. M. Berger, K. Honda, and N. Yoshida. Sequentiality and the π-calculus. In *Proceedings of TLCA 2001,* volume 2044 of *Lecture Notes in Computer Science.* Springer-Verlag, 2001.
4. D. Ghica and G. McCusker. The regular language semantics of second-order Idealised Algol. *Theoretical Computer Science,* 2003. To appear.
5. J. M. E. Hyland and C.-H. L. Ong. Pi-calculus, dialogue games and PCF. In *Proceedings of the 7th ACM Conference on Functional Programming Languages and Computer Architecture,* pages 96–107. ACM Press, 1995.
6. J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation,* 163:285–408, 2000.
7. J. Longley. Universal types and what they are good for. In *Domain Theory, Logic and Computation: Proceedings of the 2nd International Symposium on Domain Theory.* Kluwer, 2004.
8. A. M. Pitts. Relational properties of domains. *Information and Computation,* 127:66–90, 1996.

Almost Optimal Decentralized Routing in Long-Range Contact Networks*

Emmanuelle Lebhar and Nicolas Schabanel

LIP (UMR CNRS, ÉNS Lyon, INRIA, Univ. Claude Bernard Lyon I)
École Normale Supérieure de Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France.
http://perso.ens-lyon.fr/{emmanuelle.lebhar/, nicolas.schabanel/}

Abstract. In order to explain the ability of individuals to find short paths to route messages to an unknown destination, based only on their own local view of a social network (the small world phenomenon), Kleinberg (2000) proposed a network model based on a d-dimensional lattice of size n augmented with k long range directed links per node. Individuals behavior is modeled by a greedy algorithm that forwards the message to the neighbor of the current holder, which is the closest to the destination. This algorithm computes paths of expected length $\Theta(\log^2 n/k)$ between any pair of nodes. Other topologies have been proposed later on to improve greedy algorithm performance. But, Aspnes et al. (2002) shows that for a wide class of long range link distributions, the expected length of the path computed by this algorithm is always $\Omega\big(\log^2 n/(k^2 \log\log n)\big)$.
We design and analyze a new decentralized routing algorithm, in which nodes consult their neighbors near by, before deciding to whom forward the message. Our algorithm uses similar amount of computational resources as Kleinberg's greedy algorithm: it is easy to implement, visits $O\big(\log^2 n/\log^2(1+k)\big)$ nodes on expectation and requires only $\Theta(\log^2 n/\log(1+k))$ bits of memory – note that [1] shows that any decentralized algorithm visits at least $\Omega(\log^2 n/k)$ on expectation. Our algorithm computes however an almost optimal path of expected length $O\big(\log n(\log\log n)^2/\log^2(1+k)\big)$, between any pair of nodes. Our algorithm might fit better some human social behaviors (such as web browsing) and may also have successful applications to peer-to-peer networks where the length of the path along which the files are downloaded, is a critical parameter of the network performance.

1 Introduction

The small world phenomenon. Since the experiment of Milgram in 1967 [2], showing that people are able to route very efficiently messages to an unknown destination through their own local acquaintances (even if only 25% of the messages actually arrived), several models [3,4] have been designed to capture this phenomenon. Numerous real graphs (such as the co-author graph, the web graph,

* This works was supported by the CNRS AS Dynamo and AS Grands Graphes grants.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 894–905, 2004.

peer-to-peer networks...) share similar properties: a very small diameter (typically poly-logarithmic in the size of the network) and the existence of short paths between random nodes, that can be found very efficiently, based only on the local view of the network.

Models for the small world phenomenon. Models for the small world phenomenon have recently received a renew of interest for their potential application to peer-to-peer networks [5,6]. Watts and Strogatz observed in [7] that most of the small world graphs are locally strongly interconnected and proposed a random rewiring model that yields a small diameter and strong local interconnections (see also [8,9]). But these models fail to capture the specific nature of a small world. In [10], Kleinberg demonstrated that, for these models, there does not exist any *decentralized algorithm* (i.e., using only local information) that can find poly-logarithmic length paths, even when the diameter is poly-logarithmic. He then introduced a new model, that in its most general form is a d-dimensional toric lattice augmented with k random directed links per node. The d-dimensional lattice represents the underlying geographic (or *local*) relationships between the individuals. Each node $\mathbf{u}$ is also the origin of $k \leq \log n$ directed links pointing to its k *long range contacts* $\mathbf{v}_1, \ldots, \mathbf{v}_k$, chosen randomly and independently according to the s-harmonic distribution, i.e., with probability proportional to $1/\delta(\mathbf{u}, \mathbf{v})^s$, where $\delta(\mathbf{u}, \mathbf{v})$ is the lattice (Manhattan) distance between $\mathbf{u}$ and $\mathbf{v}$. [10,11] demonstrate that when $s \neq d$, no decentralized algorithm can find a poly-logarithmic length path in the d-dimensional network. For $s = d$, a simple greedy algorithm is proposed, that forwards the message to the closest[1] neighbor of the current holder to the target until it reaches its destination. When $s = d$, this algorithm computes a path of expected length $\Theta(\log^2 n/k)$, between any random pair of nodes. This result demonstrates that there is more to the small world effect than simply the existence of short paths, and that the algorithmic nature of the experiment has to be considered. Variants of this graph, with undirected long range links, based on edge percolation, have been studied in [12,13,1].

Several topologies (e.g., [14,15]) have been proposed to improve the greedy algorithm performances, in the perspective of applications to peer-to-peer networks. [6] demonstrates that for a wide class of long range links distributions on the ring (including the one mentioned above), Kleinberg's greedy algorithm computes path of expected length $\Omega\big(\log^2 n/(k \log\log n)\big)$ (if it is not allowed to "jump over" the target, and $\Omega\big(\log^2 n/(k^2 \log\log n)\big)$ otherwise). In [1,16], the greedy router is aware of the long range contacts of the local neighbors closeby (at lattice distance ≤ 1 in [1] and $\leq \log^{1/d} n$ in [16]) before forwarding the message: the expected length of the computed path is improved to $O\big(\log^2 n/(k \log k)\big)$ in [1] (the network in [1] is also slightly different), and $\Theta(\log^{1+1/d} n)$ in [16].

Our contribution. In this paper, we design and analyze a new decentralized routing algorithm on the d-dimensional Kleinberg's small world model, that computes a path of expected length $O\big(\log n \cdot (\log\log n)^2/\log^2(1+k)\big)$ between

[1] According to the lattice distance.

any pair of nodes. Our algorithm visits $O\big((\log n/\log(1+k))^2\big)$ nodes on expectation to compute this path. The network load induced by the computation of the path and the *latency*[2] of our protocol is then very close to Kleinberg's greedy algorithm. Note that [1] proves that any decentralized routing algorithm visits at least $\Omega(\log^2 n/k)$ nodes. Our algorithm requires small resources as well: it only requires $O(\log^2 n/\log(1+k))$ bits of memory to store the addresses of $O(\log n/\log(1+k))$ nodes (for instance, in the message header); and it is fairly easy to implement. Note also that it is not based on searching for the highest degree nodes, and thus avoids overloading them. Applied to peer-to-peer networks, where the path length is a critical factor of performance (since downloaded files are often large), our algorithm could possibly reduce the load of the network.

2 Model and Main Results

The network. We consider the d-dimensional variant of the small world network model with $k \leq \log n$ long-range links per node, introduced by Kleinberg in [10]. The network is an augmented d-dimensional toric lattice $\{-n,\ldots,0,\ldots,n\}^d$ of $(2n+1)^d$ nodes. In addition to its $2d$ neighbors in the lattice (its *local contacts*), each node $\mathbf{u}$ is the origin of k directed links, each of them pointing towards a node $\mathbf{v}_j$, $1 \leq j \leq k$, ($\mathbf{u}$*'s* j*-th long-range contact*), chosen independently according to the d-harmonic distribution, i.e., with a probability proportional to $1/\delta(\mathbf{u},\mathbf{v})^d$, where $\delta(\mathbf{u},\mathbf{v})$ is the distance between $\mathbf{u}$ and $\mathbf{v}$ on the toric lattice.

In all the following, log stands for the logarithm base 2; In denotes the natural logarithm, base e, and $H_n = \sum_{i=1}^{n} 1/i$. Note that $\ln(n+1) < H_n < \ln n + 1$.

Decentralized routing algorithms. We study algorithms that compute a path to transmit a message or a file from a source to a target, along the local and (directed) long range links of the network. Following Kleinberg's definition, such an algorithm is *decentralized* if it navigates through the network using only local information to compute the path. In particular, it has the knowledge 1) of the underlying lattice structure (the d-dimensional torus), 2) of the coordinates of the target in the lattice, and 3) of the nodes it has previously visited as well as their long-range contacts. But, crucially, 4) it can *only* visit nodes that are local or long-range contacts of previously visited nodes, and 5) does not know the long-range contacts of any node that has not yet been visited. However, 6) the algorithm (but *not the path it computes*) is authorized to travel backwards along any directed links it has *already* followed. As Kleinberg pointed out in [17], this is a crucial component of human ability to find short paths: one can interpret point 6) as a web user pushing the back button, or an individual returning the letter to its previous holder (who wrote his address on the envelope before sending it).

The following theorem is the main result of this paper.

Theorem 1. *For any dimension d and $k \leq \log n$, there is a decentralized routing algorithm $\mathcal{A}$ using $\Theta(\log^2 n/\log(1+k))$ bits of memory such that, for any pair of nodes $(\mathbf{s},\mathbf{t})$, $\mathcal{A}$ computes a path from $\mathbf{s}$ to $\mathbf{t}$ of expected length*

[2] Defined as the time to compute the path.

$O\big(\log n \cdot (\log\log n/\log(1+k))^2\big)$, *and visits* $O\big((\log n/\log(1+k))^2\big)$ *nodes on expectation to compute this path.*

Our algorithm computes an almost optimal path in the following sense: the expected path length is $O(\log n(\log\log n/\log(1+k))^2)$, while the diameter of Kleinberg's network is lower bounded by $\Omega(\log n/\log(1+k))$ (every node has out-degree $2d+k$). The expected path length is thus optimal up to a $(\log\log n)^2/\log(1+k)$ factor. It shows in particular that Kleinberg's greedy algorithm does not compute an optimal path, nor a constant factor approximation.

We present below the depth-first search implementation of our algorithm which is the most time-efficient. We will however analyze in the following sections an equivalent (but less time-efficient) breadth-first search implementation that improves the readability of the proofs. In order to describe the algorithm, we introduce the following definitions.

Definition 2. *We say that a link (local or long-range) from a node* **u** *to a node* **v** *is* good *if* **v** *is strictly closer to the target than* **u**, *according to the lattice distance. We say then that* **v** *is a* good contact *(local or long-range) of* **u**.

A node **v** *is said to* be h good links away *from* **u**, *if there is a path of length* $\leq h$ *from* **u** *to* **v** *only composed of good links;* **v** *is* h local good links away *from* **u** *if this path is only composed of good local links.*

Every node **u** (but the target) is the origin of at least one good (local) link (and in fact, up to d local good links depending on the relative position of **u** to the target) and, with some probability, of up to k other good (long-range) links. We now describe the algorithm with the parameters set for the one-dimensional network (the parameters for the d-dimensional network are given Section 4 - the algorithm remains unchanged).

The algorithm (depth-first search implementation - one-dimensional network). Let **s** and **t** be respectively the source and the target. Let us assume we are given three functions $h_{\max}(x)$, $b_{\max}(x)$ and $g(x)$ whose values will be given in Section 3 and 4, for $d=1$ and $d\geq 2$ respectively. Let **x** the current holder of the message and π the current path from **s** to **x**.

While $\delta(\mathbf{x},\mathbf{t}) > k\log^2 n$: explore in depth-first order the nodes $h_{\max}(x)$ good links away from **x**, record in a set F all the good long range contacts visited, but skip in the search all the long range contacts that are at lattice distance $< h_{\max}(x)+g(x)$ from any node of the current set F. The depth-first search also stops as soon as $|F| = b_{\max}(x)$. Each time a node, **z**, exactly $h_{\max}(x)$ good links away from **x** is reached, read the addresses of the long range contacts of the nodes $g(x)$ good local links away from **z** and record in a variable **y** the closest node to the target (according to the lattice distance) among the visited nodes and their contacts. At the end of the depth-first exploration, route the message from **x** to **y** along the links followed from **x** to **y** during the exploration, and extend the path π to **y** accordingly.

Once $\delta(\mathbf{x},\mathbf{t}) \leq k\log^2 n$: apply Kleinberg's greedy algorithm, i.e., forward the message to the closest contact of **x** to the target **t**, and extend the path π accordingly, until the target is reached.

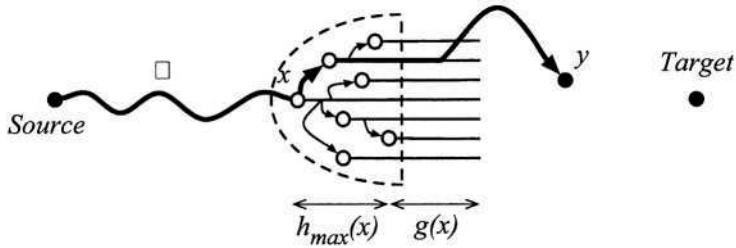


Fig. 1. Extension of the path π (in bold) at the end of an exploration step.

Figure 1 illustrates the structure visited during each exploration step: straight lines represent good local links and arrows represent good long range links; the nodes in F are represented by white circles, each of them starts a new chain of $\leq h_{\max}(x) + g(x)$ local links towards the target. The structure is composed of a (partial) $(1+k)$-ary tree of height $h_{\max}(x)$ extended by chains of local links of length $g(x)$ attached to its leaves. The chains of local links, rooted on the nodes in F, are guaranteed not to overlap, since only good long range contacts far enough from any already present node in F are considered. The tree is drawn on the plane to highlight the tree structure but is in fact mapped on the ring. At the end of the exploration step, the path is extended from $\mathbf{x}$ to the closest[3] node $\mathbf{y}$ to the target, among the explored nodes and their contacts. A new exploration step then begins from $\mathbf{y}$.

The following sections analyze this algorithm in detail and demonstrate the theorem: we start with the one-dimensional network (Section 3) and show in Section 4 how the results on the one-dimensional network extend to arbitrary d-dimensional networks.

3 One-Dimensional Network

In dimension 1, the network is an augmented ring of $2n+1$ nodes, numbered from $-n$ to n. In addition to its two neighbors in the ring (its *local contacts*), each node $\mathbf{u}$ is the origin of k $(\leq \log n)$ extra directed links, each of them pointing towards a node $\mathbf{v}_j$ ($\mathbf{u}$'s j-th long-range contact), chosen independently according to the 1-harmonic distribution, i.e., with probability $1/(2H_n\delta(\mathbf{u},\mathbf{v}))$, where $H_n = \sum_{i=1}^{n} 1/i$. We define a *chain as* a set of locally neighboring nodes, i.e., a path of local links.

In order to simplify the analysis of the algorithm, we use a breadth-first search implementation of the exploration step in our algorithm (below). The analysis consists in the study of the explored tree structure: basically, that this tree is large enough to guarantee the existence of a contact whose lattice distance to the target $\mathbf{t}$ is $\frac{\log(1+k)}{2k}$ times $\mathbf{x}$'s distance to $\mathbf{t}$. Since this analysis is independent of the way the tree is searched, it will apply to the depth-first search implementation as well.

[3] According to the lattice distance.

Routing Algorithm (Breadth-First Search Implementation)

Let $h_{\max}(x) = (\log\log x - \log\log\log n)/\log(1 + kH_x/(6H_n))$,
(*Note that* $h_{\max}(x) = O(\log n \log\log x/(\log(1+k)\log x))$)
$b_{\max}(x) = \log x/\log\log n$,
and $g(x) = \log n \log\log n/(\log(1+k)\log x)$.

Input: the source $\mathbf{s}$ and the target $\mathbf{t}$.

1. **Initialization:** $\mathbf{x} \leftarrow \mathbf{s}$.
2. While $\delta(\mathbf{x},\mathbf{t}) > k\log^2 n$, do:

Exploration step:

$x \leftarrow \delta(\mathbf{x},\mathbf{t})$, $A_0 \leftarrow \{\mathbf{x}\}$, $B_0 \leftarrow \{\mathbf{x}\}$, $F \leftarrow \{\mathbf{x}\}$, $h \leftarrow 0$.
While $h < h_{\max}(x)$ and $|B_h| < b_{\max}(x)$:
$B_{h+1} \leftarrow \varnothing$.
for each $\mathbf{u} \in B_h$ do
$B_{h+1} \leftarrow$ the good local neighbors of $\mathbf{u}$.
for each good long range contact $\mathbf{v}$ of $\mathbf{u}$ do
if $\forall \mathbf{w} \in F$, $\delta(\mathbf{v},\mathbf{w}) \geq h_{\max}(x) + g(x)$ then
$F \leftarrow F \cup \{\mathbf{v}\}$, $B_{h+1} \leftarrow B_{h+1} \cup \{\mathbf{v}\}$.
$A_{h+1} \leftarrow A_h \cup B_{h+1}$.
$h{+}{+}$.
if $|B_h| > b_{\max}(x)$ then
remove the $(|B_h| - b_{\max}(x))$ last inserted nodes from B_h and F.
$h_{\text{stop}} \leftarrow h$, $A \leftarrow A_{h_{\text{stop}}-1} \cup B_{h_{\text{stop}}}$. (*Note that* $|B_{h_{\text{stop}}}| \leq b_{\max}(x)$)
$C \leftarrow \bigcup_{b \in B_{h_{\text{stop}}}} C_b$, where C_b is the set of the nodes that are $\leq g(x)$ local good links away from b.

Message forward step:

$\mathbf{x} \leftarrow$ the closest node to the target $\mathbf{t}$, according to the lattice distance, among the local or long-range contacts of a node in $A \cup C$.
Route the message to $\mathbf{x}$, along the shortest path in $A \cup C$ to $\mathbf{x}$.

3. **Final step (*Kleinberg's greedy algorithm*):** Forward the message to the closest node towards the target among the (local or long-range) contacts of its current holder, until it reaches the target $\mathbf{t}$.

Figure 2 illustrates the notations used in the algorithm described in the frame above. A is the set of the nodes explored. The links followed during any exploration step map a non-overlapping $(1 + k)$-ary tree structure of height $h_{\text{stop}} \leq h_{\max}(x)$ on A, whose set of leaves is $B_{h_{\text{stop}}}$, and extended by $|B_{h_{\text{stop}}}| \leq b_{\max}(x)$ chains of length $g(x)$, rooted on the nodes in $B_{h_{\text{stop}}}$, pointing towards the target. The set B_h is the set of nodes at level h in the tree structure mapped on A. A may as well be seen as a set of $|F| = |B_{h_{\text{stop}}}|$ non-overlapping chains of length $\leq h_{\max}(x) + g(x)$ rooted on the nodes in F, pointing towards the target, connected one to the other by a tree structure of long range links.

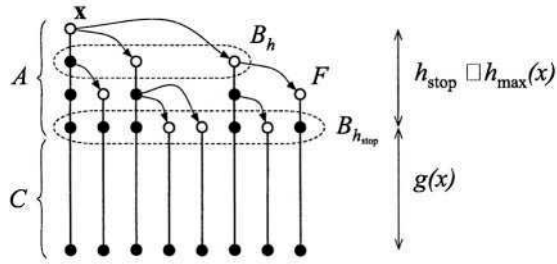


Fig. 2. The sets A, B_h, C and F (the nodes in white) during an exploration step.

Map of the proof. We will show that at the end of any exploration step, with constant probability, the message is routed to a node at lattice distance $\leq \frac{\log(1+k)}{2k}x$ from the target, where x is the lattice distance of the message at the beginning of the exploration step (Proposition 3, from which we deduce Theorem 1). In order to prove Proposition 3, we show that with constant probability, there are at least $\Omega(\frac{\log n}{\log(1+k)})$ nodes in C, whose long range contacts have not yet been explored (Lemma 7). Combined with Lemma 4, this yields Proposition 3. The proof of Lemma 7 consists in showing that the number of branches in the tree structure of A is large enough. This is ensured by lower bounding the probability that a new non-overlapping branch is created (Corollary 6), and then carefully tuning $h_{\max}(x)$ and $g(x)$ to realize a trade-off between limiting overlapping and maximizing the tree growth to minimize its height. The size of the tree is then lower bounded by the growth of a branching process, which concludes the result.

Proposition 3. *There exist two constants $p_1 > 0$ and n_0, independent of n and* **x**, *such that, for $n \geq n_0$, at the end of any exploration step, with probability $\geq p_1$, there is a node* **u** *in $A \cup C$ such that* **u** *or one of its long-range contact is at distance $\leq \frac{\log(1+k)}{2k}x$ from the target.*

The following lemma is directly inspired from [10]; its proof is omitted.

Lemma 4. *Given $\gamma > 0$, there is a constant $p_2 > 0$, such that, for any subset Γ of $\gamma \cdot \frac{\log n}{\log(1+k)}$ vertices at lattice distance in $(\frac{\log(1+k)}{2k}x, x]$ from the target, one vertex in Γ (at least) has a long-range contact at lattice distance $\leq \frac{\log(1+k)}{2k}x$ to the target, with probability at least p_2.*

The next lemma will be used to lower bound the probability of creating a new non-overlapping branch in A.

Lemma 5. *Let* **u** *be a node at lattice distance u from the target* **t**, **v** *its j-th long-range contact, Q a set of q forbidden nodes, and r an integer. The probability that* **v** *is good and at lattice distance $\geq r$ from any node of Q, is $\geq (H_{2u-1} - H_{2rq-1})/(2H_n)$.*

Proof. Let $\mathcal{E}$ be the event that **v** is good and is at distance $\geq r$ from any node of Q. $\mathcal{E}$ is the event that **v** is good and does not belong to the any of the q chains

of nodes of length $2r$ centered on the nodes of Q. We bound the probability of $\mathcal{E}$ by noticing that the probability that $\mathbf{v}$ is at distance δ from u is decreasing with δ. Therefore, the probability of $\mathcal{E}$ is minimized when the nodes in the q chains are all distinct, in the interval of nodes of radius $u-1$ around the target, and as close as possible to $\mathbf{u}$, according to the lattice distance. A simple case analysis (depending on whether $u \leq n/2$, or $n/2 \leq u \leq n - rq$, or $u \geq n - rq$) shows that the probability of $\mathcal{E}$ is then greater than the probability that $\mathbf{v}$ is at distance $\geq 2rq$ from $\mathbf{u}$, and is at distance $< u$ from the target. We conclude that: $\Pr \mathcal{E} \geq \frac{1}{2H_n} \sum_{i=2rq}^{2u-1} \frac{1}{i} = \frac{H_{2u-1}-H_{2rq-1}}{2H_n}$.

Corollary 6. *There exists a constant n_0 independent of x, n, and k, such that if $n \geq n_0$, during any exploration step, for any unvisited node $\mathbf{u}$ at lattice distance $u > \frac{\log(1+k)}{2k}x$ from the target, the probability α_u that the j-th long range contact $\mathbf{v}$ of $\mathbf{u}$, is good and is at lattice distance $\geq h_{\max}(x) + g(x)$ from any node in F, is greater than $H_x/(6H_n) =_{\text{def}} \alpha^-$.*

Proof. F contains less than $b_{\max}(x)$ nodes. By Lemma 5, $\alpha_u \geq (H_{2u-1} - H_{2b_{\max}(x)(h_{\max}(x)+g(x))-1})/2H_n$. But $h_{\max}(x) \leq \frac{6H_n}{H_x}\frac{\log\log x}{\log(1+k)} \leq 6g(x)$ and $b_{\max}(x)g(x) = \log n$, thus: $\alpha_u \geq \frac{H_{2u-1}-H_{16\log n}}{2H_n} \geq \ln(\frac{2u-1}{16\log n})/(2H_n)$. Since $x > k\log^2 n$ and $k \leq \log n$, we have $\frac{x}{k} < x^{1/3}\log n$ and $\ln\left(\frac{2u-1}{16\log n}\right) > \ln\left(\frac{\log(1+k)\cdot x/k-1}{16\log n}\right) > H_{x^{1/3}} \geq \frac{1}{3}H_x$, for $n \geq n_0$, for some constant n_0 independent of x, n, and k. We conclude that $\alpha_u \geq H_x/(6H_n)$.

The following lemma shows that at the end of any exploration step, with constant probability, either we have already reached a node in $A \cup C$ at distance $\leq \frac{\log(1+k)}{2k}x$ from the target, or the tree is wide enough to contain the required number of nodes to apply Lemma 4.

Lemma 7. *There exists a constant $p_3 > 0$, independent of n and x, such that, at the end of any exploration step, with probability at least p_3, either there exists a node in A at lattice distance $\leq \frac{\log(1+k)}{2k}x$ from the target, or $|B_{h_{\text{stop}}}| \geq \log x/(2\log\log n)$.*

Proof. Let $\mathcal{E}$ the event that at the end of the exploration step, there exists a node in A at lattice distance $\leq \frac{\log(1+k)}{2k}x$ from the target, or $|B_{h_{\text{stop}}}| \geq \log x/(2\log\log n)$.

Let $Z = \{\mathbf{z} : \delta(\mathbf{z},\mathbf{t}) > \frac{\log(1+k)}{2k}x\}$ and $\bar{Z}$ its complementary set. By Corollary 6, during any exploration step, for every unvisited node $\mathbf{u} \in Z$, for all $1 \leq j \leq k$, the probability that the j-th long range contact of $\mathbf{u}$ is good and at lattice distance $\geq h_{\max}(x) + g(x)$ of any node in the current F, is at least α^-. Thus, as long as nodes in Z are considered, each of their long range contact will be added to B_{h+1} with probability $\geq \alpha^-$. As soon as a node $\mathbf{u}$ from $\bar{Z}$ is inserted in A_h, for some h, the probability that, for a given j, its j-th long range contact is good and at lattice distance $\geq h_{\max}(x) + g(x)$ of any node in the current F, is no longer lower bounded by α^-; but the event $\mathcal{E}$ is verified. We use a probabilistic coupling argument to lower bound the probability of $\mathcal{E}$,

by virtually running the exploration step on a gadget network, constructed from the original network as follows: this gadget network has the same underlying lattice; the nodes in Z have the exact same links as in the original network; but we consider a virtual link distribution for the nodes of $\bar{Z}$ such that for every unvisited node $\mathbf{u}$, the probability that its j-th long range contact is good and at lattice distance $\geq h_{\max}(x) + g(x)$ from any set of nodes G of size $\leq b_{\max}(x)$, is α^- (note that this distribution does not need to exist effectively). We run the exploration step on this gadget network from the same $\mathbf{x}$ as in the real network, except that we don't interrupt it until $h = h_{\max}(x)$. It yields three sets families (A'_h), (B'_h) and F', such that: $A_h \cap Z = A'_h \cap Z$, $B_h \cap Z = B'_h \cap Z$, and $F \cap Z = F' \cap Z$, for all $1 \leq h \leq h_{\text{stop}}$. The links followed during the exploration of the gadget network define a non-overlapping tree structure of height exactly $h_{\max}(x)$ on $A' = \cup_h A'_h$ where B'_h is the set of the nodes at level h. Let $\mathcal{E}'$ be the event that $|B'_{h_{\max}(x)}| \geq \log x/(2 \log\log n)$. We now show that $\Pr\{\mathcal{E}\} \geq \Pr\{\mathcal{E}'\}$:

- If, in the original network, $A \cap \bar{Z} = \varnothing$, then $B'_{h_{\text{stop}}} = B_{h_{\text{stop}}}$. If $h_{\text{stop}} < h_{\max}(x)$, then $|B'_{h_{\max}(x)}| \geq |B'_{h_{\text{stop}}}| = |B_{h_{\text{stop}}}| = b_{\max}(x) = \log x/\log\log n$, and then $\mathcal{E}$ and $\mathcal{E}'$ are both verified. If $h_{\text{stop}} = h_{\max}$, $B_{h_{\text{stop}}} = B'_{h_{\max}(x)}$ and then $\mathcal{E}$ and $\mathcal{E}'$ are equivalent. Then, whatever the gadget network is inside $\bar{Z}$, $\Pr\{\mathcal{E}|A \cap \bar{Z} = \varnothing\} = \Pr\{\mathcal{E}'|A \cap \bar{Z} = \varnothing\}$.
- If, in the original network, $A \cap \bar{Z} \neq \varnothing$, then $\mathcal{E}$ is verified, so, whatever the gadget network is inside $\bar{Z}$, $\Pr\{\mathcal{E}|A \cap \bar{Z} \neq \varnothing\} = 1 \geq \Pr\{\mathcal{E}'|A \cap \bar{Z} \neq \varnothing\}$.

We now lower bound $\Pr\{\mathcal{E}'\}$. The set $A' = \cup_h A'_h$ is structured as a random tree of root $\mathbf{x}$, in which every node $\mathbf{u}$ at level h has, independently, a random number $1 + l$ of children (one local contact and l long range contacts), where l is given by a binomial law of parameters $(k, \alpha_{\mathbf{u}})$, with $\alpha_{\mathbf{u}} \geq \alpha^-$. Thus the number of nodes at level h, $|B'_h|$, stochastically dominates the random variable[4] b_h for the number of nodes at level h in the following branching process: start with one node; at step h, each node at level $h-1$ is given, independently, exactly $1 + l$ children, with probability $\rho_l = \binom{k}{l}(\alpha^-)^l(1-\alpha^-)^{k-l}$, where $0 \leq l \leq k$. Bounding the variance of b_h (omitted), gives: $\mathbb{E}[b_h] = (1 + k\alpha^-)^h$ and a constant $p_3 > 0$, independent of α^- and h, such that, with probability at least p_3, $b_h \geq \mathbb{E}[b_h]/2$. Then, since $(1 + k\alpha^-)^{h_{\max}(x)} = \log x/\log\log n$, $\Pr\{\mathcal{E}\} \geq \Pr\{\mathcal{E}'\} = \Pr\{|B'_{h_{\max}(x)}| \geq \log x/(2\log\log n)\} \geq p_3$.

Corollary 8. *For $n \geq n_0$, at the end of any exploration step, with probability at least p_3, there is a node in A at lattice distance $\leq \frac{\log(1+k)}{2k} x$ from the target or there are more than $\frac{\log n}{2\log(1+k)}$ distinct nodes in C (where p_3 is given by Lemma 7).*

Combined with Lemma 4, Corollary 8 yields Proposition 3.

Proof. (of Theorem 1) W.l.o.g., the target is $\mathbf{0}$ and the source $\mathbf{s}$ is at lattice distance s from $\mathbf{0}$. Let $\mathbf{x}$ denote the current message holder of the message and x its lattice distance from the target. First recall that at the end of each exploration

[4] i.e., for all z, $\Pr\{|B'_h| \geq z\} \geq \Pr\{b_h \geq z\}$.

step, the algorithm selects the closest node to the target among the local and long-range contacts of $A \cup C$, and that the set $A \cup C$ grows towards the target; therefore, every exploration step visits unexplored nodes, and each exploration step is independent of the previous ones.

Let T and U be the solutions to $(\frac{2k}{\log(1+k)})^T = s$ and $(\frac{2k}{\log(1+k)})^U = k\log^2 n$. Note that $T \sim \log s/\log(1+k)$ and $U \sim (2\log\log n + \log k)/\log(1+k)$. We decompose the execution of $\mathcal{A}$ in T phases. The execution is in phase i, $0 \leq i \leq T$, as long as $(\frac{2k}{\log(1+k)})^{i-1} < x \leq (\frac{2k}{\log(1+k)})^i$. We say that an exploration step in phase i *succeeds* if it leads to a phase $\leq i-1$. Let Y_i and Z_i be respectively the random variables for the number of visited nodes in phase i, and for the length of the path along which the message is routed in phase i.

Suppose that we are in phase i, with $T \geq i > U$, then $x > k\log^2 n$. According to Proposition 3, each exploration step succeeds with probability $\geq p_1$. Each exploration step visits $\leq (h_{\max}(x) + g(x))\ b_{\max}(x) \leq 7g(x)b_{\max}(x)$ nodes, and routes the message, along a path of length $\leq h_{\max}(x) + g(x) \leq 7g(x)$ towards the target. Then, $\mathbb{E}[Y_i] \leq 7g(x)b_{\max}(x)/p_1 \leq \frac{7}{p_1}\frac{\log n}{\log(1+k)}$ and $\mathbb{E}[Z_i] \leq 7g(x)/p_1 \leq \frac{7}{p_1}\frac{\log n \log\log n}{i\log^2(1+k)}$, since $\log x \geq i\log(1+k)$.

Once we reach a phase $i \leq U$, we have $x \leq k\log^2 n$ and the algorithm runs Kleinberg's greedy algorithm. From [10], we know that this greedy computes a path of expected length $\leq A(\log n \log x)/k \leq 3A(\log n \log\log n)/k$ while visiting $\leq 3A(\log n \log\log n)/k$ nodes on expectation, for some constant A.

The expected length of the path from $\mathbf{s}$ to $\mathbf{0}$ computed by our algorithm is bounded by:

$$\sum_{i=0}^{T} \mathbb{E}[Z_i] \leq 3A\,\frac{\log n \log\log n}{k} + \frac{7}{p_1}\frac{\log n \log\log n}{\log^2(1+k)} \sum_{U<i\leq T} \frac{1}{i}$$
$$= O\left(\log n \left(\frac{\log\log n}{\log(1+k)}\right)^2\right).$$

And the expected number of nodes visited by our algorithm is bounded by:

$$\sum_{i=0}^{T} \mathbb{E}[Y_i] = O\left(\left(\frac{\log n}{\log(1+k)}\right)^2\right).$$

For the last of each exploration step, our algorithm just needs $\Theta\big(\log n \cdot (b_{\max}(x) + h_{\max}(x) + g(x))\big) = O(\log^2 n/\log(1+k))$ bits of memory. Indeed, each node address requires $\log n$ bits, and each exploration step needs only to store: the address of the target, the address of the nodes in F (whose size is $\leq b_{\max}(x) = O(\log n/\log\log n) = O(\log n/\log(1+k))$), the state of the stack during the depth-first search of $A \cup C$ (whose height is bounded by $h_{\max}(x) + g(x) = O(\log n/\log(1+k))$), and both the address and the state of the stack for the current best node $\mathbf{y}$ among $A \cup C$ and $A \cup C$'s contacts.

4 d-Dimensional Network

In a d-dimensional network, $d > 1$, the underlying lattice is a d-dimensional torus $\{-n, \dots, n\}^d$. Each node $\mathbf{u}$ has k extra directed links (its long range links) each one pointing towards a node $\mathbf{v}$ chosen independently according to the d-harmonic distribution, i.e., with probability proportional to $1/\delta(\mathbf{u}, \mathbf{v})^d$.

We denote by $\mathcal{S}(\mathbf{u}, r)$ and $\mathcal{B}(\mathbf{u}, r)$, respectively the ℓ_1-sphere and ℓ_1-ball centered on $\mathbf{u}$ and of radius r. We denote by $S(r)$ and $V(r)$ their respective cardinality. Clearly, for $r \leq n$, $S(r) = \Theta(r^{d-1})$ and $V(r) = \Theta(r^d)$. More precisely, for $r \leq n$, $S(r) = \frac{2^d}{(d-1)!} r^{d-1} + \xi(r)$, and $V(r) = \frac{2^d}{d!} r^d + \eta(r)$, where $\xi(r)$ and $\eta(r)$ are positive polynomials of respective degree $d-2$ and $d-1$. These expressions are upper bounds on $S(r)$ and $V(r)$ when $r > n$.

The algorithm on a d-dimensional network. We only need to adapt the parameters of the one-dimensional routing algorithm, as follows, and everything else in the algorithm is unchanged: $b_{\max}(x)$ and $g(x)$ are unchanged; $h_{\max}(x)$ is now set to $h_{\max}(x) = (\log\log x - \log\log\log n)/\log(1 + \frac{kH_x}{2^d(2^d+2)H_n})$; and, the exploration phases now stop as soon as $x \leq k \log^{2^d+1} n$ (the while condition Item 2), and then the algorithm runs Kleinberg's greedy algorithm.

Sketch of the analysis of the algorithm on a d-network. The analysis of the algorithm is exactly identical to the one-dimensional case. Only the lower bound on the probability of creating a new branch, α^-, in the tree structure $A \cup C$ has to be evaluated in order to get the result. Lemma 10 shows that for our choice of the parameters $b_{\max}(x)$, $g(x)$, $h_{\max}(x)$ and the while condition in Item 2, this probability is again $\Theta(H_x/H_n)$, as in dimension 1, from which we get the result.

The next lemmas correspond to Lemma 5 and Corollary 6 in dimension 1; their proofs rely on the geometry of the balls in dimension d, and are omitted.

Lemma 9. *Let $1 \leq j \leq k$, $\mathbf{u}$ a node at distance $u \in (\frac{\log(1+k)}{2k} x, x]$ from the target, $\mathbf{v}$ its j-th long-range contact, Q a set of q forbidden nodes, and r an integer. The probability that $\mathbf{v}$ is good and at distance $\geq r$ from any node in Q, is $\geq \frac{H_u - 2^d H_{rq^{1/d}} - c_1}{2^d H_n + c_2}$, for two constants $c_1, c_2 \geq 0$, that only depend on d.*

Lemma 10. *There exists a constant n_1, independent of x, n, and k, such that if $n \geq n_1$, during any exploration step, for any unvisited node $\mathbf{u}$ at lattice distance $u > \frac{\log(1+k)}{2k} x$ from the target, the probability $\alpha_{u,d}$ that the j-th long range contact $\mathbf{v}$ of $\mathbf{u}$, is good and is at lattice distance $\geq h_{\max}(x) + g(x)$ from any node in F, is greater than $\frac{H_x}{2^d(2^d+2)H_n} =_{\text{def}} \alpha_d^-$.*

The properties of the tree structure on $A \cup C$ are then similar to dimension 1. Lemma 7, Property 3 and Theorem 1 follow then for any dimension $d \geq 1$.

Conclusion. Our algorithm could possibly have interesting applications in peer-to-peer networks, since its latency is comparable to Kleinberg's greedy algorithm and since it computes almost optimal paths based only on local information. Note that, if we get a bound on the expected path length between random pair of nodes, the question of the exact Kleinberg's network diameter remains open.

References

1. Manku, G.S., Naor, M., Wieder, U.: Know thy neighbor's neighbor: The power of lookahead in randomized P2P networks. To appear in Proc. of 36th ACM STOC 2004 (2004)
2. Milgram, S.: The small world problem. Psychology Today **61** (1967)
3. Newman, M.E.J.: Models of the small world. J. Stat. Phys. **101** (2000)
4. Newman, M.E.J.: The structure and function of complex networks. SIAM Review **45** (2003) 167–256
5. Zhang, H., Goel, A., Govindan, R.: Using the small-world model to improve Freenet performance. Proceedings of IEEE INFOCOM (2002)
6. Aspnes, J., Diamadia, Z., Shah, G.: Fault-tolerant routing in peer-to-peer systems. In: Proc. of ACM 3st Symp. on Princ. of Distr. Comp. (PODC 2002). Volume 31. (2002) 223–232
7. Watts, D., Strogatz, S.: Collective dynamics of small-world networks. Nature **393** (1998)
8. Newman, M.E.J., Moore, C., Watts, D.J.: Mean-field solution of the small-world network model. Phys. Rev. Lett. **84** (2000) 3201–3204
9. Bollobás, B., Chung, F.: The diameter of a cycle plus random matching. SIAM J. Discrete Math. **1** (1988) 328–333
10. Kleinberg, J.: The small-world phenomenon: an algorithmic perspective. Proc. 32nd ACM Symposium on Theory of Computing (2000) 163–170
11. Barrière, L., Fraigniaud, P., Kranakis, E., Krizanc, D.: Efficient routing in networks with long range contacts. LNCS Proc. of 15th International Symposium on Distributed Computing (DISC '01) **2180** (2001) 270–284
12. Benjamini, I., Berger, N.: The diameter of long-range percolation clusters on finite cycles. Random Structures and Algorithms **19** (2001) 102–111
13. Coppersmith, D., Gamarnik, D., Sviridenko, M.: The diameter of a long range percolation graph. Random Structures and Algorithms **21** (2002) 1–13
14. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. In: Proc. of ACM SIGCOMM 2001. (2001)
15. Manku, G.S., Bawa, M., Raghavan, P.: Symphony: Distributed hashing in a small world. In: Proc. of 4th Usenix Symp. on Internet tech. and syst. (2003)
16. Fraigniaud, P., Gavoille, C., Paul, C.: Eclecticism shrinks the world. Technical Report LRI-1376, University Paris-Sud (2003)
17. Kleinberg, J.: Small-world phenomena and the dynamics of information. Advances in Neural Information Processing Systems, MIT Press. **14** (2002)

Word Problems on Compressed Words

Markus Lohrey

Universität Stuttgart, FMI, Universitätsstr. 38, 70569 Stuttgart, Germany
lohrey@fmi.uni-stuttgart.de

Abstract. We consider a compressed form of the word problem for finitely presented monoids, where the input consists of two compressed representations of words over the generators of a monoid $\mathcal{M}$, and we ask whether these two words represent the same monoid element of $\mathcal{M}$. For compression we use straight-line programs. For several classes of monoids we obtain completeness results for complexity classes in the range from P to EXPSPACE. As a by-product of our results on compressed word problems we obtain a fixed deterministic context-free language with a PSPACE-complete membership problem. The existence of such a language was open so far. Finally, we investigate the complexity of the compressed membership problem for various circuit complexity classes.

1 Introduction

During the last decade, the massive increase in the volume of data has motivated the investigation of algorithms on *compressed data,* like for instance compressed strings, trees, or pictures. The general goal is to develop algorithms that directly work on compressed data without prior decompression. Let us mention here the work on compressed pattern matching, see, e.g., [21]. In this paper we investigate two classes of computational problems on compressed data that are of central importance in theoretical computer science since its very beginning: the *word problem* and the *membership problem.* In its most general form, the word problem asks whether two terms over an algebraic structure represent the same element of the structure. Here, we consider the word problem for *finitely presented monoids,* which are given by finite sets of generators and defining relations. In this case the input consists of two finite words over the set of generators and we ask whether these two words represent the same monoid element. The undecidability results concerning the word problem for finitely presented monoids/groups are among the first undecidability results that touched "real mathematics", see [23] for references. Moreover, these negative results motivated a still ongoing investigation of decidable subclasses of word problems and their complexity. In particular, monoids that can be presented by terminating and confluent semi-Thue systems (i.e., string rewriting systems) received a lot of attention [8]. These monoids have decidable word problems, and sharp complexity bounds are known for various subclasses [7,15,16].

In its compressed variant, the input to the word problem for a finitely presented monoid consists of two compressed representations of words over the generators. We choose *straight-line programs,* or equivalently context-free grammars

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 906–918, 2004.

that generate exactly one word, for compression. Straight-line programs turned out to be a very flexible compressed representation of strings. Several other compressed representations, like for instance Lempel-Ziv factorizations [28], can be efficiently converted into straight-line programs and vice versa [19], which implies that most of our complexity results hold for Lempel-Ziv factorizations as well. Moreover, by using straight-line programs for representing inputs, the compressed word problem becomes equivalent to the circuit equivalence problem (a generalization of the well-known circuit evaluation problem), where we ask whether two circuits over a finitely presented monoid $\mathcal{M}$ (i.e., acyclic directed graphs with leafs labeled by generators of $\mathcal{M}$ and internal nodes labeled by the monoid operation) evaluate to the same element of $\mathcal{M}$. So far this problem was only investigated for finite monoids [5]. In Section 3–5 we study the complexity of compressed word problems for several subclasses of monoids presented by terminating and confluent semi-Thue systems. We obtain completeness results for various complexity classes between P and EXPSPACE. The general phenomenon that we observe when moving from the (uncompressed) word problem to its compressed variant is an exponential jump with respect to complexity. This exponential jump is well known also from other work on the complexity of succinct problems [12,25,26].

As a by-product of our investigation of compressed word problems we obtain several new results concerning *compressed membership problems*. Here, the problem is to decide for a fixed language L, whether a given compressed representation of a word belongs to L [19]. We show that there exists a deterministic context-free (even deterministic linear) language with a PSPACE-complete compressed membership problem, which solves an open problem from [9,19]. This result is also interesting in light of recent attempts to use straight-line programs for compressing control flow traces of procedural programming languages [27]. At a certain level of abstraction, the set of all valid control flow traces is a context-free language. We also present a context-sensitive language with an EXPSPACE-complete compressed membership problem. Finally, in Section 6 we investigate the complexity of the compressed membership problem for various circuit complexity classes. We show that the levels of the logtime hierarchy [22] correspond in a compressed setting to the levels of the polynomial time hierarchy. A full version of this paper can be obtained from the author.

2 Preliminaries

We assume that the reader has some basic background in complexity theory [17]. The reflexive and transitive closure of a binary relation $\to$ is $\xrightarrow{*}$. Let Γ be a finite alphabet. The *empty word* over Γ is denoted by ε. For a word $s = a_1 a_2 \cdots a_n \in \Gamma^*$ $(a_i \in \Gamma)$ let $w^{\mathrm{rev}} = a_n a_{n-1} \cdots a_1$, $\mathrm{alph}(s) = \{a_1, \ldots, a_n\}$, $|s| = n$, $|s|_a = |\{i \mid a_i = a\}|$ (for $a \in \Gamma$), $s[i] = a_i$ (for $1 \le i \le n$), and $s[i,j] = a_i a_{i+1} \cdots a_j$ (for $1 \le i \le j \le n$). If $i > j$ we set $s[i,j] = \varepsilon$. An *involution* $\overline{\ }$ on Γ is a function $\overline{\ } : \Gamma \to \Gamma$ with $\overline{\overline{a}} = a$ for all $a \in \Gamma$. It can be extended to an involution on Γ^* by setting $\overline{a_1 \cdots a_n} = \overline{a_n} \cdots \overline{a_1}$. With $\overline{\Gamma} = \{\overline{a} \mid a \in \Gamma\}$

we always denote a disjoint copy of the alphabet Γ. Then we can define an involution $\overline{\ }$ on $\Delta = \Gamma \cup \overline{\Gamma}$ by setting $\overline{\overline{a}} = a$; this involution will be extended to Δ^* in the above way. A *weight-function* is a homomorphism $f : \Gamma^* \to \mathbb{N}$ from the free monoid Γ^* to the natural numbers (with +) such that $f^{-1}(0) = \{\varepsilon\}$. Given a linear oder $\succ$ on the alphabet Γ, we extend $\succ$ to a linear order on Γ^*, called the *lexicographic extension of* $\succ$, as follows: $u \succ v$ if v is a prefix of u or $u = wau'$ and $v = wbv'$ with $a, b \in \Gamma$ and $a \succ b$.

Semi-Thue systems and finitely presented monoids. For more details and references on the topic of this section see [8]. Let Γ be a finite alphabet. A *semi-Thue system* R over Γ is a finite subset $R \subseteq \Gamma^* \times \Gamma^*$; its elements are called rules. A rule $(s,t) \in R$ is also written as $s \to t$. The pair (Γ, R) is a *presentation.* Let $\mathrm{dom}(R) = \{s \mid \exists t : (s,t) \in R\}$. We define the binary relation $\to_R$ on Γ^* as follows: $s \to_R t$ if there exist $u, v \in \Gamma^*$ and $(\ell, r) \in R$ with $s = u\ell v$ and $t = urv$. Moreover, let ${}_R\leftarrow = (\to_R)^{-1}$, $\leftrightarrow_R = (\to_R \cup {}_R\leftarrow)$, and $\mathrm{IRR}(R) = \Gamma^* \backslash \Gamma^* \mathrm{dom}(R) \Gamma^*$ (the set of *irreducible words*). We say that (Γ, R) is *terminating* if there do not exist $s_i \in \Gamma^*$ for $i \in \mathbb{N}$ with $s_i \to_R s_{i+1}$ for all $i \in \mathbb{N}$. We say that (Γ, R) is *confluent* (resp. *locally confluent*) if for all $s, t, u \in \Gamma^*$ with $t \,{}_R\overset{*}{\leftarrow} s \overset{*}{\to}_R u$ (resp. $t \,{}_R\leftarrow s \to_R u$) there exists $v \in \Gamma^*$ such that $t \overset{*}{\to}_R v \,{}_R\overset{*}{\leftarrow} u$. By Newman's lemma, a terminating presentation is confluent if and only if it is locally confluent. Moreover, for a terminating presentation, local confluence (and hence confluence) can be checked effectively using *critical pairs,* which result from overlapping left-hand sides. The reflexive and transitive closure $\overset{*}{\leftrightarrow}_R$ is a congruence on the free monoid Γ^*, hence we can define the quotient monoid $\Gamma^* / \overset{*}{\leftrightarrow}_R$, which we denote by $\mathcal{M}(\Gamma, R)$. It is called a *finitely presented monoid,* and we say that $\mathcal{M}(\Gamma, R)$ is the *monoid presented by* (Γ, R). The *word problem* for the fixed presentation (Γ, R) is the following decision problem:

INPUT: Two words $s, t \in \Gamma^*$.
QUESTION: Does $s \overset{*}{\leftrightarrow}_R t$ hold?

It is easy to see that for two given presentations (Γ, R) and (Σ, S) such that $\mathcal{M}(\Gamma, R) \cong \mathcal{M} \cong \mathcal{M}(\Sigma, S)$, there exists a logspace reduction from the word problem for (Γ, R) to the word problem for (Σ, S). Thus, the decidability and complexity of the word problem do not depend on the chosen presentation and we may just speak of the word problem for the monoid $\mathcal{M}$.

If (Γ, R) is terminating and confluent, then every $s \in \Gamma^*$ has a unique *normal form* $\mathrm{NF}_R(s) \in \mathrm{IRR}(R)$ satisfying $s \overset{*}{\to}_R \mathrm{NF}_R(s)$. Moreover, $s \overset{*}{\leftrightarrow}_R t$ if and only if $\mathrm{NF}_R(s) = \mathrm{NF}_R(t)$. Thus, the word problem is decidable. On the other hand, the calculation of normal forms does not yield any upper bound on the complexity of the word problem [3]. Complexity results on word problems for restricted classes of finitely presented monoids can be found for instance in [7,15,16].

Grammar based compression. Following [19], a *straight-line program (SLP) over the alphabet* Γ is a context-free grammar $G = (V, \Gamma, S, P)$, where V is the set of nonterminals, Γ is the set of terminals, $S \in V$ is the initial nonterminal, and $P \subseteq V \times (V \cup \Gamma)^*$ is the set of productions, such that (i) for every $X \in V$

there is exactly one $\alpha \in (V \cup \Gamma)^*$ with $(X, \alpha) \in P$ and (ii) there is no cycle in the relation $\{(X, Y) \in V \times V \mid \exists \alpha : (X, \alpha) \in P, Y \in \mathrm{alph}(\alpha)\}$.[1] The language generated by the SLP G contains exactly one word that is denoted by $\mathrm{eval}(G)$. More generally, every nonterminal $X \in V$ produces exactly one word that is denoted by $\mathrm{eval}_G(X)$. We omit the index G if the underlying SLP is clear from the context. We also write $P(G)$ for the set of productions P. The size of G is $|G| = \sum_{(X,\alpha)\in P} |\alpha|$. Every SLP can be transformed in polynomial time into an equivalent SLP that is in Chomsky normal form (as a context-free grammar). We may also allow exponential expressions of the form A^i for $A \in V$ and a binary coded integer $i \in \mathbb{N}$ in the right-hand sides of productions. Such a production can be replaced by $O(\log(i))$ many ordinary productions. The following tasks can be solved in polynomial time; the first two problems can be reduced to simple arithmetic, whereas the third problem requires more subtle techniques:

- Given a SLP G, calculate $|\mathrm{eval}(G)|$.
- Given a SLP G and a number $i \in \{1, \ldots, |\mathrm{eval}(G)|\}$, calculate $\mathrm{eval}(G)[i]$.
- Given SLPs G_1 and G_2, decide whether $\mathrm{eval}(G_1) = \mathrm{eval}(G_2)$ [18].

Let (Γ, R) be a fixed presentation. The *compressed word problem* for (Γ, R) is the following problem:

INPUT: Two SLPs G_1 and G_2 over the terminal alphabet Γ.
QUESTION: Does $\mathrm{eval}(G_1) \overset{*}{\leftrightarrow}_R \mathrm{eval}(G_2)$ hold?

Here, the input size is $|G_1| + |G_2|$. It is easy to see that also for the compressed word problem the complexity does not depend on the chosen presentation, which allows to speak of the compressed word problem for the monoid $\mathcal{M} = \mathcal{M}(\Gamma, R)$. We can view the compressed word problem also from another perspective. A *circuit* $\mathcal{C}$ *over* $\mathcal{M}$ is a finite directed acyclic graph with exactly one node of outdegree 0. The nodes of indegree 0 are labeled with elements from Γ. All nodes of indegree greater than zero are labeled with the multiplication of $\mathcal{M}$. Such a circuit computes in a natural way an element of $\mathcal{M}$. Then, the question, whether two given circuits over $\mathcal{M}$ compute the same monoid element, is equivalent to the compressed word problem for $\mathcal{M}$. In [5], it was shown that for a finite non-solvable monoid the compressed word problem is P-complete, whereas for every finite solvable monoid the compressed word problem belongs to $\mathrm{DET} \subseteq \mathrm{NC}^2 \subseteq \mathrm{P}$. Our work can be seen as a first step towards extending the work from [5] to infinite monoids.

For a given language $L \subseteq \Gamma^*$ we also consider the *compressed membership problem* for the language L, which is the following problem:

INPUT: A SLP G over the terminal alphabet Γ.
QUESTION: Does $\mathrm{eval}(G) \in L$ hold?

Most of our complexity results can be also transfered to other compression schemes, like for instance Lempel-Ziv 77 (LZ77) [28]. If G is a SLP of size n with $\mathrm{eval}(G) = w$, then $\mathrm{LZ}(w)$ (the LZ77-compressed representation of w) has

[1] Usually, the term "straight-line program" is used in order to denote a linear sequence of instructions. In our context, the only instruction is the concatenation of words.

size $O(n)$ and can be constructed in polynomial time [19]. On the other hand, if n is the size of $\mathrm{LZ}(w)$, then we can construct in polynomial time a SLP of size $O(n^2 \cdot \log(n))$ generating w [19]. Thus, if we allow polynomial time reductions, the completeness results from Section 4-6 also hold, if we use LZ77 for compression. P-hardness results cannot be transfered directly, because the transformation from a SLP to the LZ77-compressed representation might be P-hard.

3 Polynomial Time Cases

It is obvious that for every finite monoid the compressed word problem belongs to P. In this section we present a class of infinite monoids with polynomial time solvable compressed word problems. This class contains all free groups. In fact, it turns out that for every non-abelian free group the compressed word problem is P-complete.

A presentation (Γ, R) is 2-homogeneous if for every $(\ell, r) \in R$: $|\ell| = 2$ and $r = \varepsilon$ [6]. In [16] it was shown that for every 2-homogeneous presentation the word problem is in logspace. Moreover, the uniform variant of the word problem for 2-homogeneous presentations, where the presentation is part of the input, is complete for symmetric logspace [16]. The following result was shown in [6]:

Proposition 1. *For every 2-homogeneous presentation (Γ, R) there exists a 2-homogeneous and confluent presentation (Σ, S) with $\mathcal{M}(\Gamma, R) \cong \mathcal{M}(\Sigma, S)$.*

For the further consideration let us fix a 2-homogeneous presentation (Γ, R). By Prop. 1 we may assume that (Γ, R) is confluent. Then we have:

Lemma 1 (cf. [16]). *There exist pairwise disjoint sets $\Sigma_\ell, \Sigma_r, \Delta \subseteq \Gamma$, an involution $\bar{\ } : \Delta \to \Delta$, and a semi-Thue system $S \subseteq \{(ab, \varepsilon) \mid a \in \Sigma_\ell,\ b \in \Sigma_r\}$ such that $\Gamma = \Sigma_\ell \cup \Sigma_r \cup \Delta$ and $R = S \cup \{(a\bar{a}, \varepsilon) \mid a \in \Delta\}$.*

We say that (Γ, R) is *N-free,* if $a, b \in \Sigma_\ell$, $c, d \in \Sigma_r$ (where Σ_ℓ and Σ_r result from the previous lemma), and ac, ad, $bc \in \mathrm{dom}(R)$ imply $bd \in \mathrm{dom}(R)$.

Theorem 1. *If (Γ, R) is 2-homogeneous, confluent, and N-free, then the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in P.*

In the next section we will see that Thm. 1 cannot be extended to the non-*N*-free case unless P = NP. For the proof of Thm. 1 we need a generalization of straight-line programs from [9]: A *composition system* $G = (V, \Gamma, S, P)$ is defined analogously to a SLP, but in addition to ordinary productions it may also contain productions of the form $A \to B[i, j]$ for $B \in V$ and $i, j \in \mathbb{N}$. For such a production we define $\mathrm{eval}_G(A) = \mathrm{eval}_G(B)[i, j]$.[2] As for SLPs we define $\mathrm{eval}(G) = \mathrm{eval}_G(S)$. In [9] it was shown that for two given composition systems G_1 and G_2, the equality $\mathrm{eval}(G_1) = \mathrm{eval}(G_2)$ can be verified in polynomial time, which generalizes the corresponding result for SLPs from [18]. The proof of Thm. 1 is based on:

[2] In [9], only productions of the form $A \to B[j, |\mathrm{eval}_G(B)|]C[1, i]$ are allowed. But this definition is easily seen to be equivalent to our formalism.

Lemma 2. *Assume that (Γ, R) is 2-homogeneous, confluent, and N-free. Then the following problem belongs to P:*

INPUT: Composition systems G_1 and G_2 with $\mathrm{eval}(G_1), \mathrm{eval}(G_2) \in \mathrm{IRR}(R)$.
QUESTION: Does $\mathrm{eval}(G_1)\mathrm{eval}(G_2) \xrightarrow{*}_R \varepsilon$ hold?

Proof of Thm. 1. Let (Γ, R) be 2-homogeneous, confluent, and N-free. Given SLPs G_1 and G_2 over the terminal alphabet Γ, we have to verify in polynomial time, whether $\mathrm{NF}_R(\mathrm{eval}(G_1)) = \mathrm{NF}_R(\mathrm{eval}(G_2))$. Using the result of [9] mentioned before, it suffices to prove that given a SLP G in Chomsky normal form over the terminal alphabet Γ, we can construct in polynomial time a composition system H such that $\mathrm{eval}(H) = \mathrm{NF}_R(\mathrm{eval}(G))$. We construct H inductively by adding more and more rules. Initially, *P(H)* contains all rules from *P(G)* of the form $A \to a$ with $a \in \Gamma$. Now assume that $A \to BC$ belongs to *P(G)* and that H already contains enough rules such that $\mathrm{eval}_H(B) = \mathrm{NF}_R(\mathrm{eval}_G(B))$ and $\mathrm{eval}_H(C) = \mathrm{NF}_R(\mathrm{eval}_G(C))$. If i is the largest number such that

$$\mathrm{eval}_H(B) = u_1u_2, \quad \mathrm{eval}_H(C) = v_1v_2, \quad |u_2| = |v_1| = i, \quad u_2v_1 \xrightarrow{*}_R \varepsilon, \qquad (1)$$

then clearly $\mathrm{NF}_R(\mathrm{eval}_G(A)) = u_1v_2$. For a given $i \in \mathbb{N}$, we can check (1) in polynomial time by Lemma 2. Since i is bounded exponentially in the input size, the largest i satisfying (1) can be easily calculated in polynomial time by doing a binary search. For this largest i we add to the current H the production $A \to B[1, |\mathrm{eval}_H(B)| - i]C[i+1, |\mathrm{eval}_H(C)|]$. □

For Γ an alphabet, the monoid $F(\Gamma) = \mathcal{M}(\Gamma \cup \overline{\Gamma}, \{(c\bar{c}, \varepsilon) \mid c \in \Gamma \cup \overline{\Gamma}\})$ is a group, namely the *free group* generated by Γ. In case $|\Gamma| = n$ we also write F_n for $F(\Gamma)$. It is known that the (uncompressed) word problem for a free group is in logspace [14]. Moreover, the word problem for F_2 is hard for uniform NC^1 [20]. By Thm. 1, the compressed word problem for every free group is in P. By a reduction from the monotone circuit value problem we can prove:

Theorem 2. *The compressed word problem for F_2 is P-complete.*

4 Between P and PSPACE

P^{NP} is the class of all languages that can be accepted by a deterministic polynomial time machine that has additional access to an NP-oracle; it is contained in PSPACE. Several complete problems for P^{NP} can be found in [11].

Theorem 3. *If (Γ, R) is 2-homogeneous and confluent (but not necessarily N-free), then the compressed word problem for $\mathcal{M}(\Gamma, R)$ is in P^{NP}.*

Proof. The key observation is that for a 2-homogeneous and confluent (but not necessarily N-free) presentation (Γ, R), the problem from Lemma 2 is in coNP: If $u_i = \mathrm{eval}(G_i)$ $(i = 1, 2)$ with $u_1, u_2 \in \mathrm{IRR}(R)$, then $u_1u_2 \xrightarrow{*}_R \varepsilon$ if and only if $|u_1| = |u_2| = n$ and $u_1[i]u_2[n - i + 1] \in \mathrm{dom}(R)$ for every $1 \le i \le n$. For a single i, the latter condition can be easily checked in polynomial time. Now the decision procedure from the proof of Thm. 1 in the previous section gives us a $\mathrm{P}^{\mathrm{coNP}}$-, i.e., P^{NP}-algorithm in the present situation. □

By a reduction from the complementary problem of SUBSETSUM, we can show:

Theorem 4. *Let* $\Gamma = \{a, b, c, d\}$ *and* $R = \{(ac, \varepsilon), (ad, \varepsilon), (bc, \varepsilon)\}$ *The compressed word problem for* $\mathcal{M}(\Gamma, R)$ *is coNP-hard.*

The precise complexity of the compressed word problem for 2-homogeneous, confluent, but non-N-free presentations remains open; it is located somewhere between coNP and P^{NP}.

5 Polynomial Space and Above

Our PSPACE upper bounds rely all on the following simple fact:

Proposition 2. *If the membership problem for the language* L *(the word problem for a finitely presented monoid* $\mathcal{M}$*) belongs to* $\bigcup_{c>0} NSPACE(\log^c(n))$*, then the compressed membership problem for* L *(the compressed word problem for* $\mathcal{M}$*) belongs to PSPACE.*

A presentation (Γ, R) is *weight-reducing* if there is a weight-function f on Γ^* with $f(s) > f(t)$ for all $(s, t) \in R$. A special case of weight-reducing presentations are *length-reducing presentations,* where $|s| > |t|$ for all $(s, t) \in R$. In [15] the author has shown that for every fixed weight-reducing and confluent presentation the (uncompressed) word problem is in LOGCFL [24]. Since $\mathrm{LOGCFL} \subseteq \mathrm{NSPACE}(\log^2(n))$ [13], Prop. 2 implies:

Proposition 3. *For every weight-reducing and confluent presentation* (Γ, R)*, the compressed word problem for* $\mathcal{M}(\Gamma, R)$ *is in PSPACE.*

In the rest of this section, we show that PSPACE-hardness can be deduced already for a quite small subclass of weight-reducing and confluent presentations.

A presentation (Γ, R) is called *monadic* if for every $(\ell, r) \in R$: $|\ell| > |r|$ and $|r| \leq 1$. A *2-monadic* presentation is a monadic presentation (Γ, R) such that moreover $|\ell| = 2$ for every $\ell \in \mathrm{dom}(R)$. In the following, we present a construction that reduces the reachability problem for directed forests to the (uncompressed) word problem of a fixed 2-monadic and confluent presentation (Γ, R). Let $\Gamma = \{b_0, b_1, c_0, c_1, c_2, \#, \$, \triangleright, 0\}$ and let R be the 2-monadic semi-Thue system consisting of the following rules:

(1) $b_0 x \to \varepsilon$ for all $x \in \{\$, c_0, c_1, c_2\}$	(2) $b_1 c_0 \to \varepsilon$
(3) $b_1 \$ \to \triangleright$	(4) $\triangleright c_i \to \triangleright$ for all $i \in \{0, 1, 2\}$
(5) $\triangleright \$ \to \$$	(6) $\# \$ \to \varepsilon$
(7) $b_1 c_2 \to 0$	
(8) $0x \to 0$ for all $x \in \Gamma$	(9) $x0 \to 0$ for all $x \in \Gamma$

Only the rules involving the absorbing symbol 0 produce overlappings. In the resulting critical pairs, both words can be reduced to 0. Thus, R is confluent.

Assume now that (V, E) is a directed forest, where $V = \{v_1, \ldots, v_n\}$ and $i < j$ whenever $(v_i, v_j) \in E$. Let $v_\alpha \in V$ and $U \subseteq V$ be a set of nodes such that every

node in U has outdegree 0. For $i \le j$ we define the interval $I_{i,j} = \{v_k \mid i \le k \le j\}$. Thus, $I_{1,n} = V$. If $i > j$ we set $I_{i,j} = \emptyset$. For every $i \in \{1, \ldots, n\}$ let:

$$\delta_i = \begin{cases} c_0^{n-j+i+1} & \text{if } (v_i, v_j) \text{ is the unique outgoing edge at node } v_i \\ c_1 & \text{if } v_i \in V \setminus U \text{ and } v_i \text{ has no outgoing edge} \\ c_2 & \text{if } v_i \in U \text{ (and thus has no outgoing edge)} \end{cases}$$

For an interval $I_{i,j}$ $(i \le j)$ let $\sigma[I_{i,j}] = \delta_i \$ \delta_{i+1} \$ \cdots \delta_j \$$. We set $\sigma[\emptyset] = \varepsilon$. Using the rules in (4) and (5) we get $\triangleright\sigma[I_{i,j}] \xrightarrow{*}_R \$\sigma[I_{i+1,j}]$ if $i \le j$. Finally, define

$$\beta = |\sigma[I_{1,\alpha-1}]| \quad \text{and} \quad w(v_\alpha, U) = (\#b_1^n)^n b_0^\beta \sigma[I_{1,n}].$$

Lemma 3. *We have* $w(v_\alpha, U) \overset{*}{\leftrightarrow}_R 0$ *if and only if* $\exists v_i \in U : (v_\alpha, v_i) \in E^*$.

The previous lemma yields the following result that is of independent interest. It sharpens a corresponding result of [4] for monadic systems.

Theorem 5. *There exists a fixed 2-monadic and confluent presentation* (Γ, R) *such that the word problem for* $\mathcal{M}(\Gamma, R)$ *is L-hard under* NC^1*-reductions.*

Theorem 6. *There exists a fixed 2-monadic and confluent presentation with a PSPACE-complete compressed word problem.*

Proof. We show that the compressed word problem for the 2-monadic presentation (Γ, R) from the previous discussion is PSPACE-complete. The upper bound follows from Prop. 3. For the lower bound we have to repeat a construction from [15]. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, q_f)$ be a deterministic linear bounded automaton (where Q is the set of states, Σ is the tape alphabet, q_0 (resp. q_f) is the initial (resp. final) state, and $\delta : Q \setminus \{q_f\} \times \Sigma \to Q \times \Sigma \times \{-1, +1\}$ is the transition function) that accepts a PSPACE-complete language. Such an automaton exists, see, e.g., [3]. Let $w \in \Sigma^*$ be an input for $\mathcal{A}$ with $|w| = N$. We may assume that $\mathcal{A}$ operates in phases, where a single phase consists of a sequence of $2 \cdot N$ transitions of the form $q_1\gamma_1 \overset{*}{\Rightarrow}_{\mathcal{A}} \gamma_2 q_2 \overset{*}{\Rightarrow}_{\mathcal{A}} q_3\gamma_3$, where $\gamma_1, \gamma_2, \gamma_3 \in \Sigma^N$ and $q_1, q_2, q_3 \in Q$. During the sequence $q_1\gamma_1 \overset{*}{\Rightarrow}_{\mathcal{A}} \gamma_2 q_2$ (resp. $\gamma_2 q_2 \overset{*}{\Rightarrow}_{\mathcal{A}} q_3\gamma_3$) only right-moves (resp. left-moves) are made. The automaton $\mathcal{A}$ accepts, if it reaches the final state q_f. Let $c > 0$ be a constant such that if w is accepted by $\mathcal{A}$, then $\mathcal{A}$, started on w, reaches the final state q_f after at most $2^{c \cdot N}$ phases. Let $\widehat{\Sigma}$ be a disjoint copy of Σ and similarly for $\widehat{Q}$. Let $\Delta = \Sigma \cup \widehat{\Sigma} \cup \{\triangleleft, 0, 1, \pounds\}$ and $\Theta = Q \cup \widehat{Q} \cup \Delta$ and let S be the semi-Thue system over Θ that consists of the following rules, where x ranges over all symbols from Δ :

$0\widehat{q}x \to \widehat{q}\pounds x$ for all $q \in Q \setminus \{q_f\}$	$xqa \to x\widehat{b}p$ if $\delta(q,a) = (p,b,+1)$
$1\widehat{q}x \to 0qx$ for all $q \in Q \setminus \{q_f\}$	$\widehat{a}\,\widehat{q}x \to \widehat{p}bx$ if $\delta(q,a) = (p,b,-1)$
$xq\pounds \to x1q$ for all $q \in Q \setminus \{q_f\}$	$xq\triangleleft \to x\widehat{q}\triangleleft$ for all $q \in Q \setminus \{q_f\}$

Note that $\mathrm{dom}(R) \subseteq \Delta(Q \cup \widehat{Q})\Delta$. Moreover, (Θ, S) is length-preserving and for any linear order on Θ satisfying $Q \succ 1 \succ 0 \succ \widehat{\Sigma} \succ \widehat{Q}$ we have (for the lexicographic extension of $\succ$) $s \succ t$ whenever $s \to_S t$. Let us choose such a linear order that moreover satisfies $Q \succ \Delta \succ \widehat{Q}$. In [15] the author argued that w is accepted by $\mathcal{A}$ if and only if $1q_0\pounds^{c\cdot N}w\triangleleft \xrightarrow{*}_S v$ for some word v with $\mathrm{alph}(v) \cap \{q_f, \widehat{q}_f\} \neq \emptyset$ (we have slightly modified the construction form [15] but the principal idea is the same). For $m = (c+1)N$ let $V = \bigcup_{i=0}^{m} \Delta^{i+1}(Q \cup \widehat{Q})\Delta^{m-i+1}$. Note that any S-derivation starting from $1q_0\pounds^{c\cdot N}w\triangleleft$ is completely contained in V. On the set V we construct a directed forest *(V, E)* by taking $E = (V \times V) \cap \to_S$. If we order V lexicographically by $\succ$ and write $V = \{v_1, \ldots, v_n\}$ with $v_1 \succ v_2 \succ \cdots \succ v_n$, then $(v_i, v_j) \in E$ implies $i < j$, i.e., *(V, E)* is an ordered directed forest. Note that $n = 2(m+1) \cdot |Q| \cdot |\Delta|^{m+2}$, which belongs to $2^{O(N)}$. Let $U = \{v \in V \mid \mathrm{alph}(v) \cap \{q_f, \widehat{q}_f\} \neq \emptyset\}$ and $v_\alpha = 1q_0\pounds^{c\cdot N}w\triangleleft$. Thus, $\alpha - 1$ is the number of words from V that are lexicographically larger than $1q_0\pounds^{c\cdot N}w\triangleleft$. The number α can be easily calculated in polynomial time from the input w.

The automaton $\mathcal{A}$ accepts w if and only if there is a path in *(V, E)* from v_α to a node in U. By Lemma 3 this is equivalent to $w(v_\alpha, U) \overset{*}{\leftrightarrow}_R 0$. Thus, it remains to show that $w(v_\alpha, U) \in \Gamma^*$ can be generated by a small SLP. Recall the definition of the words δ_i and $\sigma[I] \in \Gamma^*$, where $1 \leq i \leq n$ and I is an interval of $(V, \succ)$, from the discussion preceding Lemma 3. Note that if $v_i = u_1\ell u_2 \to_S u_1 r u_2 = v_j$ with $(\ell, r) \in S$, then the number $j - i$ (i.e., the number of words from V that are lexicographically between v_i and v_j) only depends on the rule (ℓ, r) (and thus ℓ) and $|u_2|$. We call this number $\lambda(\ell, |u_2|)$; it is of size $2^{O(N)}$. We now describe a small SLP that generates the word $\sigma[V] \in \Gamma^*$. Assume that $Q = \{p_1, \ldots, p_{n_1}\}$ and $\Delta = \{a_1, \ldots, a_{n_2}\}$ with $p_i \succ p_{i+1}$, $\widehat{p}_i \succ \widehat{p}_{i+1}$, and $a_i \succ a_{i+1}$. We introduce the following productions ($\prod_{i=1}^{k} u_i$ abbreviates $u_1 \cdots u_k$):

$$A_i \to \prod_{j=1}^{n_2} B_{i,j} A_{i+1} \widehat{B}_{i,j} \text{ for } 0 \leq i < m, \qquad A_m \to \prod_{j=1}^{n_2} B_{m,j} \widehat{B}_{m,j}$$

$$B_{i,j} \to \prod_{k=1}^{n_1} \prod_{\ell=1}^{n_2} (C_{i,j,k,\ell}\$)^{|\Delta|^{m-i}} \text{ for } 0 \leq i \leq m,\ 1 \leq j \leq n_2$$

$$C_{i,j,k,\ell} \to \begin{cases} c_0^{n-\lambda(a_j p_k a_\ell, m-i)+1} & \text{if } a_j p_k a_\ell \in \mathrm{dom}(R) \\ c_1 & \text{if } a_j p_k a_\ell \notin \mathrm{dom}(R) \text{ and } p_k \neq q_f \\ c_2 & \text{if } p_k = q_f \end{cases}$$

$$\widehat{B}_{i,j} \to \prod_{k=1}^{n_1} \prod_{\ell=1}^{n_2} (\widehat{C}_{i,j,k,\ell}\$)^{|\Delta|^{m-i}} \text{ for } 0 \leq i \leq m,\ 1 \leq j \leq n_2$$

$$\widehat{C}_{i,j,k,\ell} \to \begin{cases} c_0^{n-\lambda(a_j \widehat{p}_k a_\ell, m-i)+1} & \text{if } a_j \widehat{p}_k a_\ell \in \mathrm{dom}(R) \\ c_1 & \text{if } a_j \widehat{p}_k a_\ell \notin \mathrm{dom}(R) \text{ and } \widehat{p}_k \neq \widehat{q}_f \\ c_2 & \text{if } \widehat{p}_k = \widehat{q}_f \end{cases}$$

The integer exponents that appear in the right-hand sides of these productions are all of size $2^{O(N)}$ and can therefore be easily replaced by ordinary productions.

Note that $\mathrm{eval}(C_{i,j,k,\ell}) = \delta_s$ for every $v_s \in \Delta^i a_j p_k a_\ell \Delta^{m-i}$ and $\mathrm{eval}(\widehat{C}_{i,j,k,\ell}) = \delta_s$ for every $v_s \in \Delta^i a_j \widehat{p}_k a_\ell \Delta^{m-i}$. It follows that for all $0 \le i \le m$, all $u \in \Delta^i$, and all $1 \le j \le n_2$ we have (note that $ua_jQ\Delta^{m-i+1} \subseteq V$ is an interval of $(V, \succ)$)

$$\mathrm{eval}(B_{i,j}) = \sigma[ua_jQ\Delta^{m-i+1}] \quad \text{and} \quad \mathrm{eval}(\widehat{B}_{i,j}) = \sigma[ua_j\widehat{Q}\Delta^{m-i+1}].$$

By induction on $i \in \{0, \ldots, m\}$ (for $i = m$ down to 0), we can show that $\sigma[I] = \mathrm{eval}(A_i)$, where I is the interval $\bigcup_{j=1}^{m-i+1} u\Delta^j(Q \cup \widehat{Q})\Delta^{m-i-j+2}$ of the linear order $(V, \succ)$ and $u \in \Delta^i$ is arbitrary. For $i = 0$ we get $\mathrm{eval}(A_0) = \sigma[V]$. The number $\beta = |\sigma[I_{1,\alpha-1}]| \in 2^{O(N)}$ can be calculated from the input word w using simple arithmetic. Now it is easy to construct a SLP G of size polynomial in the input size N with $\mathrm{eval}(G) = (\#b_1^n)^n b_0^\beta \sigma[V] = w(v_\alpha, U)$. This concludes the proof. □

Since (Γ, R) is monadic and confluent, the language $\{w \in \Gamma^* \mid w \xrightarrow{*}_R 0\}$ is deterministic context-free [8, Thm. 4.2.7]. Thus, we obtain a fixed deterministic context-free language with a PSPACE-complete compressed membership problem. This solves an open problem from [9,19]. We can even show a slightly stronger result: In [10] a language is called *deterministic linear* if it is accepted by a deterministic 1-turn pushdown automaton. It is easy to see that the language $\{w \in \Gamma^* \mid w \xrightarrow{*}_R 0\} \cap (\#b_1^+)^+ b_0^+ ((c_0^+ \cup c_1 \cup c_2)\$)^+$ is deterministic linear. Moreover, it contains all words of the form $w(v_\alpha, U)$. Thus, we obtain:

Corollary 1. *There exists a fixed deterministic linear language L such that the compressed membership problem for L is PSPACE-complete.*

Also a uniform variant of the compressed membership problem for context-free languages is PSPACE-complete:

Theorem 7. *The following problem is PSPACE-complete:*
INPUT: A context-free grammar G and a SLP H
QUESTION: $\mathrm{eval}(H) \in L(G)$*?*

Finally, we take a look at EXPSPACE-complete cases: A presentation (Γ, R) is *weight-lexicographic* if there are a linear order $\succ$ on Γ and a weight-function f on Γ^* with $f(\ell) > f(r)$ or $(f(\ell) = f(r) \wedge \ell \succ r)$ for all $(\ell, r) \in R$. If $|\ell| > |r|$ or $(|\ell| = |r| \wedge \ell \succ r)$ for all $(\ell, r) \in R$, then (Γ, R) is *length-lexicographic*. A slight variation of a construction from [15] yields the following two results:

Theorem 8. *For every weight-lexicographic and confluent presentation, the compressed word problem is in EXPSPACE. There is a fixed length-lexicographic and confluent presentation with an EXPSPACE-complete compressed word problem.*

Theorem 9. *There exists a fixed context-sensitive language L such that the compressed membership problem for L is EXPSPACE-complete.*

6 Circuit Complexity and Compression

In this section we study compressed membership problems for languages from very low complexity classes, which are usually defined by uniform families of small depth Boolean circuits. An equivalent and for our purpose more suitable definition is based on *alternating Turing-machines* with logarithmic time bounds. See [17] for background on alternating Turing-machines. When dealing with logarithmic time bounds it is necessary to enrich the machine model with a random access mechanism in form of a special address tape that contains a binary coded number p. If the machine enters a special query state, then it has random access to the p-th input position. ALOGTIME is the class of all languages that can be recognized on an alternating Turing-machine in time $O(\log(n))$, it is equal to uniform NC^1. Within ALOGTIME, we can define the logtime hierarchy: For $k \geq 1$, $\Sigma_k^{\log}$ (resp. $\Pi_k^{\log}$) is the class of all languages that can be decided by an alternating Turing-machine in time $O(\log(n))$ within $k-1$ alternations, starting in an existential (resp. universal) state. In [2], $\Sigma_k^{\log} \cup \Pi_k^{\log}$ is proposed as a uniform version of the circuit complexity class AC_k^0. The union $\bigcup_{k\geq 1} \Sigma_k^{\log} \cup \Pi_k^{\log}$ is called the *logtime hierarchy* LH [22]. It turns out that in a compressed setting, the levels of LH and the polynomial time hierarchy $\mathrm{PH} = \bigcup_{k\geq 1} \Sigma_k^{\mathrm{poly}} \cup \Pi_k^{\mathrm{poly}}$ (see [17] for details on PH) are in a tight correspondence:

Theorem 10. *For every language in $\Sigma_k^{\log}$ ($\Pi_k^{\log}$) the compressed membership problem belongs to Σ_k^{poly} (Π_k^{poly}). There is a fixed language in $\Sigma_k^{\log}$ ($\Pi_k^{\log}$) with a Σ_k^{poly}-complete (Π_k^{poly}-complete) compressed membership problem.*

Every language in $\bigcup_{c>0} \mathrm{NSPACE}(\log^c(n))$ has a compressed membership problem within PSPACE (Prop. 2). Languages with a PSPACE-complete compressed membership problem can be already found in $\mathrm{ALOGTIME} \subseteq \mathrm{DSPACE}(\log(n))$:

Theorem 11. *There exists a fixed language L in ALOGTIME such that the compressed membership problem for L is PSPACE-complete.*

It is *not* the case that for every ALOGTIME-complete language the compressed membership problem is PSPACE-complete (unless P = PSPACE): The word problem for the finite group S_5 is ALOGTIME-complete [1] but its compressed word problem is in P. Thus, a general upgrading theorem analogously to [25] does not hold for straight-line programs. This is similar to the situation for hierarchical graphs [12], where the correlation between the complexity of a problem in its compressed and uncompressed variant, respectively, is quite loose.

References

1. D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.,* 38:150–164, 1989.
2. D. A. M. Barrington, C.-J. Lu, P. B. Miltersen, and S. Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *Proc. STACS 98,* LNCS 1373, pages 73–83. Springer, 1998.

3. G. Bauer and F. Otto. Finite complete rewriting systems and the complexity of the word problem. *Acta Inf.,* 21:521–540, 1984.
4. M. Beaudry, M. Holzer, G. Niemann, and F. Otto. McNaughton families of languages. *Theor. Comput. Sci.,* 290(3):1581–1628, 2003.
5. M. Beaudry, P. McKenzie, P. Péladeau, and D. Thérien. Finite monoids: From word to circuit evaluation. *SIAM J. Comput.,* 26(1):138–152, 1997.
6. R. V. Book. Homogeneous Thue systems and the Church–Rosser property. *Discrete Math.,* 48:137–145, 1984.
7. R. V. Book, M. Jantzen, B. Monien, C. P.Ó'Dúnlaing, and C. Wrathall. On the complexity of word problems in certain Thue systems. In *Proc. MFCS'81,* LNCS 118, pages 216–223. Springer, 1981.
8. R. V. Book and F. Otto. *String–Rewriting Systems.* Springer, 1993.
9. L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding (extended abstract). In *Proc. SWAT 1996,* LNCS 1097, pages 392–403. Springer, 1996.
10. M. Holzer and K.-J. Lange. On the complexities of linear LL(1) and LR(1) grammars. In *Proc. FCT'93,* LNCS 710, pages 299–308. Springer, 1993.
11. M. W. Krentel. The complexity of optimization problems. *J. Comput. Syst. Sci.,* 36(3):490–509, 1988.
12. T. Lengauer and E. Wanke. The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems. *J. Comput. Syst. Sci.,* 44:63–93, 1992.
13. P. M. Lewis II, R. E. Stearns, and J. Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Proc. Sixth Annual IEEE Symp. on Switching Circuit Theory and Logic Design,* pages 191–202, 1965.
14. R. J. Lipton and Y. Zalcstein. Word problems solvable in logspace. *J. Assoc. Comput. Mach.,* 24(3):522–526, 1977.
15. M. Lohrey. Word problems and confluence problems for restricted semi-Thue systems. In *Proc. RTA 2000,* LNCS 1833, pages 172–186. Springer, 2000.
16. M. Lohrey. Word problems for 2-homogeneous monoids and symmetric logspace. In *Proc. MFCS 2001,* LNCS 2136, pages 500–511. Springer, 2001.
17. C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.
18. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA '94,* LNCS 855, pages 460–470. Springer, 1994.
19. W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa,* pages 262–272. Springer, 1999.
20. D. Robinson. *Parallel Algorithms for Group Word Problems.* PhD thesis, University of California, San Diego, 1993.
21. W. Rytter. Compressed and fully compressed pattern matching in one and two dimensions. *Proc. IEEE,* 88(11):1769–1778, 2000.
22. M. Sipser. Borel sets and circuit complexity. In *Proc. STOC 1983,* pages 61–69. ACM Press, 1983.
23. J. Stillwell. The word problem and the isomorphism problem for groups. *Bull. Am. Math. Soc., New Ser.,* 6(1):33–56, 1982.
24. I. H. Sudborough. On the tape complexity of deterministic context–free languages. *J. Assoc. Comput. Mach.,* 25(3):405–414, 1978.
25. H. Veith. Succinct representation, leaf languages, and projection reductions. *Inf. Control,* 142(2):207–236, 1998.
26. K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Inf.,* 23(3):325–356, 1986.

27. Y. Zhang and R. Gupta. Path matching in compressed control flow traces. In *Proc. DCC 2002,* pages 132–141. IEEE Computer Society Press, 2002.
28. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. on Inf. Theory,* 23(3):337–343, 1977.

Complexity of Pseudoknot Prediction in Simple Models

Rune B. Lyngsø

Dept. of Statistics, Oxford University, Oxford, OX1 3TG, United Kingdom
lyngsoe@stats.ox.ac.uk

Abstract. Efficient exact algorithms for finding optimal secondary structures of RNA sequences have been known for a quarter of a century. However, these algorithms are restricted to structures without overlapping base pairs, or pseudoknots. The ability to include pseudoknots has gained increased attention over the last five years, but three recent publications indicate that this might leave the problem intractable. In this paper we further investigate the complexity of the pseudoknot prediction problem in two simple models based on base pair stacking. We confirm the intractability of pseudoknot prediction by proving it **NP** hard for binary strings in one model, and for strings over an unbounded alphabet in the other model. Conversely, we are also able to present a polynomial time algorithm for pseudoknot prediction for strings over a fixed size alphabet in the second model and a polynomial time approximation scheme for pseudoknot prediction for strings over a fixed size alphabet in the first model.

1 Introduction

Proteins usually get all the attention when talk is on molecular biological processes, with ribonucleic acids, or RNA, relegated a simple messenger role. It is, however, well known that functional, or non coding, RNA is a key component in several vital processes, perhaps most notably by making up most of the ribosome, the molecule translating messenger RNA to proteins. Moreover, new non coding RNA's and functionally important parts of messenger RNA's are constantly being discovered. The pervasiveness of functional RNA in core biological processes has even led to the theory of an RNA world [1], a time near the origin of life when biology was based on RNA or RNA-like molecules, and DNA and proteins had not yet been added to the apparatus of life.

The major driving force of structure formation for RNA molecules is Watson–Crick and wobble G, U base pair formation, and in particular stacking of neighbouring base pairs. If $i \cdot j$ denotes a base pair between the i'th and the j'th base of an RNA sequence, two base pairs $i \cdot j$ and $i' \cdot j'$ are stacking if $i' = i+1$ and $j' = j-1$; a maximal contiguous sequence of $m+1$ consecutively stacking base pairs, $i \cdot j, \ldots, (i+m) \cdot (j-m)$, is called a helix of length $m+1$. The set of base pairs in the three dimensional structure of an RNA molecule is denoted the secondary structure of that RNA molecule. More generally, secondary structure is used to refer to any (legal) set of base pairs for an RNA sequence. Algorithms for finding optimum secondary structures for an RNA sequence in thermodynamic models taking base pair stacking and loop (i.e. regions of unpaired bases) destabilising effects into account have been known for almost twenty five years [2]. A major deficiency of these algorithms, however, is that they do not consider structures

J. Diaz et at. (Eds.): ICALP 2004, LNCS 3142, pp. 919–931, 2004.

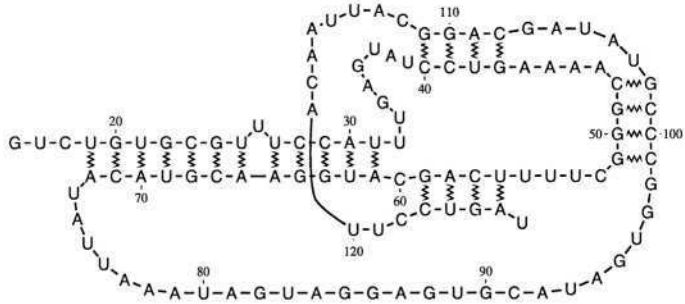


Fig. 1. Secondary structure of the *Escherichia coli* α operon mRNA from position 16 to position 127, cf. [12, Figure 1]. The backbone of the RNA molecule is drawn as straight lines while base pairings are shown with zigzagged lines. E.g. the base pairs $20 \cdot 71$, $40 \cdot 111$, $41 \cdot 110$, and $59 \cdot 123$, together with the parts of the backbone connecting the involved bases, form a non-planar substructure equivalent to $K_{3,3}$.

containing pseudoknots. Though it is not known whether pseudoknots are essential per se, there are numerous examples where evolution has led to a non coding RNA gene with a pseudoknot substructure essential for its functioning [3,4].

At its simplest a pseudoknot is just two overlapping base pairs. Two base pairs $i \cdot j$ and $i' \cdot j'$ are overlapping if $i < i' < j < j'$. More generally pseudoknots are used to refer to pairs of substructures, e.g. helices, that contain overlapping base pairs. If the stability of a secondary structure is modelled by independent contributions from the base pairs of the structure, we can find the most stable structure, including arbitrary pseudoknots, by maximum weighted matching [5]. However, evidence exists in abundance that considering base pairs in isolation is an oversimplification. Hence, some attempts have been made to expand the set of structures considered in [2] to allow structures containing some pseudoknots while still allowing similar thermodynamic energy rules and efficient exact algorithms for finding the optimum structure [6,7,8, 9]. Conversely, several recent publications indicate that extending the set of structures considered to allow arbitrary pseudoknots leaves the problem of finding the optimum structure **NP** hard [7,10,11].

One can criticise the **NP** hardness results of these three papers for assuming unrealistic models of RNA secondary structure formation, though. In [10] the scoring function is not fixed but assumed part of the input, i.e. the scores of structural elements varies with the sequence. In [7,11] the set of legal structures is restricted to be planar. A structure is planar if the graph consisting of the bases as nodes and the backbone and base pair connections as edges is planar. The requirement of planarity is not based on observed real world restrictions as non-planar structures are known, cf. Fig. 1.

The contribution of this paper is to investigate the computational complexity of finding optimum general secondary structures, i.e. structures that may contain non-planar pseudoknots, with structures scored by two of the simplest possible functions taking stacking into account. One function, introduced in [11], scores a secondary structure by the number of base pair stackings it contains. The rationale for this is that base

pair stackings by and large is the only structural element with a stabilising contribution to secondary structures in the canonical parametrisation, cf. [13], of the energy model assumed by [2]. For this scoring function we provide a simple proof that it is **NP** hard to find the optimum structure of an RNA sequence, and strengthen this to also hold for binary strings. We further present a polynomial time approximation scheme (PTAS) for finding structures with a score close to optimum. The other scoring function considered counts the number of stacking base pairs. For this function we are only able to establish the **NP** hardness of finding the optimum structure when allowed an unbounded alphabet. We complement this result with an algorithm that for strings over any alphabet of fixed size finds the optimum structure in polynomial time. The practical relevance of this algorithm is diminished by polynomial time being $O\left(n^{81}\right)$ for RNA sequences.

In Sect. 2 we give a formal specification of the models and scoring functions investigated in this paper. In Sect. 3 we provide proofs that finding an optimum secondary structure with pseudoknots is hard. In Sect. 4 we present a polynomial time algorithm for finding the optimum structure according to one scoring function, and a PTAS for finding a structure approximating the optimum score according to the other scoring function. Finally, in Sect. 5 we briefly discuss some open questions.

2 Folding Model

We will assume a model for secondary structures where only some types of base pairs are allowed, each base forms at most one base pair, and the two bases in a base pair are separated by at least three bases in the string. This last requirement is inconsequential to the proofs in the next section, as the reductions also work with the requirement removed. However, it is a consequence of steric constraints for real RNA molecules, and is thus included. This model is a straightforward generalisation of the model assumed in [2].

Definition 1 (General folding model). *For a string $s \in \Sigma^*$ over an alphabet Σ with an associated set $\mathcal{B} \subseteq \Sigma \times \Sigma$ of legal base pairs, a legal secondary structure S is a set of base pairs such that if $i \cdot j \in S$ then*

- $i, j \in \{1, \ldots, |s|\}$ *and* $i \leq j - 4$
- *if* $i' \cdot j' \in S$ *then* $\{i, j\} \cap \{i', j'\} \neq \emptyset \Leftrightarrow i \cdot j = i' \cdot j'$
- $(s_i, s_j) \in \mathcal{B}$

One instance of the above model would be the canonical RNA folding model usually assumed for finding thermodynamically optimal RNA structures. In this model only canonical, i.e. Watson–Crick and G,U wobble base pairs, are allowed.

Definition 2 (Canonical RNA folding model). *For an RNA sequence $s \in \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{U}\}^*$, a legal secondary structure S is a set of base pairs such that if $i \cdot j \in S$ then*

- $i, j \in \{1, \ldots, |s|\}$ *and* $i \leq j - 4$
- *if* $i' \cdot j' \in S$ *then* $\{i, j\} \cap \{i', j'\} \neq \emptyset \Leftrightarrow i \cdot j = i' \cdot j'$
- $\{s_i, s_j\} \in \{\{\mathsf{C}, \mathsf{G}\}, \{\mathsf{A}, \mathsf{U}\}, \{\mathsf{G}, \mathsf{U}\}\}$

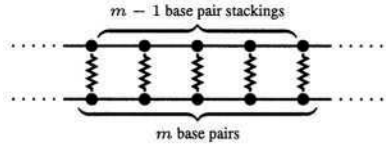


Fig. 2. A helix of m stacking base pairs contains $m - 1$ base pair stackings.

Table 1. Illustration of the differences between the three scoring functions.

	Helix length					
	1	2	3	4	5	m
Number of base pairs	1	2	3	4	5	m
Number of base pair stackings	0	1	2	3	4	$m-1$
Number of stacking base pairs	0	2	3	4	5	m

Evidently not all secondary structures that are legal by our folding model will be physically realisable due to steric constraints. We will briefly return to this in Sect. 5.

The number of base pairs in a secondary structure S is just the size of S. As previously mentioned, finding a structure with a maximum number of legal base pairs is just an instance of maximum matching, which can be solved efficiently [5]. In this paper we focus on two slight generalisations of looking at each base pair in isolation. We consider scoring functions where the score of a base pair depends on the presence of a neighbouring, or stacking, base pair in S, either by scoring a structure by the number of base pair stackings it contains or by the number of stacking base pairs it contains.

Definition 3 (Number of base pair stackings). *For a legal secondary structure S, the number of base pair stockings is defined as*

$$\mathrm{BPS}(S) = |\{i \cdot j \in S \mid (i+1) \cdot (j-1) \in S\}|$$

Definition 4 (Number of stacking base pairs). *For a legal secondary structure S, the number of stacking base pairs is defined as*

$$\mathrm{SBP}(S) = |\{i \cdot j \in S \mid (i+1) \cdot (j-1) \in S \vee (i-1) \cdot (j+1) \in S\}|$$

The difference between these scoring functions for a helix of stacking base pairs is illustrated in Fig. 2 and in Table 1. The score of an entire structure is just the sum of scores of the helices it contains.

3 Complexity Results

In this section we investigate some complexity issues for pseudoknot prediction by establishing the **NP** hardness of finding legal secondary structures that are optimum using the BPS and SBP scoring functions. We start with a simple proof that finding a

structure with a maximum number of base pair stackings in the canonical RNA folding model, cf. Def. 2, is **NP** hard. We strengthen this result to also hold for strings over a binary alphabet. Finally we prove that finding a structure with a maximum number of stacking base pairs is **NP** hard if we are allowed to use an unbounded alphabet.

3.1 Number of Base Pair Stackings

Apart from illustrating the difference between the BPS and SBP scoring functions, Fig. 2 also illustrates that under the BPS scoring function the contribution of a helix is always one less than the length of the helix, i.e. the number of base pairs in the helix. Hence, for a fixed number of base pairs, each helix these base pairs are distributed over reduces the BPS score of the structure by one. Assume that we have an RNA sequence s for which all legal base pairs have to contain a particular type of base, say a C. Further assume that the C's in s are grouped in k substrings of lengths $a_1, \ldots, a_k$, and that the bases at either end of these k substrings cannot form a legal base pair with any base in s. If a structure for s has the C's in each of the k substrings form base pairs that neatly stacks in one contiguous helix, then the BPS score of the structure is exactly $\sum_{i=1}^{k} a_i - k$. If for any of the substrings the C's are split among two or more helices, or some left unpaired, the BPS score will be less than $\sum_{i=1}^{k} a_i - k$. So to rephrase, the optimum BPS score depends on whether we can 'pack' the base pairs of each substring to form contiguous helices, or whether we have to distribute the base pairs over two or more helices, or leave part of a substring unpaired, for one or more of the k substrings.

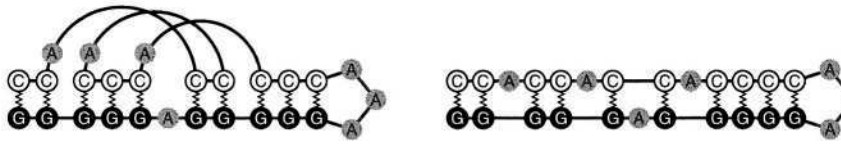
Theorem 1. *Given an RNA sequence s and a target K, it is* **NP** *hard to determine whether there is a structure S that is legal under the canonical RNA folding model and with $BPS(S) \geq K$.*

Proof. The proof is by reduction from the BIN PACKING problem, known to be *strongly* **NP** hard [14]. In the BIN PACKING problem we are given k items of sizes $a_1, \ldots, a_k$ and B bins each with capacity C, and have to determine whether the items fit into the bins. Or in more mathematical terms, we need to determine whether the k elements $a_1, \ldots, a_k$ can be partitioned into B sets, with the sum of elements in any set at most C.

Given an instance of BIN PACKING we construct the RNA sequence

$$s = \mathsf{C}^{a_1}\mathsf{AC}^{a_2}\mathsf{A}\ldots\mathsf{AC}^{a_k}\mathsf{AAA}\underbrace{\mathsf{G}^{C}\mathsf{AG}^{C}\mathsf{A}\ldots\mathsf{AG}^{C}}_{B \text{ substrings of } C \text{ G's}}$$

and the target $K = \sum_{i=1}^{k} a_i - k$. As A's can only form base pairs with U's in the canonical RNA folding model, all base pairs in a legal structure for s will be C · G base pairs and s clearly meets the assumptions discussed above. Furthermore, any C in s is separated from any G in s by at least three other bases, so any otherwise unpaired C can form a legal base pair with any otherwise unpaired G in s. Hence, we can find a structure S with BPS(S) = K iff we can partition the k substrings of C's of lengths $a_1, \ldots a_k$ into B groups that can each be fully base paired using one substring of C consecutive G's; i.e. the total length of the substrings of C's in any group can be at most C. Clearly this is possible iff the original BIN PACKING problem has a solution. The idea behind the construction is illustrated in Fig. 3.



(a) An optimum structure for the RNA sequence constructed from an instance of BIN PACKING with four items of sizes 2, 2, 3, and 3, and two bins of capacity 5.

(b) An optimum structure for the RNA sequence constructed from an instance of BIN PACKING with four items of sizes 2, 2, 2, and 4, and two bins of capacity 5.

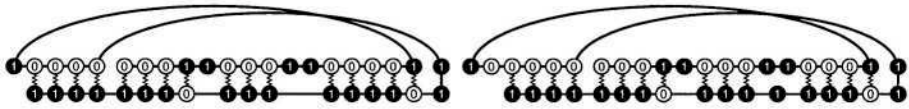
Fig. 3. Illustration of how the number of helices can be kept to one per item for an RNA sequence constructed from a 'yes' instance of BIN PACKING, while the base pairs of at least one substring corresponding to an item have to be split over at least two helices if the RNA sequence is constructed from a 'no' instance of BIN PACKING.

The length of s is $\sum_{i=1}^{k} a_i + BC + k + B + 1$. As BIN PACKING is strongly **NP** hard we can assume that $a_1, \ldots, a_k, B, C$ are all polynomially bounded by the size of the original BIN PACKING instance. Hence, $|s|$ is also polynomially bounded by the size of the original BIN PACKING instance. Clearly the same holds for a fair representation of the target K. Constructing s and K in time polynomial in the size of their representations is trivial. □

We now proceed to study the problem of finding optimum secondary structures for strings over a binary alphabet. I.e., in the following we will assume an alphabet $\Sigma = \{0,1\}$ and a set of legal base pairs $\mathcal{B} = \{(0,1),(1,0)\}$ in the context of the general folding model, cf. Def. 1. A biological motivation for considering strings over a binary alphabet could be that the only purine/pyrimidine base pair not frequently observed in real RNA secondary structures are A,C base pairs. So one could imagine just representing an RNA sequence by its purine/pyrimidine sequence, find the optimum structures for this reduced sequence, and finally eliminating all A,C base pairs from these structures as a heuristic for finding good secondary structures for RNA sequences. But the main motivation for considering strings over a binary alphabet is of course to find the simplest possible model for which the pseudoknot prediction problem remains **NP** hard.

Theorem 2. *Given a string $s \in \{0,1\}^*$ and a target K, it is* **NP** *hard to determine whether there is a structure S with $BPS(S) \geq K$ that is legal under the general folding model with $\mathcal{B} = \{(0,1),(1,0)\}$.*

Proof. The proof is a slight modification of the proof for Theorem 1, but with only a binary alphabet we do not have the equivalent of the A's to separate the substrings representing items and bins with something guaranteed not to form base pairs. We will need slightly stronger assumptions about the BIN PACKING instances, namely that $3 \leq a_i \leq C$ for $1 \leq i \leq k$ and $2 \leq B \leq k$. By inspection of the proof in [14] one can check that the BIN PACKING problem remains **NP** hard when imposing these assumptions. Given such an instance of BIN PACKING we now construct the string



(a) An optimum structure for the string constructed from an instance of BIN PACKING with four items of sizes 3, 3, 4, and 4, and two bins of capacity 7.

(b) An optimum structure for the string constructed from an instance of BIN PACKING with four items of sizes 3, 3, 3, and 5, and two bins of capacity 7.

Fig. 4. Illustration of the reduction from BIN PACKING for a binary alphabet, compare to Fig. 3.

$$s = 0^{a_1}110^{a_2}11\ldots110^{a_k}110\underbrace{1^C01^C0\ldots01^C}_{B \text{ substrings of } C \text{ 1's}}$$

and the target $K = \sum_{i=1}^{k} a_i - k + B$. Note that the target is exactly k less than the number of 0's in s. Hence, if a structure has all 0's base paired and contains k helices it will exactly meet the target. It is not possible to find a structure with all 0's base paired and less than k helices. This follows from s not containing the substring 1001 as a consequence of the assumption that $a_i \geq 3$ for all i. Hence, the 0's of the two substrings of 0's representing two different items cannot form base pairs that are part of the same helix. If the base pairs formed by any of the substrings of 0's are split over more than one helix or if some 0's are left unpaired the score of the structure will be less than K. So again we can only find a structure meeting the target if the k items can be packed into the B bins.

If the k items can be packed into the B bins we still need to argue that this allows us to find a secondary structure with K base pair stackings. By just base pairing the substrings of 0's representing the items to substrings of 1's representing the bin the corresponding item is assigned to in a solution of the BIN PACKING instance we only get $\sum_{i=1}^{k} a_i$ base pair stackings, i.e. we are still B base pair stackings short. With only the B 0's separating the bin representations left we thus need to argue that each of these can form a legal base pair that extends an already existing helix. As the substring of 0's representing an item is always followed by a 1, in general we can form these extra base pairs by base pairing the 0 preceding the representation of a bin with the 1 following the representation of one of the items that is assigned to that bin, and making sure that the helix involving the 0's representing this item is formed with the first 1's of the bin representation. As there are at least as many items as bins we may safely assume that the solution to the BIN PACKING instance has at least one item assigned to each bin, so this strategy yields an extra B base pair stackings.

The only detail left is whether the base pairs violate the requirement of the two bases in a base pair being separated by at least three other bases. Any 0 representing an item and any 1 representing a bin are separated by at least three other bases, so these base pairs are not a problem. But if the 1 following the k'th item was chosen to form a base pair with the 0 preceding the first bin, this would result in a base pair where the two bases are only separated by one other base. But as a permutation of the bin assignment

for a solution to the original BIN PACKING problem will again be a solution, by the requirement that $B \geq 2$ we can assume that the solution does not assign the k'th item to the first bin. The construction is illustrated in Fig. 4. That s and K can be constructed in polynomial time again follows from BIN PACKING being strongly **NP** hard. □

3.2 Number of Stacking Base Pairs

The thermodynamic parameters described in [13] assigns an energy to each loop in a secondary structure. A loop is any closed, base pair free region of the structure, e.g. the region enclosed by two stacking base pairs and the parts of the backbone connecting them or the region enclosed by base pairs $43 \cdot 108$ and $48 \cdot 102$ and the parts of the backbone connecting them in Fig. 1. The BPS scoring function is thus seemingly a better simplification of the scoring function actually used for RNA secondary structure prediction than the SBP scoring function – the BPS scoring function scores regions enclosed by stacking base pairs rather than the base pairs enclosing these regions. However, certain elements of the thermodynamic parameters show closer relationship to the SBP scoring function: A, U and G, U base pairs at the end of helices are penalised to account for the fact that these types of base pairs have fewer hydrogen bonds than C, G base pairs. Hence, we claim that it is of interest to also investigate the computational aspect of finding structures that are optimal under the SBP scoring function.

Theorem 3. *It is* **NP** *hard, given an alphabet Σ, a set of legal base pairs $\mathcal{B} \subseteq \Sigma \times \Sigma$, a string $s \in \Sigma^*$ and a target K, to determine whether s has a structure S that is legal under the general folding model with* $\mathrm{SBP}(S) \geq K$.

Proof. The proof is by a reduction from RESTRICTED SATISFIABILITY, a restricted version of SATISFIABILITY where each literal occurs once or twice. An instance of SATISFIABILITY consists of a boolean formula ϕ in 3CNF form, i.e. the formula is a conjunction of clauses, each clause being the disjunction of at most three literals. A literal is either a variable, x_i, or its negation, $\neg x_i$. The problem is to decide whether there is an assignment of Boolean values to the variables in ϕ such that ϕ becomes **true**. A proof of the **NP** hardness of RESTRICTED SATISFIABILITY is provided in [14].

Given an instance ϕ of RESTRICTED SATISFIABILITY we construct an alphabet Σ, a set of legal base pairs $\mathcal{B}$, a target K, and a string $s \in \Sigma^*$ such that s has a secondary structure with K stacking legal base pairs iff ϕ is satisfiable. that consists of three parts: a variable part, a clause part, and a literal part. By construction of $\mathcal{B}$ we will ensure that legal base pairs can only be formed between a base from the literal part of s and a base from either the variable or clause part of s. Stacking base pairs between bases representing a particular literal l in the literal part and bases in the variable part will be used to indicate a truth assignment where l is **false**. Stacking base pairs between bases representing a particular literal l in the literal part and bases representing a particular clause c in the clause part will be used to indicate that l ensures that c is satisfied. I.e. bases in the literal part can be used as a witness of either the truth assignment of a variable or that a clause is satisfied, but not both. A secondary structure for s with all non-\$ bases in the clause and variable parts forming stacking base pairs will correspond to every variable having been assigned a truth value and every clause containing at least one **true**

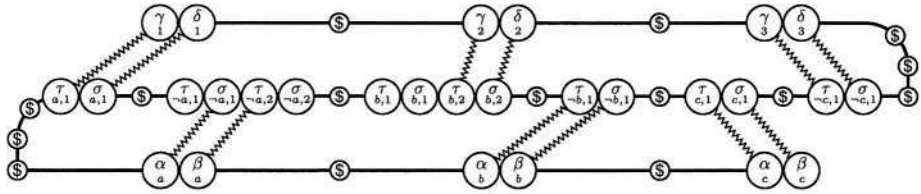


Fig. 5. Illustration of the reduction from RESTRICTED SATISFIABILITY used in the proof of Theorem 3 for the formula $\phi = (a \vee b \vee c) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b \vee \neg c)$. The secondary structure corresponds to a truth assignment with $a =$ **true**, $b =$ **true**, and $c =$ **false**. From top to bottom the constituent parts are the clause part, the literal part, and the variable part.

literal, i.e. a satisfying truth assignment for ϕ; for convenience we will be using \$ as a separating character that is guaranteed not to form a base pair, i.e. \$ will not appear in any of the base pairs in the set of legal base pairs $\mathcal{B}$ constructed.

The literal part consists of one block for each literal occurring in ϕ, with two unique bases for each occurrence of the literal, with the blocks separated by \$ bases. I.e. if the literal l occurs once in ϕ the block $\sigma_{l,1}\tau_{l,1}$ is added to s, and if l occurs twice in ϕ the block $\sigma_{l,1}\tau_{l,1}\sigma_{l,2}\tau_{l,2}$ is added to s.

The variable part consists of one block of two unique bases for each variable occurring in ϕ, with the blocks separated by \$ bases. I.e. if variable x_i occurs in ϕ the block $\alpha_{x_i}\beta_{x_i}$ is added to s. Legal base pairs are added to $\mathcal{B}$ such that the block corresponding to x_i can form a pair of stacking base pairs with the two middle bases of the two blocks representing the literals x_i and $\neg x_i$ in the literal part. I.e. if l is either x_i or $\neg x_i$ and l occurs once in ϕ, the base pairs $\{\sigma_{l,1}, \beta_{x_i}\}$ and $\{\tau_{l,1}, \alpha_{x_i}\}$ are added to $\mathcal{B}$. If l occurs twice in ϕ, the base pairs $\{\tau_{l,1}, \beta_{x_i}\}$ and $\{\sigma_{l,2}, \alpha_{x_i}\}$ are added to $\mathcal{B}$; this latter case where bases representing the two different occurrences of a literal are tied together by the legal base pairs added to $\mathcal{B}$ is the point of the reduction where it is crucial that the scoring scheme only assigns a positive contribution to base pairs if they are stacking.

The clause part consists of one block of two unique bases for each clause of ϕ, with the blocks separated by \$ bases. I.e. for the i'th clause of ϕ the block $\gamma_i\delta_i$ is added to s. Legal base pairs are added to $\mathcal{B}$ such that two stacking base pairs can be formed with any two bases representing a literal occurrence in the i'th clause. I.e. if the j'th occurrence of literal l is in the i'th clause, the base pairs $\{\gamma_i, \tau_{l,j}\}$ and $\{\delta_i, \sigma_{l,j}\}$ are added to $\mathcal{B}$.

The three parts are joined with three \$ bases separating each part. The target K is set to twice the sum of the number of unique variables occurring in ϕ and the number of clauses in ϕ, i.e. the number of non-\$ bases in the variable and clause parts of s. The alphabet Σ is the set of bases used in s.

If ϕ has a satisfying truth assignment we can form pairs of stacking base pairs between bases in the variable part and bases in the literal part corresponding to literals that become false by the truth assignment, while still being able to find two bases corresponding to a literal occurrence for each clause that has not been paired with bases in the variable part, i.e. we can find a structure for s with K stacking legal base pairs. Conversely, a structure with K stacking legal base pairs for s will have all non-\$ bases in the variable and clause parts forming base pairs. A truth assignment obtained by requiring a literal

to be **false** iff bases representing it in the literal part form base pairs with bases in the variable part will clearly satisfy ϕ, as for each clause we can find a literal whose negation is **false**. The construction is illustrated in Fig. 5. □

4 Algorithmic Results

It is somewhat unsatisfying that Theorem 3 assumes an unbounded alphabet. For one thing, the result does not establish that it is **NP** hard to find the optimum RNA secondary structure with arbitrary pseudoknots when structures are scored by the number of stacking base pairs they contain. But as we shall see in this section, such a result would be quite surprising. For strings over any fixed alphabet, the problem of finding the optimum secondary structure using the SBP scoring function turns out to be in **P**.

To see this, consider the helix of five stacking base pairs in Fig. 2. This contributes 5 to the overall score under the SBP scoring function. Breaking it into two helices of lengths two and three, the total contribution is still 2 + 3 = 5. Any helix of stacking base pairs, i.e. any helix of length at least two, can be broken into helices of lengths two or three. Hence, finding an optimum structure when only helices up to length three are considered will result in an optimum structure under the SBP scoring function.

So for a string s we could partition it into singletons, dinucleotides, and trinucleotides in all possible ways, and for each partition find a maximum weighted matching where matchings of complementary dinucleotides has weight 2 and matchings of complementary trinucleotides has weight 3. However, there is an exponential number of different partitions. But the important part of a partition, in terms of SBP score, is not the partition itself, but the number of each of the dinucleotides and trinucleotides it contains. Hence, for any prefix $s_1 \ldots s_i$ of s and count $\mathbf{c}$ of yet unpaired occurrences of each of the dinucleotides and trinucleotides in $s_1 \ldots s_i$ we can find the optimum number of stacking base pairs that can be formed in $s_1 \ldots s_i$ by the following recursion.

$$A(i,\mathbf{c}) = \max \begin{cases} A(i-1,\mathbf{c}), A(i-2,\mathbf{c}[s_{i-1}s_i\downarrow]), A(i-3,\mathbf{c}[s_{i-2}s_{i-1}s_i\downarrow] \\ \max\limits_{\sigma_1,\sigma_2;(\sigma_1,s_i),(\sigma_2,s_{i-1})\in\mathcal{B}} \{2 + A(i-2,\mathbf{c}[\sigma_1\sigma_2\uparrow]\} \\ \max\limits_{\sigma_1,\sigma_2,\sigma_3;(\sigma_1,s_i),(\sigma_2,s_{i-1}),(\sigma_3,s_{i-2})\in\mathcal{B}} \{3 + A(i-3,\mathbf{c}[\sigma_1\sigma_2\sigma_3\uparrow]\} \end{cases} \quad (1)$$

The notation $\mathbf{c}[t\uparrow]$ ($\mathbf{c}[t\downarrow]$) denotes a count identical to $\mathbf{c}$, except that the count of the string t is increased (reduced) by one. The rationale of the recursion is that we can either leave the trailing singleton, dinucleotide, or trinucleotide of $s_1 \ldots s_i$ unpaired for now, and update the count $\mathbf{c}$ accordingly. Or we can pair the trailing dinucleotide (trinucleotide) with a complementary dinucleotide (trinucleotide). The recursion of (1) can be used as the basis of a dynamic programming algorithm, where the optimum score equals $A(n,\mathbf{0})$, where $n = |s|$. Optimum structures can be determined by traceback.

The count of any dinucleotide or trinucleotide and the number of different prefixes of s is $O(n)$. The number of different dinucleotides and trinucleotides is $c = |\Sigma|^2 + |\Sigma|^3$, so the number of different entries of A we need to maintain and compute is $O(n^{1+c})$. Any one entry can be computed in time $O(1)$, so we can find $A(n,\mathbf{0})$ in time $O(n^{1+c})$. The space complexity can be reduced to $O(n^c)$ by applying the method described in [15].

For a four letter alphabet like the RNA alphabet this means a time complexity of $O(n^{81})$ and a space complexity of $O(n^{80})$.

The observant reader will have noticed that (1) does not guarantee that all base pairs formed are between bases that are separated by at least three other bases in the string. This can be amended by adding a careful, constant time bookkeeping of the status of the last few bases in the prefix. The recursion can readily be modified to allow individual scores for each type of base pair.

For the BPS scoring function, we can not apply the above technique to find the score of an optimum structure. Indeed, the fact that breaking one helix into two smaller helices reduces the score by one was the foundation of the reduction in Sect. 3.1. But considering helices up to length k would only break a helix of length m into $\lceil m/k \rceil$ helices, i.e. the contribution to the overall score counted for that particular helix would only be decreased by $\lceil m/k \rceil - 1 \leq (m-1)/k$ or a fraction $1/k$ of its actual contribution. So by amending (1) to consider substrings up to length k and using the BPS scoring function, we can find a structure with a score that is at least $(1 - 1/k)$ of the optimum score. There are $d_k = \sum_{i=2}^{k} |\Sigma|^i = (|\Sigma|^{k+1} - 1)/(|\Sigma| - 1) - |\Sigma| - 1$ different substrings over alphabet Σ of lengths between 2 and k. Hence, we can approximate the optimum score within $(1 - \epsilon)$ under the BPS score function in time $O(n^{d_{\lceil 1/\epsilon \rceil}+1})$ and space $O(n^{d_{\lceil 1/\epsilon \rceil}})$, i.e. in polynomial time for any fixed $\epsilon > 0$. This establishes the existence of a PTAS for pseudoknot prediction under the BPS scoring function. It is unlikely that a Fully PTAS, i.e. an approximation scheme where the time complexity depends only polynomially on $1/\epsilon$, exists as an $(1 - 1/n)$ approximation would equal the optimum score due to the integral nature of scores.

5 Discussion

In this paper we have proven that it is **NP** hard to find an optimum RNA secondary structure when we allow any set of base pairings, as long as all base pairs are canonical, no base is included in more than one base pair, and the bases of any base pair obey the minimum separation requirement. A lot of structures that are legal under these assumptions will not be realisable by three dimensional structures due to steric constraints. Defining a model that allows those structures, and only those structures, that can be realised by a three dimensional structure without in essence turning the secondary structure prediction problem in the model into a tertiary structure prediction problem seems a daunting task. However, by increasing the number of A's separating the item representations and the bin representations in the proof of Theorem 1 it should be possible to add enough freedom of movement of the substrings of C's and G's to meet constraints based on reasonable steric considerations. This trick can not be applied to the string constructed in the proof of Theorem 2, however, as we do not have a separator symbol that is guaranteed not to form base pairs.

Though we did manage to develop a polynomial time algorithm for finding the optimum structure of an RNA sequence under the SBP scoring function, the time complexity of $O(n^{81})$ (and space complexity of $O(n^{80})$) does render it rather useless in practice. From Theorem 3 we would expect an exponential dependence on the alphabet size. But

this still allows for the possibility of a, say, $O\left(n^{|\Sigma|}\right)$ algorithm for finding optimum structures under the SBP scoring function.

One open problem that remains is whether we can strengthen Theorem 3 to hold for strictly complementary alphabets. In the proof, some of the bases added to the alphabet can form legal base pairs with more than one other type of base, similar to the presence of the $G \cdot U$ wobble base pair in the set of legal base pairs for RNA sequences. It is still unanswered whether the pseudoknot prediction problem remains **NP** hard under the SBP scoring function if each base is included in only one legal base pair, similar to the set of Watson–Crick base pairs. One hint that this just might affect the complexity is that a strictly complementary alphabet allows us to decrease the complexity of an algorithm based on the recursion in (1) to $O\left(n^{1+c/2}\right)$. This follows as we can group dinucleotides and trinucleotides into complementary pairs for which we only need to consider cases where at most one of them have a count larger than zero.

Acknowledgements. This work was supported by EPSRC grant HAMJW, and MRC grant HAMKA. The author would like to thank the anonymous referees for useful comments, and in particular one referee for supplying errata.

References

1. Joyce, G.F.: The antiquity of RNA-based evolution. Nature **418** (2002) 214–221
2. Zuker, M., Stiegler, P.: Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Research **9** (1981) 133–148
3. Felden, B., Massire, C, Westhof, E., Atkins, J.F., Gesteland, R.F.: Phylogenetic analysis of tmRNA genes within a bacterial subgroup reveals a specific structural signature. Nucleic Acids Research **29** (2001) 1602–1607
4. Tanaka, Y., Hori, T., Tagaya, M., Sakamoto, T., Kurihara, Y., Katahira, M., Uesugi, S.: Imino proton NMR analysis of HDV ribozymes: nested double pseudoknot structure and Mg^{2+} ion-binding site close to the catalytic core in solution. Nucleic Acids Research **30** (2002) 766–774
5. Tabaska, J.E., Cary, R.B., Gabow, H.N., Stormo, G.D.: An RNA folding method capable of identifying pseudoknots and base triples. Bioinformatics **14** (1998) 691–699
6. Rivas, E., Eddy, S.: A dynamic programming algorithm for RNA structure prediction including pseudoknots. Journal of Molecular Biology **285** (1999) 2053–2068
7. Akutsu, T.: Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. Discrete Applied Mathematics **104** (2000) 45–62
8. Uemura, Y., Hasegawa, A., Kobayashi, S., Yokomori, T.: Tree adjoining grammars for RNA structure prediction. Theoretical Computer Science **210** (1999) 277–303
9. Reeder, J., Giegerich, R.: From RNA folding to thermodynamic matching, including pseudoknots. Technical Report 03, Technische Fakultät, Universität Bielefeld (2003)
10. Lyngsø, R.B., Pedersen, C.N.S.: RNA pseudoknot prediction in energy based models. Journal of Computational Biology **7** (2000) 409–428
11. Ieong, S., Kao, M.Y., Lam, T.W., Sung, W.K., Yiu, S.M.: Predicting RNA secondary structures with arbitrary pseudoknots by maximizing the number of stacking pairs. In: Proceedings of the 2nd Symposium on Bioinformatics and Bioengineering. (2001) 183–190
12. Gluick, T.C., Draper, D.E.: Thermodynamics of folding a pseudoknotted mRNA fragment. Journal of Molecular Biology **241** (1994) 246–262

13. Mathews, D.H., Sabina, J., Zuker, M., Turner, D.H.: Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. Journal of Molecular Biology **288** (1999) 911–940
14. Papadimitriou, C.M.: Computational Complexity. Addison-Wesley Publishing Company, Inc. (1994)
15. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequence. Communications of the ACM **18** (1975) 341–343

Property Testing of Regular Tree Languages*

Frédéric Magniez[1] and Michel de Rougemont[2]

[1] CNRS–LRI, UMR 8623 Université Paris–Sud, France,
magniez@lri.fr
[2] LRI & Université Paris II, France
mdr@lri.fr

Abstract. We consider the Edit distance with moves on the class of words and the class of ordered trees. We first exhibit a simple tester for the class of regular languages on words and generalize it to the class of ranked regular trees. In the complete version of the paper, we show that the distance problem is NP-complete on ordered trees.

1 Introduction

Inspired by the notion of Self-Testing [3,4], Property Testing has been initially defined and studied for graph properties [7]. It has been successfully extended for various classes of finite structures. Let $\mathbf{K}$ be a class of finite structures and a distance function dist, i.e. a function between structures of $\mathbf{K}$. An ε-tester for a class $\mathbf{K}_0 \subseteq \mathbf{K}$ is a randomized algorithm which takes a structure U_n of size n as input and decides if $U_n \in \mathbf{K}_0$ or if U_n is ε-far from $\mathbf{K}_0$ with high probability. A class $\mathbf{K}_0$ is testable if for every sufficiently small ε there exists an ε-tester for $\mathbf{K}_0$ whose time complexity is in $O(f(\varepsilon))$, *i.e.* independent of n.

For the Hamming distance, regular languages and Σ_2-definable graphs are testable [2,1]. Testers have also been generalized to the infinite regular languages [5]. In this paper we initiate the study of Property Testing with the Edit distance, when insertions and deletions of letters on words, of nodes and edges on trees, are the elementary operations. We specifically require an additional operation: the *move* of any entire subword or subtree in one step.

First (Section 3), we develop a new tester for regular languages on words that greatly simplifies the tester of [2] and improves its complexity by a $\log(1/\varepsilon)$ factor.

Then (Section 4), we initiate the study of Property Testing on trees. The testability of regular tree languages is a well known open problem [5] for the standard Edit distance. We solve this problem when moves are allowed, by proving the testability of regular ranked tree languages.

The Word Edit distance with moves decision problem and the standard Tree Edit distance decision problem are computable in polynomial time [6,10]. In the complete version of the paper, we prove that the Tree Edit distance with moves

* Complete version at http://www.lri.fr/~magniez. Work supported by *ACI Sécurité Informatique: VERA* of the French Ministry of Research.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 932–944, 2004.

is NP-complete. It is then interesting to point out that this apparently more complex distance yields a tester for regular languages, whereas we do not know such a tester for the classical Tree Edit distance.

Finally (Section 5), we discuss the possibility of generalizing the testability to unranked trees. As a direct application, it would imply than one can decide in constant time if a large XML document follows a DTD or is far from it.

2 Preliminaries

2.1 Property Testing

Recall the notion of a (Property) tester [7] on a class $\mathbf{K}$ of finite structures for which a distance function between structures has been defined.

We say that two structures $U_n, V_m \in \mathbf{K}$, whose domains are respectively of size n and m, are *ε-close* if their distance is less than $\varepsilon \times \max(n, m)$. They are *$\varepsilon$-far* if they are not ε-close. In this paper, we consider this notion of closeness for words and trees since the representation of their structure is of linear size. For other classes, such as graphs, one may define the closeness relatively to the representation size (for e.g., εn^2 for graphs) instead of the domain size.

Definition 1. *Let $\varepsilon \geq 0$ be a real. An ε-tester for a class $\mathbf{K}_0 \subseteq \mathbf{K}$ is a randomized algorithm A such that:*
(1) If $U \in \mathbf{K}_0$, A always accepts;
(2) If U is ε-far from $\mathbf{K}_0$, then Pr[A rejects] $\geq 2/3$.

The *query complexity* is the number of boolean queries to the structure U of $\mathbf{K}$. The *time complexity* is the usual time complexity where the complexity of a query is one and the time complexity of an arithmetic operation is also one.

A class $\mathbf{K}_0 \subseteq \mathbf{K}$ is *testable* if for every sufficiently small $\varepsilon > 0$, there exists an ε-tester whose time complexity depends only on ε.

2.2 Words

Let Σ be a finite alphabet of constant size and for the sake of simplicity, the reader might think that $\Sigma = \{0, 1\}$. We now consider the words on the alphabet Σ. Every word W is a finite structure $(N, [N], l : [N] \to \Sigma)$, where $[N]$ denote the set $\{1,..., N\}$. The class $\mathbf{K}$ is the set of all such structures. We will denote a subclass $\mathbf{K}_0$ of $\mathbf{K}$ as a subset $L \subseteq \Sigma^*$. In this context, a query i to some word W asks the letter $W[i] = l(i)$. Let W be a word. A word w is a *subword* of W if $w = W[i, \ldots, j]$, for some i, j.

An *elementary operation (on words)* is a deletion or an insertion of a letter, or a move: given a subword and a position, a *move* is a transformation of the word, where the subword has been removed of its current position and inserted in the given position. Notice we omit letter replacement operations since such an operation can be simulated using one deletion and one insertion.

The standard Edit distance only considers the operations without moves, and this new distance is essential for most of the arguments.

Definition 2. *The distance between two words W and W' is the minimum number of elementary operations necessary to reach W' from W, noted* $\mathsf{dist}(W, W')$. *The distance between W and a language L, noted* $\mathsf{dist}(W, L)$, *is the minimum* $\mathsf{dist}(W, W')$ *when* $W' \in L$.

2.3 Trees

Let T be an ordered Σ-tree, *i.e.* a tree with labels $\sigma \in \Sigma$ on the nodes. It is *ranked* if the degree is bounded by a fixed constant, and *unranked* otherwise. We omit the term 'ordered', since all our trees will be ordered.

Let us first consider r-ranked trees for some fixed constant r. An r-ranked tree T is a finite structure

$$(N, [N], root, l : [N] \to \Sigma, d : [N] \to [r], s : [N] \times [r] \to [N]),$$

where N is the size of T, *root* is the distinguished element representing the root of T, l is the label function, d is the degree function which gives the degree of any node, and s is the successor partial function which associates to every node v and any position $i \in [d(v)]$ the i-th successor of v.

The class $\mathbf{K}$ is the set of all such structures. We will denote a subclass $\mathbf{K}_0$ of $\mathbf{K}$ as a subset L of all r-ranked trees. In this context, a query (v, i) to some tree T asks the label and the degree of the node v and its i-th node successor in T, if $i \leq d(v)$.

The classical Tree Edit distance [10] assumes basic insertions, deletions on a tree and modifications of labels (see Figure 1). A *node insertion* (u, σ) on an edge (v_1, v_2) replaces the edge (v_1, v_2) by the edge (v_1, u), set v_2 to be the only successor of u, and labels u by σ. A *node deletion* is the inverse of a node insertion. An *edge insertion* (v, u, σ, i) to a node v of T inserts the leaf u with label σ between the $(i-1)$-th and the i-th successor of v, provided that $d(v) < r$. The inverse operation is an *edge deletion.*

We will also allow some moves in T (see Figure 1). A *complete subtree* t of T takes a node of T as root and is the substructure restricted to a subset of nodes such that all leaves of t are also leaves of T. A *move* (t, v, i) of a complete subtree

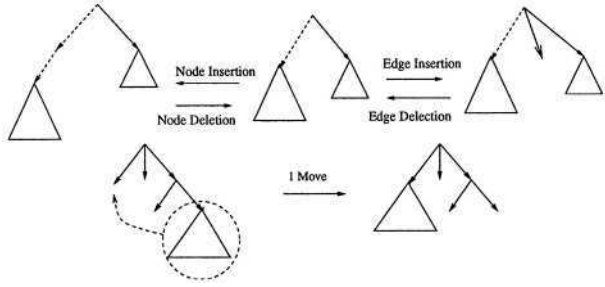


Fig. 1. Elementary operations on trees.

t to a node v moves in one step t between the $(i-1)$-th and the i-th successor of v, provided the degree of v allows it. An *elementary operation (on trees)* is one of the above operations. We define $\mathsf{dist}(T, T')$ and $\mathsf{dist}(T, L)$ as in Definition 2, for any trees T, T' and tree language L. For unranked trees, the above definitions might be generalized by removing the degree condition and replacing $[r]$ by $[N]$. Moreover the definition of a complete subtree is adapted so that t is a complete subtree of T if in addition it satisfies: every successors in t of every node v of t are subwords of the successors in T of v.

3 Testing Regular Languages

3.1 Basic Definitions

Let $\mathcal{A}$ be a deterministic automaton on words with m states, $m \geq 2$, which recognizes a language L. We say that w *connects* the states q_1 to the state q_2 when starting from q_1, the automaton $\mathcal{A}$ reaches q_2 after reading word w. If w connects q_1 to q_2, we also say that q_1 *is connected to* q_2. This notion will be used for random subwords w of a fixed word W.

Proposition 1. *Let q_1 be a state connected to q_2. Then there exists a word w of size at most m that connects q_1 to q_2.*

Let $G(\mathcal{A})$ be the directed graph whose vertices are the states of $\mathcal{A}$ and edges connects states that are connected by a word of size 1, that is a letter. We assume without lost of generality that $G(\mathcal{A})$ is connected. Since we will only speak about strongly connected components, we omit the term 'strongly'. A connected component C of $G(\mathcal{A})$ is *truly connected* if there is a non empty path of $G(\mathcal{A})$ inside C. Observe that a nontruly connected component is necessarily a singleton. We will denote by $\widehat{G}(\mathcal{A})$ the graph of the connected components of $G(\mathcal{A})$.

Let $G(\mathcal{A})$ be the directed graph whose vertices are the states of $\mathcal{A}$ and edges connects states that are connected by a word of size 1, that is a letter. We assume without lost of generality that $G(\mathcal{A})$ is connected. Since we will only speak about strongly connected components, we omit the term 'strongly'. A connected component C of $G(\mathcal{A})$ is *truly connected* if there is a non empty path of $G(\mathcal{A})$ inside C. Observe that a nontruly connected component is necessarily a singleton. Let $\widehat{G}(\mathcal{A})$ denote the graph of the connected components of $G(\mathcal{A})$.

Definition 3. *Let $\Pi = (C_1, \ldots, C_k)$ be a path of $\widehat{G}(\mathcal{A})$. Then Π is* admissible *if C_1 (resp. C_k) contains an initial (resp. final) state.*

Definition 4.

1. *Let C be a truly connected component of $G(\mathcal{A})$. A word w is* C-simply feasible *if it connects two states of C.*
2. *Let Π be a path of $\widehat{G}(\mathcal{A})$. A word w of is* Π-feasible *if it connects two states q_1 and q_2 along a path visiting only some of the connected components of Π.*

A word w is *(simply) Π-infeasible* if it is not (simply) Π-feasible.

A *cut* of a word W is an ordered partition of W in subwords. We will think on this partition as an ordered forest of words. Below we omit the term 'ordered'. A cut F is *Π-feasible* if every word of F is Π-feasible.

3.2 The Tester

The tester takes random subwords of finite length of W and will test feasibility for finitely many Π, that is at most 2^m where m is the number of state of the automaton. The Robustness lemma will insure that if a word W is far, then with high probability a random subword of finite length will be infeasible.

Tester for regular language $(\mathcal{A}, \varepsilon, W)$:

If the size n of W is less than $15m^2/\varepsilon$
  then simply evaluate $\mathcal{A}$ on W and accept iff $\mathcal{A}$ accepts W.
Else do the following:
For $i = 1, \ldots, \log(5m^2/\varepsilon)$ {
  Compute $N_i = \Theta(\frac{2^{-i} m^3 \log(m^2/\varepsilon)}{\varepsilon})$.
  Choose N_i random subwords w_j^i of W of size 2^{i+1}, for $j = 1, \ldots, N_i$ }
For every admissible path Π of $\widehat{G}(\mathcal{A})$ {
  If all the w_j^i are Π-feasible then accept W (and stop) }
Reject W.

Theorem 1. *For every real $\varepsilon > 0$, every automaton $\mathcal{A}$ with m states, and every word W, the algorithm* ***Tester for regular language*** *$(\mathcal{A}, \varepsilon, W)$ is an ε-tester for the language recognized by $\mathcal{A}$. Moreover, its query complexity is in $O(m^3 \log^2(m^2/\varepsilon)/\varepsilon)$, and its time complexity in $O(2^m m^3 \log^2(m^2/\varepsilon)/\varepsilon)$.*

Proof. We can assume w.l.o.g. that the size n of W is at least $15m^2/\varepsilon$, otherwise the proof of the correctness is obvious.

First, if $W \in L$ then W is Π-feasible for some admissible Π. Therefore every subword of W is Π-feasible for this path Π. Thus the tester accepts W with probability 1.

Suppose that $\mathsf{dist}(W, L) > \varepsilon n$ and fix an admissible path Π. Using the Robustness lemma (Lemma 1), we get that the probability to accept W for this Π is in $O(2^{-m})$. Since there is at most 2^m candidates Π, we can conclude, using the union bound, that the acceptance probability is upper bounded by 1/3. □

Corollary 1. *Regular properties of words are ε-testable.*

We now state the Robustness lemma. The notion of robustness was first defined in [9] and studied in [8]. In the rest of this section, we fix an automaton $\mathcal{A}$ and we call L its associated language.

Lemma 1 (Robustness). *Let $n \geq 15m^2/\varepsilon$, and let W be a word of size n such that $\mathsf{dist}(W, L) \geq \varepsilon n$. Then for every admissible path Π of $\widehat{G}(\mathcal{A})$, there exists an integer $1 \leq i \leq \log(5m^2/\varepsilon)$, such that the number of Π-infeasible subwords of size 2^{i+1} is at least $\frac{2^{i+1}}{90m^2 \log(5m^2/\varepsilon)} \times \varepsilon n$.*

The sketch of the proof of the Robustness lemma takes the following steps (see Figure 2):

1. The Splitting lemma shows that if the distance between W and L is large then there are many infeasible disjoint subwords. Its proof is by contraposition:
   a) First, from a cut of minimal infeasible subwords, we construct a close feasible cut F.
   b) Then the Merging lemma which shows that if a cut F is feasible, then it is close to L.
2. The Amplifying lemma shows that if there many infeasible words, then there are many short ones.

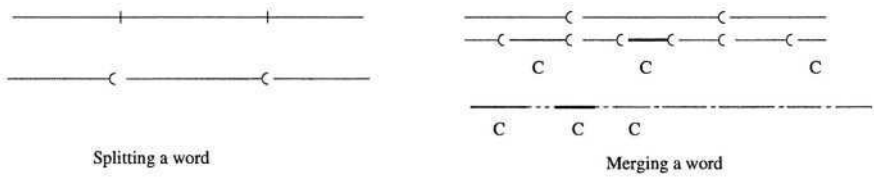


Fig. 2. The correction (steps 1.a and 1.b) of a word with two infeasible subwords where C is some connected components (and $h' = 3$ for the proof of Lemma 2).

3.3 Robustness of the Tester

Lemma 2 (Splitting). *Let Π be an admissible path of $\widehat{G}(\mathcal{A})$. Let W be a word such that $\mathsf{dist}(W, L) > h$. Then W has more than $\frac{h-3m^2}{2m^2}$ Π-infeasible disjoint subwords.*

Proof. The proof is by contraposition and we understand feasible as Π-feasible. First we construct a cut $\mathcal{P}$ of W of size h' whose $h'-1$ first subwords are minimal infeasible and disjoint subwords. The last subword of $\mathcal{P}$ is either infeasible or feasible. And in this last case, the entire word W might feasible and $h' = 1$.

We visit W from the left to the right and the construction of each infeasible subword $W[i, \ldots, j]$ is done by induction on that walk.

Initially: $h' = 0$ and $i = j = 1$.
While $(j \leq |W|)$ {
  While (subword $W[i, \ldots, j]$ is Π-feasible and $j < |W|$) {increase j}
  $h' = h' + 1$, $w_{h'} = W[i, \ldots, j]$, $i = j + 1$, $j = i$.}

At the end of the procedure we get the desired partition $\mathcal{P} = (w_i)_{1 \leq i \leq h'}$.

Now we explain how to get a word $W' \in L$. Let w'_i be w_i without the last letter, for $i = 1, \ldots, h'$. When $w_{h'}$ is feasible then $w'_{h'} = w_{h'}$. By construction of w_i, the subwords w'_i are feasible. Let F be the cut of the $(w'_i)_{1 \leq i \leq h'}$. Applying Lemma 3, we get that $\mathsf{dist}(F, L) \leq m + 2m^2 \times h'$. Because $\mathsf{dist}(W, F) \leq h'$, then $\mathsf{dist}(W, L) \leq m + 2m^2 \times h'$. But by assumption, $h' - 1 \leq \frac{h - 3m^2}{2m^2}$, therefore $\mathsf{dist}(W, L) \leq h$. □

Lemma 3 (Merging). *Let $\Pi = (C_1, \ldots, C_k)$ be an admissible path of $\widehat{G}(\mathcal{A})$. Let F be a Π-feasible cut of size h'. Then* $\mathsf{dist}(F, L) \leq m + 2m^2 h'$.

Proof. First, we split each subword of F in C-feasible subwords, for some $C \in \Pi$. Given a Π-feasible subword w which connects $p \in C_i$ to $q \in C_j$, we follow the automaton from p to q on w, and we delete each letter leading to a new connected component. Then the subword is cut along each deleted letter.

This technique keeps subwords that are C-feasible for some truly connected component C. Moreover, each initial subword of F splits in at most k subwords from which at most k letters are deleted, where k is less than m, where m is the number of state of the automaton. Let $(w_i)_{1 \leq i \leq l}$ be the remaining subwords of F, where $1 \leq l \leq m \times h'$.

Now we explain how to move and glue the remaining subwords w_i in order to get a subword $W' \in L$. Let C'_i be a component of Π such that w_i is C'_i-feasible. Let $p_i, q_i \in C'_i$ such that w_i connects p_i to q_i. Then, we do $(l - 1)$ moves so that the components C'_i are in the order defined by Π. Up to some renaming, we assume now that $(C'_1, \ldots, C'_l)$ are in the same order than $(C_1, \ldots, C_k)$, up to some repetitions.

We glue by induction. Let q_0 be an initial state of C_1, and let p_{l+1} be an accepting state of C_k. For $i = 0$ to $i = l$ do the following. By Proposition 1, let g_i be a word of size at most m that connects q_i to p_{i+1}. By inserting g_i between w_i and w_{i+1}, we get the world $W' = g_0.w_1.g_1 \ldots w_l.g_h$. By construction $W' \in L$. In this last step, we did at most $m \times (l + 1)$ insertions.

The total number of elementary operations is less than $(mh') + (l - 1) + (m(l + 1)) \leq m + 2m^2 \times h'$, since $l \leq mh'$ and $m \geq 2$. □

Lemma 4 (Amplifying). *Let Π be a path of $\widehat{G}(\mathcal{A})$. Let W be a word of length n with at least h' Π-infeasible disjoint subwords. Then there exists an integer $1 \leq i \leq \log(2n/h')$ such that the number of Π-infeasible subwords of size 2^{i+1} is at least $\frac{2^i(h' - 4)}{6 \log(2n/h')}$.*

Proof. In this proof, we understand feasible as Π-feasible.

Let $w_1, \ldots, w'_h$ be some infeasible disjoint subwords of W. Let a be a positive integer. For every integer $i \geq 1$, let $s_i = |\{w_j : 2^{i-1} + 1 \leq |w_j| \leq 2^i\}|$. Since we have $|\{w_j : |w_j| > 2^a\}| \leq \frac{n}{2^a}$, we therefore get $\sum_{i=1}^{a} s_i \geq h' - \frac{n}{2^a}$.

Take $a = \log(2n/h')$. Then $\sum_{i=1}^{a} s_i \geq \frac{h'}{2}$, thus there exists some $1 \leq i \leq a$ such that $s_i \geq \frac{h'}{2a}$.

To lower bound the number of infeasible subwords of size 2^{i+1}, we count the number of subwords of size 2^{i+1} that contains a least one subword w_j whose size is in $[2^{i-1}+1, 2^i]$. These subwords are also infeasible since they contain one of the infeasible subwords w_j. Note that since the subwords w_j are disjoint, each infeasible subword of length 2^{i+1} contains at most 3 of the w_j of length greater than 2^{i-1}. Moreover, each infeasible subword w_j of length at most 2^i is included in at least 2^i subwords of length 2^{i+1} (except, maybe, the two first and the two last subwords). We then get that the number of infeasible subwords of size 2^{i+1} is at least $\frac{2^i}{3} \times \frac{h'-4}{2a}$. □

Proof (of Lemma 1). From the Splitting lemma with $h = \varepsilon n$, the word W has more than $h' = \frac{2\varepsilon n}{5m^2}$ Π-infeasible disjoint subwords. Now, by the Amplifying lemma, there exists an integer $1 \leq i \leq \log(5m^2/\varepsilon)$ such that the number of Π-infeasible subwords of size 2^{i+1} is at least $\frac{2^i((2\varepsilon n/5m^2)-4)}{6\log(5m^2/\varepsilon)} \geq \frac{2^{i+1}}{90m^2\log(5m^2/\varepsilon)} \times \varepsilon n$. □

4 Testing Regular Ranked Tree Languages

4.1 Basic Definitions

A *r-ranked tree automaton* is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, (I_\sigma)_{\sigma\in\Sigma}, F)$ where Q is the set of states, $F \subseteq Q$ is the set of accepting states, $I_\sigma \subseteq Q$ the set of initial states for σ, and $\delta : Q^{\leq r} \times \Sigma \to Q$ is the transition function.

A *subtree* t of T takes a node v_1 of T as root and is the substructure restricted to nodes $\{v_1, \ldots, v_m\}$ where $v_2, \ldots, v_m$ are connected to v_1. The leaves of T among $\{v_1, \ldots, v_m\}$ are leaves of t, while some nodes are leaves in t but not in T and called $*$-nodes where the new label is $*$. By extension, a *subtree* t is a tree where some of the leaves are $*$-nodes.

An *assignment* λ for a subtree t determines states for its leaves such that if u is a leaf with label $l(u)$, then $\lambda(u) \in I_{l(u)}$. A *run* on a tree T extends λ on all the nodes of the subtree such that if u is a node with successors $v_1, \ldots, v_l$ where $l \leq r$ in states $\lambda(v_1), \ldots, \lambda(v_l)$ then $\lambda(u) = \delta(\lambda(v_1), \ldots, \lambda(v_l), l(u))$. A run *accepts* if the state of the root is in F.

Two states q and q' are *connected* if there exists a finite subtree t of size at most r^m and a run λ such that one leaf of t is assigned the state q, and the root of t is assigned the state q'. Let $G(\mathcal{A})$ be the directed graph whose vertices are the states of $\mathcal{A}$ and edges connect states that are connected by a subtree of depth 1.

We assume without lost of generality that $G(\mathcal{A})$ is connected. We define $\widehat{G}(\mathcal{A})$ and the notion of *truly connected* as in Section 3.1, and we omit the term 'strongly'. We consider a set Π of connected components of $G(\mathcal{A})$ and generalize the notions of *Π-feasibility* for subtrees.

Definition 5. *Let Π be a set of connected components of $G(\mathcal{A})$. Then Π is* admissible *if there is a pair (T_0, λ_0), the* witness *of Π, such that λ_0 is an assignment of the tree T_0 which visits every connected components Π, and no more.*

Observe that T_0 can be always chosen such that its size is at most r^m.

Definition 6. *Let Π be a set of connected components of $G(\mathcal{A})$. A path σ from a leaf to the root of a subtree t is* Π-feasible *if there exists a run which visits along σ only some connected components of Π. A subtree t is* simply Π-feasible *if there exists a path σ in T such that σ is Π-feasible. A subtree t is* Π-feasible *if there exists a run λ such that for all paths σ in t, σ is Π-feasible for λ.*

A subtree t is *(simply) Π-infeasible* if it is not (simply) Π-feasible.

We say that two subtrees of a tree T are *disjoint* if they are node disjoint except in one node that might be both a $*$-node of one subtree and the root of the other subtree. A *cut* of a tree T is a partial ordered partition of T in subtrees. We will think on this partition as an ordered forest of subtrees. A *forest* of subtrees is a partial ordered set of subtrees. Below we omit the term 'ordered'.

We naturally extend the Tree Edit distance (with moves) to forests, where the move operation can now either be applied to two subtrees of the forest or take one subtree and generate two new subtrees. Since the Tree Edit distance and the Tree Forest Edit distance are 4-equivalent (see Proposition 2), we do not distinguish them for the sake of simplicity. In other words, the Tree Forest Edit distance allows for some temporarily disconnection of complete subtrees.

Proposition 2. *If two trees T, T' have Tree Edit distance h then their Tree Forest Edit distance is in $[h/4, h]$.*

A forest of subtrees is *Π-feasible* if every subtree is Π-feasible.

4.2 The Tester

The tester generates random k-subtrees in the following way. A *k-subtree of T from v* is a subtree of T with v as a root and containing every nodes at distance at most k below v. The tester is going to select subtrees t_i^j, for $j = 1, \ldots, \Theta(\frac{mr}{\varepsilon^2})$, of depth i, for $i = 1, \ldots, r^{2m}/\varepsilon$, and check if they are all Π-feasible, for some admissible Π.

Tester for regular ranked tree language $(\mathcal{A}, \varepsilon, T)$**:**

If the size n of T is in $O(r^{2m+1}/\varepsilon)$
  then simply evaluate $\mathcal{A}$ on T and accept iff $\mathcal{A}$ accepts T.
Else do the following:
Compute $N = \Theta(mr^{4m+3}/\varepsilon^2)$
For $i = 1, \ldots, 2r^{2m+1}/\varepsilon$ {
  Choose N random nodes v_j^i, for $j = 1, \ldots, N$.
  Query the i-subtree t_j^i of T from v_j^i, for $j = 1, \ldots, N$ }
For every admissible set Π of connected component of $G(\mathcal{A})$ {
  If all the t_j^i are Π-feasible then accept T (and stop) }
Reject T.

Theorem 2. *For every real $\varepsilon > 0$, every r-ranked tree automaton $\mathcal{A}$ with m states, and every r-ranked tree T, the algorithm* ***Tester for regular ranked tree language*** *$(\mathcal{A}, \varepsilon, T)$ is an ε-tester for the language recognized by $\mathcal{A}$. Moreover, its query complexity is in $O(mr^{4m+3}2^{(r^{2m+1})/\varepsilon}/\varepsilon^2)$, and its time complexity in $O(2^m mr^{4m+3}2^{(r^{2m+1})/\varepsilon}/\varepsilon^2)$.*

Corollary 2. *Regular properties of trees are ε-testable.*

In the rest of this section, we fix an r-ranked automaton $\mathcal{A}$ and we call L its associated language. The proof of Theorem 2 follow the same arguments of Theorem 1 using the Robustness lemma for trees.

Lemma 5 (Robustness). *Let $n = \Omega(r^{2m+1}/\varepsilon)$, and let T be a r-ranked tree of size n such that $\mathsf{dist}(T, L) \geq \varepsilon n$. Then for every admissible set Π of connected components of $G(\mathcal{A})$, there exists an integer $1 \leq i \leq 2r^{2m+1}/\varepsilon$, such that the number of Π-infeasible i-subtrees is in $\Omega(\frac{1}{r^{4m+3}} \times \varepsilon^2 n)$.*

The structure of the proof of the Robustness lemma is the same than the one of Lemma 1 (see Figure 3).

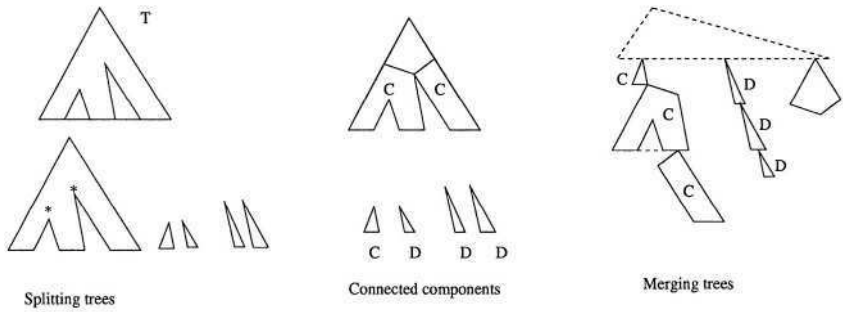


Fig. 3. The correction of a tree with two infeasible subtrees where we mention C and D as some connected components (and $h' = 3$ for the proof of Lemma 6).

4.3 Robustness of the Tester

In this section, all the trees we consider are r-ranked trees.

Lemma 6 (Splitting). *Let Π be an admissible set of connected components of the graph $G(\mathcal{A})$. Let T be a tree such that $\mathsf{dist}(T, L) > h$. Then T has more than $\frac{1}{3r^{m+1}}(\frac{h}{r^m} - 1) - 1$ Π-infeasible subtrees.*

Proof. The proof is by contraposition and we understand feasible as Π-feasible. First we construct a cut $\mathcal{P}$ of T of size h' whose $h' - 1$ last subtrees are minimal infeasible and disjoint subtrees. It might be the case that the top subtree of

$\mathcal{P}$ is Π-feasible. We visit T from the left to the right, and bottom-up. While visiting a node v, if the subtree below v is Π-infeasible, we add it in our cut and we consider v as a $*$-node in the remaining part of T. At the end of the procedure we get the desired cut $\mathcal{P} = (t_i)_{1 \leq i \leq h'}$, ordered as T and having at most h' $*$-node.

Now we explain how to get a tree $T' \in L$. Since t_i has a root of degree at most r, let $t_i^1, \ldots, t_i^r$ be the r subtrees from the root of t_i (some of them might be empty), for $i = 1, \ldots, h'$. By construction of t_i, the subtrees $t_i^1, \ldots, t_i^r$ are Π-feasible. When $t_{h'}$ is feasible then $t_{h'}^1 = t_{h'}$ and others $t_{h'}^j$ are empty. Let F be the forest $(t_i^1, \ldots, t_i^r)_{i=1,\ldots,h'}$ of size at most rh', in the same order than T. To get F from T, we use only rh' moves and h' edge deletions. Moreover F has at most h' $*$-nodes. Applying Lemma 7, we get that $\mathsf{dist}(F, L) \leq r^m(1 + h' + 2r^{m+1}h')$ and since $\mathsf{dist}(T, F) \leq (r+1)h'$, we conclude that $\mathsf{dist}(T, L) \leq r^m(1 + 3r^{m+1}h')$, majoring $(r+1) + r^m(1 + 2r^{m+1})$ by $r^m \times 3r^{m+1}$. But by assumption, $h' - 1 \leq \frac{1}{3r^{m+1}}(\frac{h}{r^m} - 1) - 1$, therefore $\mathsf{dist}(T, L) \leq h$. □

Lemma 7 (Merging). *Let Π be an admissible set of connected components of $G(\mathcal{A})$. Let F be a Π-feasible forest of size h_1' with at most h_2' $*$-nodes. Then* $\mathsf{dist}(F, L) \leq r^m(1 + h_2' + 2r^m h_1')$.

Proof. First, we split each subtree t of F in simply C-feasible subtrees, for some C of Π. Fix such a $t \in F$. Let λ be a run of t such that all paths of t are Π-feasible. Fix a path σ of t and let C be the connected component of the root of t. We follow σ top-down until we leave C after a node v. Then we cut t just before leaving C, that is between v and its successors using r edge deletions and r moves. This leads to one simply C-feasible subtree from the root of t where the label of v is now $*$, and r new Π-feasible subtrees from the successors of v. We iterate the argument for the last r subtrees using the restrictions of the same run λ, so that the next paths of the last r subtrees will start with the next connected component. At the end of the process, at most $1 + r + \ldots + r^{m-1} \leq r^m$ C-feasible subtrees are generated from t using r^m edge deletions and r^m moves.

We only consider subtrees that are simply C-feasible for some truly connected component C of Π, and delete the other ones, of size 1, using at most $r^m \times h_1'$ node deletions. Let $(t_i)_{1 \leq i \leq k}$ be the remaining subtrees of F, where $1 \leq k \leq r^m \times h_1'$.

We now explain how to move and glue the remaining subtrees w_i in order to get a tree $T' \in L$. Let C_i be a connected component of Π such that t_i is simply C_i-feasible. We first move and glue linearly each subtrees t_i with the same C_i. At each $*$-node, a tree of size r^m is also inserted so that the resulting subtree is simply C_i-feasible and without any $*$-nodes. Then the remaining subtrees are connected to T_0 in order to get a tree $T' \in L$. We have done $(k-1)$ moves and $r^m \times (k+1) + r^m \times h_2'$ insertions and the total number of operations is less than: $(2r^m \times h_1') + (r^m \times h_1') + (r^m \times h_1' + r^m \times (r^m \times h_1' + 1) + r^m \times h_2')$ which is less than $r^m + 2r^{2m} \times h_1' + r^m \times h_2'$. □

Lemma 8 (Amplifying). *Let Π be an admissible set of connected components of $G(\mathcal{A})$. Let T be a tree of size n with at least h' Π-infeasible disjoint subtrees.*

Then there exists an integer $1 \leq i \leq 2n/h'$ *such that the number of* Π*-infeasible* i*-subtrees is at least* $\frac{1}{r} \times \frac{h'}{4n} \times h'$.

Proof. In this proof, we understand feasible as Π-feasible and we follow the structure of the proof of Lemma 4.

Let $t_1, \ldots, t_{h'}$ be some infeasible disjoint subtrees of T. Let a be a positive integer. For every integer $i \geq 1$, let $s_i = |\{t_j : \text{depth}(t_j) = i\}|$. Since the root of a subtree may be shared a with the leaf of another subtree as a $*$-node, we have $|\{t_j : \text{depth}(t_j) > a\}| \leq \frac{n}{a+1}$, and therefore $\sum_{i=1}^{a} s_i \geq h' - \frac{n}{a}$. Take $a = \frac{2n}{h'}$. Then $\sum_{i=1}^{a} s_i \geq \frac{h'}{2}$, thus there exists some $1 \leq i \leq a$ such that $s_i \geq \frac{h'}{2a}$.

To lower bound the number of infeasible i-subtrees, we count the number of i-subtrees that contains a least one subtree t_j of depth i. These subtrees are also infeasible since they contain one of the infeasible subtrees t_j. Note that since the subtrees t_j are disjoint, each infeasible i-subtrees contains at most r of the t_j of depth i. Moreover, each infeasible subtree t_j of depth i is included in at least one infeasible i-subtree. We then get that the number of infeasible i-subtrees is at least $\frac{1}{r} \times \frac{h'}{2a}$. □

5 Extension to Unranked Trees

An *unranked tree automaton* generalizes the transition function to $\delta : Q \times \Sigma \to 2^{Q^*}$ such that $\delta(q, a)$ is a regular language on Q. A run λ is generalized such that if u is a node with successors $v_1, ..., v_l$ in states $\lambda(v_1), \ldots, \lambda(v_l)$ and there is a q such that $\lambda(v_1), \ldots, \lambda(v_l) \in \delta(q, l(u))$, then $\lambda(u) = q$.

We consider two approaches to generalize the **Tester for regular ranked tree language** to unranked regular trees. In a direct approach we are able to prove a Splitting lemma and a Merging lemma for any unranked tree automaton. The remaining main obstacle is the existence of an efficient random generator of subtrees for a corresponding Amplifying lemma.

Another possible approach consists to encode unranked trees T by binary trees $e(T)$ using a classical encoding to construct a binary automaton that accepts the encoded unranked trees accepted by the unranked automaton, and to apply the tester on binary trees. In case of XML files, assume they are given by their DOM (Document Object Model) structures. We can efficiently generate any random k-subtrees on the encoded tree from the DOM and simulate efficiently the **Tester for regular ranked tree language** on the encoded tree. There is a remaining obstacle consisting in lower bounding the distance of two encoded trees $\text{dist}(e(T), e(T'))$ by $\text{dist}(T, T')$. Even if it is clear that $\text{dist}(e(T), e(T')) \leq 2\text{dist}(T, T')$, the opposite inequality is rather technical.

References

1. N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20:451–476, 2000.

2. N. Alon, M. Krivelich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing,* 30(6), 2000.
3. M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM,* 42(1):269–291, 1995.
4. M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences,* 47(3):549–595, 1993.
5. H. Chockler and O. Kupferman. ω-regular languages are testable with a constant number of queries. In *Proceedings of the 6th Workshop on Randomization and Approximation Techniques in Computer Science,* pages 26–38, 2002. LNCS volume 2483.
6. G. Cormode. *Sequence Distance Embeddings.* PhD thesis, University of Warwick, 2003.
7. O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM,* 45(4):653–750, 1998.
8. R. Rubinfeld. On the robustness of functional equations. *SIAM Journal on Computing,* 28(6):1972–1997, 1999.
9. R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing,* 25(2):23–32, 1996.
10. K. Tai. The tree-to-tree correction problem. *Journal of the ACM,* 26:422–433, 1979.

Entropy as a Fixed Point

Keye Martin

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD
kmartin@comlab.ox.ac.uk
http://web.comlab.ox.ac.uk/oucl/work/keye.martin

Abstract. We present general ideas about the complexity of objects and how complexity can be used to define the information in objects. In essence, the idea is that while complexity is relative to a given class of processes, information is process independent: information is complexity relative to the class of all conceivable processes. We test these ideas on the complexity of classical states. A domain is used to specify the class of processes, and both qualitative and quantitative notions of complexity for classical states emerge. The resulting theory can be used to give new proofs of fundamental results from classical information theory, to give a new characterization of entropy, to derive lower bounds on algorithmic complexity and even to establish new connections between physics and computation. All of this is a consequence of the setting which gives rise to the *fixed point theorem:* The least fixed point of the copying operator above complexity is information.

1 Introduction

We can think of domains ([1][11]) as a qualitative way of reasoning about informative objects, and measurement ([6] [9]) as a way of determining the amount of information in an object. But neither set of ideas attempts to answer the question "What is information?" In this paper, we offer one possible answer to this question which has pragmatic value and is of interest to computer science.

To begin, we assume that the words 'complexity' and 'information' are just that – words. We start from a clean slate, forgetting the various connotations these words have in the sciences, and simply begin talking about them intuitively. We might say:

- The complexity of a *secret* is the amount of work required to *guess* it.
- The complexity of a *problem* is the amount of work required to *solve* it.
- The complexity of a *rocket* is the amount of work required to *escape gravity.*
- The complexity of a *probabilistic state* is the amount of work required to *resolve* it.

In all cases, there is a task we want to accomplish, and a way of measuring the work done by a process that *actually achieves* the task; such a process belongs to a prespecified class of processes which themselves are the stuff that science is meant to discover, study and understand. Then there are two points not to miss about complexity:

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 945–958, 2004.

(i) It is relative to a prespecified class of processes,
(ii) The use of the word 'required' necessitates the *minimization* of quantities like work over the class of processes.

Complexity is *process dependent.* Now, what is information in such a setting?

Information, in seeming stark contrast to complexity, is *process independent.* Here is what we mean: *Information is complexity relative to the class of all conceivable processes.* For instance, suppose we wish to measure the complexity of an object x with respect to several different classes $P_1, \ldots, P_n$ of processes. Then the complexity of x varies with the notion of process: It will have complexities $c_1(x), \ldots, c_n(x)$, where c_i is calculated with respect to the class P_i. However, because information is complexity relative to the class of *all* conceivable processes, the information in an object like x will *not* vary. That is what we mean when we say information is process independent: It is an element present in *all* notions of complexity. So we expect

$$\text{complexity} \geq \text{information}$$

if only in terms of the mathematics implied by the discussion above. For example, this might allow us to *prove* that the amount of work you expect to do in solving a problem always exceeds the a priori uncertainty (information) you have about its solution: The less you know about the solution, the more work you should expect to do. An inequality like the one above could be valuable.

To test these ideas, we study the complexity of classical states relative to a class of processes. A class of processes will be derived from a domain (D, μ) with a measurement μ that supports a new notion called *orthogonality.* Write $c_D(x)$ for the complexity of a classical state x relative to (D, μ). Then we will see that

$$\inf_{D \in \Sigma} c_D = \sigma \tag{1}$$

where σ is Shannon entropy and Σ is the class of domains (D, μ). This equation provides a setting where it is clear that information in the sense of the discussion above is σ, and that the class of all conceivable processes is Σ. By (1), our intuitive development of 'complexity' turns out to be capable of deriving lower bounds on the complexity of algorithms such as sorting and searching. Another limit also exists,

$$\bigcap_{D \in \Sigma} \leq_D = \leq \tag{2}$$

where $\leq_D$ is a relation on classical states which means $x \leq_D y$ iff for all processes p on (D, μ), it takes more work for p to resolve x than y. This is *qualitative complexity,* and the value of the intersection above $\leq$ just happens to be a relation called *majorization.* Muirhead [5] discovered majorization in 1903, and in the last 100 years his relation has found impressive applications in areas such as economics, computer science, physics and pure mathematics [2] [4]. We will see that majorization is a continuous dcpo on the subset of monotone classical states and that the complexity c_D is determined by its value on this subset.

The limits (1) and (2) comprise what we call *the universal limit,* because it is taken over the class of *all* domains. The pair $(\sigma, \leq)$ can also be derived on a *fixed* domain (D, μ) provided one has the ability to *copy* processes. The mathematics of copying necessitates the addition of algebraic structure $\otimes$ to domains (D, μ) already supporting orthogonality. It is from this setting, which identifies the essential mathematical structure required to execute classical information theory [12] over the class of semantic domains, that the *fixed point theorem* springs forth: As with recursive programs, the semantics of *information* can also be specified by a least fixed point:

$$\text{fix}(\Phi) = \bigsqcup_{n \geq 0} \Phi^n(\perp) = \sigma$$

where Φ is the copying operator and $\perp$ is the complexity c_D, i.e., the least fixed point of domain theory connects complexity in computer science to entropy in physics. We thus learn that one can use domains to define the complexity of objects in such a way that information becomes a concept derived from complexity in a precise and systematic manner (as a least fixed point). **Note: All proofs of unproved theorems are given in [7].**

2 Classical States

We begin with the objects whose complexity we wish to study. These are the classical states.

Definition 1. The set of *classical n-states* is

$$\Delta^n := \{x \in [0,1]^n : \sum_{i=1}^{n} x_i = 1\}$$

The set of *monotone decreasing n-states* is

$$\Lambda^n := \{x \in \Delta^n : (\forall i < n)\, x_i \geq x_{i+1}\}$$

for $n \geq 2$.

In 1903, Muirhead [5] discovered an important relation on classical states called *majorization.*

Definition 2. For $x, y \in \Lambda^n$, it is

$$x \leq y \equiv (\forall k)\, s^k x \leq s^k y,$$

where

$$s^k x := \sum_{i=1}^{k} x_i$$

for all $k \in \{0, \ldots, n\}$. Note that $s^0 x = 0$ for all $x \in \Lambda^n$.

In the last one hundred years, majorization has arisen in a number of contexts, including economics, computer science, physics and mathematics ([2][4]). It is a domain.

Theorem 1. *$(\Lambda^n, \leq)$ is a continuous dcpo with least element $\perp = (1/n, \ldots, 1/n)$.*

(i) *If (x_i) is an increasing sequence in Λ^n, then*

$$\bigsqcup_{i \geq 1} x_i = \lim_{i \to \infty} x_i$$

where the limit is in the Euclidean topology on Λ^n.

(ii) *For all $t < 1$, $\pi_{\perp x}(t) \ll x$, where $\pi_{\perp x}$ is the straight line path from $\perp$ to x.*

Basic domain theoretic ideas are given in [7]. We write

$$\langle x|y \rangle := \sum_{i=1}^{n} x_i \cdot y_i$$

for the standard inner product on $\mathbb{R}^n$.

Lemma 1. *For $x, y \in \Lambda^n$, we have $x \leq y$ if and only if for all increasing $a : \{1, \ldots, n\} \to [0, \infty)$, $\langle a|x \rangle \geq \langle a|y \rangle$.*

3 Processes from the Order on a Domain

To study processes which may result in one of several different outcomes, we have to know what 'different' means. This is what *orthogonality* does: It provides an order theoretic definition of 'distinct.' Let (D, μ) be a continuous dcpo with a measurement $\mu : D \to [0, \infty)^*$ and least element $\perp$. Recall that $[0, \infty)^*$ is the set of nonnegative reals in the order opposite to their usual one.

Definition 3. A pair of elements $x, y \in D$ are *orthogonal* if $\mu(\uparrow x \cap \uparrow y) \subseteq \{0\}$. This is written $x \perp y$.

Definition 4. By a *domain* (D, μ), we will mean a continuous dcpo D whose measurement $\mu \to \sigma_D$ satisfies $\mu\perp = 1$ and

$$\mu(\bigwedge F) \geq \sum_{x \in F} \mu x$$

for each finite set $F \subseteq D$ of pairwise orthogonal elements.

By replacing μ with $\mu/\mu\perp$ if necessary, we can always assume $\mu(\perp) = 1$. The inequality for pairwise orthogonal sets is worth comparing to its "opposite": That $\mu(x \sqcap y) \leq \mu x + \mu y$ if x and y are consistent. The latter allows one to derive metrics on $\ker \mu$ [8].

Lemma 2. *The closed subintervals of* [0,1] *with the length measurement,* $(\mathbf{I}[0,1],\mu)$, *form a domain in the sense of the previous definition.*

The following results give techniques for proving (D,μ) is a domain.

Lemma 3. *Let* $\phi : (D,\mu) \to (E,\mu)$ *be a monotone map with* $\mu\phi = \mu$ *which preserves orthogonality. If* (E,μ) *is domain, then* (D,μ) *is also a domain.*

Here is one reason ϕ might preserve orthogonality:

Proposition 1. *Let* $\phi : (D,\mu) \to (E,\mu)$ *be an order embedding with* $\mu\phi = \mu$ *whose image is dense in the Scott topology. If no compact element of D has measure zero, and each* $x \in E$ *with* $\mu x > 0$ *has* $\uparrow x \neq \emptyset$, *then*

$$x \perp y \Rightarrow \phi x \perp \phi y$$

for all $x, y \in D$. *Thus, if* (E,μ) *is a domain, then so is* (D,μ).

Example 1. Let $p \in \Delta^n$ be a classical state with all $p_k > 0$ and Σ^∞ the streams over the alphabet $\Sigma = \{0,\ldots,n-1\}$. Define $\mu : \Sigma^\infty \to [0,\infty)^*$ by $\mu\perp = 1$ and $\mu i = p_{i+1}$ for $i \in \Sigma$, and then extend it homomorphically by

$$\mu(s \cdot t) = \mu s \cdot \mu t$$

where the inner dot is concatenation of finite strings. The unique Scott continuous extension, which we call μ, yields a *domain* (D,μ).

We first embed (Σ^∞,μ) into $\mathbf{I}[0,1]$. Visualize an interval $x \in \mathbf{I}[0,1]$ as a line segment partitioned into n consecutive line segments having lengths $p_{i+1} \cdot \mu x$ for $0 \leq i \leq n-1$. Let $\phi_i(x)$ be the i^{th} such interval. The map $\phi : \Sigma^\infty \to \mathbf{I}[0,1]$ is

$$\phi(s) = \begin{cases} \perp & \text{if } s = \perp \\ \phi_i(\phi(t)) & \text{if } s = t \cdot i \end{cases}$$

Having defined ϕ on finite strings, we take its unique Scott continuous extension, and call this ϕ. It is an order embedding whose image is dense in the Scott topology because all $p_k > 0$. Now Prop. 1 applies.

An immediate corollary is the case $p = (1/2, 1/2) \in \Delta^2$ and $\Sigma = \{0,1\} = 2$, the binary streams with the usual measurement: $(2^\infty, 1/2^{|\cdot|})$ is a domain. This is the basis for the study of binary codes. The fact that it is a domain *implies* the vital *Kraft inequality* of classical information theory.

Theorem 2 (Kraft). *We can find a finite antichain of* Σ^∞ *which has finite word lengths* $a_1, a_2, \ldots, a_n$ *iff*

$$\sum_{i=1}^{n} \frac{1}{|\Sigma|^{a_i}} \leq 1.$$

Finite antichains of finite words are sometimes also called *instantaneous codes.* The inequality in Kraft's result can be derived as follows:

Example 2. The Kraft inequality. We apply the last example with

$$p = (1/|\Sigma|, \ldots, 1/|\Sigma|) \in \Delta^{|\Sigma|}.$$

A finite subset of $\Sigma^{<\infty}$ is pairwise orthogonal iff it is an antichain. Thus,

$$\mu(\bigwedge F) \geq \sum_{x \in F} \mu x.$$

In particular, $1 = \mu\perp \geq \mu(\bigwedge F)$, using the monotonicity of μ. Notice that the bound we derive on the sum of the measures is more precise than the one given in the Kraft inequality. We call μ the *standard measurement* and assume *it* when writing (Σ^{∞}, μ), unless otherwise specified.

Finally, the order theoretic structure of a domain (D, μ) gives rise to a notion of *process:* A set of outcomes which are (a) different, and (b) achievable in finite time.

Definition 5. A *process* on (D, μ) is a function $p : \{1, \ldots, n\} \to D$ such that $p_i \perp p_j$ for $i \neq j$ and $\mu p > 0$. $P^n(D)$ denotes the set of all such processes.

It is interesting to notice that $\mathbf{I}[0,1]$, like Σ^{∞}, also satisfies the converse to the Kraft inequality, i.e., the direction we did not prove. This direction permits us to characterize the vectors representable by processes on each of these domains.

Example 3. Processes on binary streams. The function $-\log\mu : P^n(D) \to (0, \infty)^n$ that takes a process $p \in P^n(D)$ to the vector

$$-\log \mu p = (-\log \mu p_1, \ldots, -\log \mu p_n)$$

produces positive vectors $a = -\log \mu p$ which by the orthogonality of $\mathrm{Im}(p)$ satisfy

$$\sum_{i=1}^{n} \frac{1}{2^{a_i}} \leq 1.$$

In the case of streams, a will also be *integer valued.* However, using the converse to the Kraft inequality, we can say that these vectors are *exactly* the image of $-\log \mu$. That is, any such integer valued vector a can be represented by a process on the domain of binary streams. For $\mathbf{I}[0,1]$ we get all positive vectors obeying the Kraft inequality.

We will now use this notion of process to define the complexity of classical states. Two notions arise: A quantitative measure, called h_D, and a qualitative measure, $\leq_D$, which takes the form of a relation on classical states Λ^n.

4 Complexity (Quantitative)

By considering processes on $(2^\infty, \mu)$, it is clear that the expected work done by an algorithm which takes one of n different computational paths $p : \{1, \ldots, n\} \to D$ is $\langle -\log \mu p | x \rangle$. Thus, the complexity of a state $c : \Delta^n \to [0, \infty)^*$ is

$$c(x) := \inf\{\langle -\log \mu p | x \rangle : p \in P^n(D)\}.$$

The function sort^+ reorders the components of a vector so that they increase; its dual sort^- reorders them so that they decrease. The first major step is to prove that the complexity of a classical state does not depend on the order of the probabilities within it:

Proposition 2. *For all* $x \in \Delta^n$,

$$c(x) = \inf\{\langle \mathrm{sort}^+(-\log \mu p) | \mathrm{sort}^-(x) \rangle : p \in P^n(D)\}.$$

In particular, the function c *is symmetric.*

So we can restrict our attention to monotone decreasing states Λ^n.

Definition 6. The *expectation* of a process $p \in P^n(D)$ is $\langle p \rangle : \Lambda^n \to [0, \infty)^*$ given by

$$\langle p \rangle x = \langle \mathrm{sort}^+(-\log \mu p) | x \rangle.$$

If the outcomes of process p are distributed as $x \in \Lambda^n$, then the work we expect p will do when taking one such computational path is $\langle p \rangle x$. And finally:

Definition 7. The *complexity* of a state $h : \Lambda^n \to [0, \infty)^*$ is

$$h(x) = \inf\{\langle p \rangle x : p \in P^n(D)\}.$$

Thus, the relation of h to c is that $c(x) = h(\mathrm{sort}^-(x))$ for all $x \in \Delta^n$. The *Shannon entropy* $\sigma : \Delta^n \to [0, \infty)$

$$\sigma x := -\sum_{i=1}^{n} x_i \log x_i$$

can also be viewed as a map on Λ^n, and as a map on *all* monotone states. Its type will be clear from the context.

Lemma 4. *If* $a : \{1, \ldots, n\} \to (0, \infty)$ *is a vector, there is a unique classical state* $y \in \Delta^n$ *such that*

$$\langle a | y \rangle - \sigma y = \inf\{\langle a | x \rangle - \sigma x : x \in \Delta^n\}.$$

The state y *is given pointwise by* $y_i = 2^{-a_i}/Za$ *and satisfies*

$$\langle a | y \rangle - \sigma y = -\log Za$$

where $Za := \sum_{i=1}^{n} \frac{1}{2^{a_i}}$. *In addition, if* a *is increasing, then* $y \in \Lambda^n$.

This lemma will be extremely valuable to us. (It's the existence and uniqueness of the equilibrium state associated to energy observable a from thermodynamics.)

Proposition 3. *If (D, μ) is a domain, then the complexity $h_D : (\Lambda^n, \leq) \to [0, \infty)^*$ is Scott continuous and satisfies $h_D \geq \sigma$ where σ is entropy.*

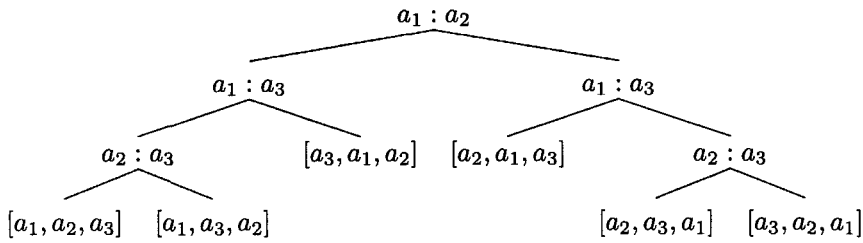
Proof. The continuity of h is in [7]. For $h_D \geq \sigma$, given a process $p \in P^n(D)$, the vector $a = \mathrm{sort}^+(-\log \mu p) : \{1, \ldots, n\} \to (0, \infty)$ satisfies

$$Za = \sum_{i=1}^{n} \mu p_i \leq \mu(\bigwedge \mathrm{Im}(p)) \leq \mu\bot = 1$$

where we appeal to the pairwise orthogonality of $\mathrm{Im}(p)$. Then by Lemma 4, using $-\log Z(a) \geq 0$, we have $\langle p \rangle x = \langle a | x \rangle \geq \sigma x$, and since $h_D(x)$ is the infimum of such terms, $h_D(x) \geq \sigma x$. Thus, $h_D \geq \sigma$. □

We have now *proven* the following: The amount of work we expect to do when solving a problem exceeds our a priori uncertainty about the solution. That is, the less you know about the solution, the more work you should expect to do:

Example 4. Lower bounds on algorithmic complexity. Consider the problem of sorting lists of n objects by comparisons. Any algorithm which achieves this has a binary decision tree. For example, for lists with three elements, a_1, a_2, a_3, it is



where a move left corresponds to a decision $\leq$, while a move right corresponds to a decision $>$. The leaves of this tree, which are labelled with lists representing potential outcomes of the algorithm, form an antichain of $n!$-many finite words in 2^∞ using the correspondence $\leq \mapsto 0$ and $> \mapsto 1$. This defines a process $p : \{1, \ldots, n!\} \to 2^\infty$. If our knowledge about the answer is $x \in \Lambda^{n!}$, then

$$\begin{aligned} \text{avg. comparisons} &= \langle -\log \mu p | x \rangle \\ &\geq \langle p \rangle (\mathrm{sort}^- x) \\ &\geq h(\mathrm{sort}^- x) \\ &\geq \sigma x. \end{aligned}$$

Assuming complete uncertainty about the answer, $x = \bot$, we get

$$\text{avg. comparisons} \geq \sigma\bot = \log n! \approx n \log n.$$

In addition, we can derive an entirely *objective conclusion:* In the *worst case,* we must do at least

$$\max(-\log \mu p) \geq \langle p \rangle \perp \geq \sigma \perp \approx n \log n$$

comparisons. Thus, sorting by comparisons is in general at least $O(n \log n)$. A similar analysis shows that searching by comparison is at least $O(\log n)$.

We have used *domain theoretic structure* as the basis for a new approach to counting the number of leaves in a binary tree. Different domains can give rise to different complexity classes, for the simple reason that changing the order changes the notion of process. An example of this is $(L, \mu) \subseteq (2^\infty, \mu)$ which models linear search (Example 6).

5 Complexity (Qualitative)

Each domain (D, μ), because it implicitly defines a notion of process, provides an intuitive notion of what it means for one classical state to be more complex than another: x is more complex than y iff for all processes $p \in P^n(D)$, the work that p does in resolving x exceeds the work it does in resolving y. This is *qualitative complexity.*

Definition 8. For $x, y \in \Lambda^n$, the relation $\leq_D$ is

$$x \leq_D y \equiv (\forall p \in P^n(D))\ \langle p \rangle x \geq \langle p \rangle y.$$

Only one thing is clear about $\leq_D$: The qualitative analogue of Prop. 3.

Lemma 5. *For each domain* (D, μ), $\leq \subseteq \leq_D$.

The calculation of $\leq_D$ requires knowing more about the structure of D. We consider domains whose orders allow for the simultaneous description of *orthogonality* and *composition.* In the simplest of terms: These domains allow us to say what *different* outcomes are, and they allow us to form *composite* outcomes from pairs of outcomes.

Definition 9. A domain (D, μ) is *symbolic* when it has an associative operation $\otimes : D^2 \to D$ such that $\mu(x \otimes y) = \mu x \cdot \mu y$ and

$$x \perp u \text{ or } (x = u \ \&\ y \perp v) \Rightarrow x \otimes y \perp u \otimes v$$

for all $x, y, u, v \in D$.

Notice that $\otimes$ has a qualitative axiom and a quantitative axiom. One example of a symbolic domain is (Σ^∞, μ) for an alphabet Σ with $\otimes$ being concatenation.

Example 5. The tensor on $\mathbf{I}[0,1]$ is

$$[a, b] \otimes [y_1, y_2] = [a + y_1 \cdot (b - a), a + y_2 \cdot (b - a)].$$

$(\mathbf{I}[0,1], \otimes)$ is a monoid with $\perp \otimes x = x \otimes \perp = x$ and the measurement μ is a homomorphism! We can calculate zeroes of real-valued functions by repeatedly tensoring $\text{left}(\perp) = [0, 1/2]$ and $\text{right}(\perp) = [1/2, 1]$, i.e., the bisection method.

We can tensor processes too: If $p : \{1, \ldots, n\} \to D$ and $q : \{1, \ldots, m\} \to D$ are processes, then $p \otimes q : \{1, \ldots, nm\} \to D$ is a process whose possible actions are $p_i \otimes q_j$, where p_i is any possible action of p, and q_j is any possible action of q. The exact indices assigned to these composite actions for our purposes is immaterial. We can characterize qualitative complexity on symbolic domains:

Theorem 3. *Let $(D, \otimes, \mu)$ be a symbolic domain. If there is a binary process $p : \{1, 2\} \to D$, then the relation $\leq_D = \leq$.*

6 The Universal Limit

We now see that $\leq$ and σ are two sides of the same coin: The former is a qualitative limit; the latter is a quantitative limit. Each is taken over the class of domains.

Theorem 4. *Let $\sigma : \Lambda^n \to [0, \infty)^*$ denote Shannon entropy and Σ denote the class of domains. Then*

$$\inf_{D \in \Sigma} h_D = \sigma$$

and

$$\bigcap_{D \in \Sigma} \leq_D = \leq$$

where the relation $\leq$ on Λ^n is majorization.

Corollary 1. *Shannon entropy $\sigma : (\Lambda^n, \leq) \to [0, \infty)^*$ is Scott continuous.*

By Theorem 4, the optimum value of $(h_D, \leq_D)$ is $(\sigma, \leq)$. But when does a domain have a value of $(h_D, \leq_D)$ that is close to $(\sigma, \leq)$? Though it is subtle, if we look at the case when $\leq_D$ achieves $\leq$ in the proof of Theorem 3, we see that a strongly contributing factor is the ability to *copy* processes – we made use of this idea when we formed the process $\bigotimes_{i=1}^{n} p$. We will now see that the ability to copy on a given domain *also guarantees* that h is close to σ.

7 Inequalities Relating Complexity to Entropy

We begin with some long overdue examples of complexity. It is convenient on a given domain (D, μ) to denote the complexity in dimension n by $h_n : \Lambda^n \to [0, \infty)$.

Example 6. Examples of h.

(i) On the lazy naturals $(L, \mu) \subseteq (2^\infty, \mu)$, where the L is for linear,

$$h_n(x) = x_1 + 2x_2 + \ldots + (n-1)x_{n-1} + (n-1)x_n$$

which is the average number of comparisons required to find an object among n using linear search.

(ii) On the domain of binary streams $(2^\infty, \mu)$,

$$h_2(x) \equiv 1$$

$$h_3(x) = x_1 + 2x_2 + 2x_3 = 2 - x_1$$

$$h_4(x) = \min\{2, x_1 + 2x_2 + 3x_3 + 3x_4\} = \min\{2, 3 - 2x_1 - x_2\}$$

In general, $h_n(x)$ is the average word length of an optimal code for transmitting n symbols distributed according to x.

(iii) On $(\mathbf{I}[0,1], \mu)$, $h_n(x) = -\sum_{i=1}^n x_i \log x_i$, Shannon entropy.

These examples do little to help us understand the relation of h to σ. What we need is some math. For each integer $k \geq 2$, let

$$c(k) := \inf\{\max(-\log \mu p) : p \in P^k(D)\}.$$

Intuitively, over the class $P^k(D)$ of algorithms with k outputs, $c(k)$ is the worst case complexity of the algorithm whose worst case complexity is *least.*

Theorem 5. *Let $(D, \otimes, \mu)$ be a symbolic domain with a process $p \in P^k(D)$. Then*

$$\sigma \leq h \leq \frac{c(k)}{\log k} \cdot (\log k + \sigma)$$

where h and σ can be taken in any dimension.

The mere existence of a process on a *symbolic* domain (D, μ) means not only that $\leq_D = \leq$ but also that h and σ are of the same order. Without the ability to 'copy' elements using $\otimes$, h and σ can be very different: Searching costs $O(n)$ on L, so h_L and σ are not of the same order. We need a slightly better estimate.

Definition 10. If $(D, \otimes, \mu)$ is a symbolic domain, then the integer

$$\inf\{k \geq 2 : c(k) = \log k\}$$

is called the *algebraic index* of (D, μ) when it exists.

By orthogonality, $c(k) \geq \log k$ always holds, so to calculate the algebraic index we need only prove $c(k) \leq \log k$. The value of the index for us is that:

Corollary 2. *If $(D, \otimes, \mu)$ is a symbolic domain with algebraic index $k \geq 2$, then*

$$\sigma \leq h \leq \log k + \sigma$$

where h and σ can be taken in any dimension.

There are results in [7] which explain why the algebraic index is a natural idea, but these use the *Gibbs map* and *partition function* from thermodynamics, which we do not have the space to discuss. But, it is simple to see that the algebraic index of $\mathbf{I}[0,1]$ is 2, the algebraic index of Σ^∞ is $|\Sigma|$ and in general, if there is a process $p \in P^n(D)$ on a symbolic domain with $(\mu p_1, \ldots, \mu p_n) = \perp \in \Lambda^n$ for some n, then D has an algebraic index $k \leq n$.

8 The Fixed Point Theorem

Let Λ be the set of *all* monotone decreasing states and let $\otimes : \Lambda \times \Lambda \to \Lambda$ be

$$x \otimes y := \text{sort}^{-}(x_1 y, \ldots, x_n y).$$

That is, given $x \in \Lambda^n$ and $y \in \Lambda^m$, we multiply any x_i by any y_j and use these nm different products to build a vector in Λ^{nm}.

Definition 11. The copying operator $! : X \to X$ on a set X with a tensor $\otimes$ is

$$!x := x \otimes x$$

for all $x \in X$.

If $p \in P^n(D)$ is a process whose possible outputs are distributed as $x \in \Lambda^n$, then two independent copies of p considered together as a single process $!p$ will have outputs distributed according to $!x$. Now let $[\Lambda \to [0,\infty)^*]$ be the dcpo with the pointwise order $f \sqsubseteq g \equiv (\forall x)\, f(x) \geq g(x)$.

Theorem 6. *Let $(D, \otimes, \mu)$ be a symbolic domain whose algebraic index is $k \geq 2$. Then the least fixed point of the Scott continuous operator*

$$\Phi : [\Lambda \to [0,\infty)^*] \to [\Lambda \to [0,\infty)^*]$$

$$\Phi(f) = \frac{f!}{2}$$

on the set $\uparrow (h + \log k)$ is

$$\text{fix}(\Phi) = \bigsqcup_{n \geq 0} \Phi^n(h + \log k) = \sigma,$$

where $h : \Lambda \to [0,\infty)$ is the complexity on all states.

Proof. By Corollary 2, let $k \geq 2$ satisfy $\sigma \leq h \leq \log k + \sigma$. First, we prove $h + \log k \sqsubseteq \Phi(h + \log k)$. Note that $\sigma \leq h!/2 \leq \log k/2 + \sigma$ using $\sigma = \sigma!/2$. Then

$$\left(h + \frac{\log k}{2}\right) - \left(\frac{h!}{2}\right) \geq \left(\sigma + \frac{\log k}{2}\right) - \left(\frac{\log k}{2} + \sigma\right) = 0$$

and this is exactly the statement that $h + \log k \sqsubseteq \Phi(h + \log k)$. Thus, Φ has a least fixed point on $\uparrow (h + \log k)$ given by $\text{fix}(\Phi)$. Since $\sigma = \Phi(\sigma) \in \uparrow (h + \log k)$, we must have $\text{fix}(\Phi) \sqsubseteq \sigma$. However, we also have

$$\sigma + \frac{\log k}{2^n} \sqsubseteq \Phi^n(h + \log k) \sqsubseteq \text{fix}(\Phi)$$

and since this holds for all n, we get $\sigma \sqsubseteq \text{fix}(\Phi)$. This proves $\text{fix}(\Phi) = \sigma$. □

This iterative process is very sensitive to where one begins. First, Φ has many fixed points above σ: Consider $c \cdot \sigma$ for $c < 1$. Thus, Φ cannot be a contraction on any subset containing $\uparrow h$. But Φ also has fixed points *below* σ: The map $f(x) = \log \dim(x) = \sigma \perp_{\dim(x)}$ is one such example. This proves that σ is genuinely a *least* fixed point.

The fixed point theorem can be used to derive Shannon's noiseless coding theorem [7]. In the proof of Theorem 6, we can regard Λ a continuous dcpo by viewing it as a disjoint union of domains. But we could just view it as a set. And if we do, the function space is still a dcpo, the theorem remains valid, and we obtain a new characterization of entropy:

Corollary 3. *Let $(D, \otimes, \mu)$ be a symbolic domain with algebraic index $k \geq 2$. Then there is a greatest function $f : \Lambda \to [0, \infty)$ which satisfies $h \geq f$ and $f(x \otimes x) \geq f(x) + f(x)$. It is Shannon entropy.*

The question then, "Does h approximate σ, or is it σ which approximates h" is capable of providing one with hours of entertainment. In closing, we should mention that Φ might also provide a systematic approach to defining information $\mathrm{fix}(\Phi)$ from complexity h in situations more general than symbolic domains.

9 The Quantum Case

The fixed point theorem also holds for quantum states where one replaces σ by von Neumann entropy, and $\otimes$ on domains by the algebraic tensor $\otimes$ of operators. (The domain theoretic tensor can also be mapped homomorphically onto the tensor of quantum states in such a way that domain theoretic orthogonality implies orthogonality in Hilbert space.) Several new connections emerge between computer science and quantum mechanics whose proofs combine new results with work dating as far back as Schrödinger [10] in 1936. The bridge that connects them is domain theory and measurement. One such result proves that reducing entanglement by a technique called local operations and classical communication is equivalent to simultaneously reducing the average case complexity of all binary trees, a major application of Theorem 3 that we could not include in this paper due to space limitations. These and related results are in [7].

References

1. S. Abramsky and A. Jung. *Domain theory.* In S. Abramsky, D. M. Gabbay, T. S. E. Maibaum, editors, Handbook of Logic in Computer Science, vol. III. Oxford University Press, 1994.
2. P. M. Alberti and A. Uhlmann. *Stochasticity and partial order: doubly stochastic maps and unitary mixing.* Dordrecht, Boston, 1982.
3. L. G. Kraft. *A device for quantizing, grouping and coding amplitude modulated pulses.* M.S. Thesis, Electrical Engineering Department, MIT, 1949.
4. A. W. Marshall and I. Olkin. *Inequalities: Theory of majorization and its applications.* Academic Press Inc., 1979.

5. R. F. Muirhead. *Some methods applicable to identities and inequalities of symmetric algebraic functions of n letters.* Proc. Edinburgh Math. Soc., 21:144-157, 1903.
6. K. Martin. *A foundation for computation.* Ph.D. Thesis, Department of Mathematics, Tulane University, 2000.
7. K. Martin. *Entropy as a fixed point.* Oxford University Computing Laboratory, Research Report PRG-RR-03-05, February 2003, `http://web.comlab.ox.ac.uk/oucl/publications/tr/rr-03-05.html`
8. K. Martin. *A triangle inequality for measurement.* Applied Categorical Structures, Vol. 11, No. 1, 2003.
9. K. Martin. *The measurement process in domain theory.* Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, Vol. 1853, Springer-Verlag, 2000.
10. E. Schrödinger. Proceedings of the Cambridge Philosophical Society **32,** 446 (1936).
11. D. Scott. *Outline of a mathematical theory of computation.* Technical Monograph PRG-2, Oxford University Computing Laboratory, November 1970.
12. C. E. Shannon. *A mathematical theory of communication.* Bell Systems Technical Journal 27, 379–423 and 623–656, 1948.

Transparent Long Proofs: A First PCP Theorem for $\mathrm{NP}_{\mathbb{R}}$

K. Meer*

Department of Mathematics and Computer Science
Syddansk Universitet, Campusvej 55, 5230 Odense M, Denmark

Abstract. We introduce and study the notion of probabilistically checkable proofs for real number algorithms. Our starting point is the computational model of Blum, Shub, and Smale and the real analogue $\mathrm{NP}_{\mathbb{R}}$ of NP in that model. Our main result is, to the best of our knowledge, the first PCP theorem for $\mathrm{NP}_{\mathbb{R}}$. It states $\mathrm{NP}_{\mathbb{R}} \subseteq \mathrm{PCP}_{\mathbb{R}}(poly, O(1))$. The techniques used extend ideas from [7] for self-testing and -correcting certain functions over so-called rational domains to more general domains over the real numbers. Thus, independently from real number complexity theory, the paper can be seen as a contribution to constructing self testers and correctors for linear functions over real domains.

1 Introduction

One of the most striking results of the last decade in theoretical computer science is the PCP theorem, [1,2]. It gives a characterization of the complexity class NP in terms of so-called probabilistically checkable proofs. In this paper we want to investigate similar questions for the real number model introduced by Blum, Shub, and Smale, see [4]. So far, neither approximation classes nor probabilistically checkable proofs have been defined and studied in that model. There is only one work dealing with interactive protocols over the reals, see [6]. In the present paper we want to start this research by giving a first non-trivial PCP theorem for the class $\mathrm{NP}_{\mathbb{R}}$.

Our main result, formally stated as $\mathrm{NP}_{\mathbb{R}} \subseteq \mathrm{PCP}_{\mathbb{R}}(poly, O(1))$, shows that each problem in $\mathrm{NP}_{\mathbb{R}}$ admits a verifier that produces polynomially many bits and afterwards inspects a constant number of proof-components. Since polynomially many random bits result in exponentially long proofs the result establishes the existence of transparent long proofs for $\mathrm{NP}_{\mathbb{R}}$. The full analogue to the classical PCP theorem, i.e. whether $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), O(1))$, remains a challenging open problem.

The major problems in proving our main theorem arise from the domains we have to deal with. Our proof techniques rely on ideas present in [7] and extend them to particular real number domains.

* partially supported by the EU Network of Excellence PASCAL and by the Danish Natural Science Research Council SNF.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 959–970, 2004.

2 Basic Notions; Verifiers and PCP Classes over the Reals

We assume the reader's familiarity with real number complexity theory [4].

Definition 1. *(Verifiers) Let $r, q : \mathbb{N} \mapsto \mathbb{N}$ be two functions. A $(r(n), q(n))$-restricted verifier V in the BSS model is a particular randomized real number algorithm working as follows. For an input $x \in \mathbb{R}^* := \bigcup_{n=1}^{\infty} \mathbb{R}^n$ of algebraic size $size_{\mathbb{R}}(x) := n$ and another vector $y \in \mathbb{R}^*$ representing a potential membership proof of x in a certain language, the verifier first produces a sequence of $r(n)$ many random bits (under the uniform distribution on $\{0,1\}^{r(n)}$). Given x and these $r(n)$ many random bits V computes in deterministic polynomial time the indices of $q(n)$ many components of y. Finally, V uses the input x together with the values of the chosen components of y in order to perform a deterministic polynomial time algorithm (in the BSS model). At the end of this algorithm V either accepts or rejects x. We denote by $V(x, y, \rho)$ the result of V supposed the random sequence generated for input (x, y) was ρ.*

Definition 2. *($\mathrm{PCP}_{\mathbb{R}}$ classes) Let $r, q : \mathbb{N} \mapsto \mathbb{N}$; a real number decision problem $L \subseteq \mathbb{R}^*$ is in class $\mathrm{PCP}_{\mathbb{R}}(r(n), q(n))$ iff there exists a $(r(n), q(n))$-restricted verifier V such that conditions i) and ii) below hold:*

i) For all $x \in L$ there is a $y \in \mathbb{R}^*$ such that for all randomly generated strings $\rho \in \{0,1\}^{r(size_{\mathbb{R}}(x))}$ the verifier accepts: $\Pr_{\rho}\{V(x, y, \rho) = 'accept'\} = 1$.

ii) For any $x \notin L$ and for each $y \in \mathbb{R}^*$ $\Pr_{\rho}\{V(x, y, \rho) = 'reject'\} \geq \frac{1}{2}$.

The probability is the uniform one over $\{0,1\}^{r(size_{\mathbb{R}}(x))}$.

Obviously, $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(0, poly)$. Our main theorem reads

Theorem 1. $\mathrm{NP}_{\mathbb{R}} \subseteq \mathrm{PCP}_{\mathbb{R}}(poly, O(1))$.

The theorem, to the best of our knowledge, is the first non-trivial PCP theorem for the class $\mathrm{NP}_{\mathbb{R}}$. Though the basic proof idea follows known lines, a lot of difficulties occur due to the presence of real domains over which these ideas have to be worked out. These domains are much more general than the finite field $\mathbb{Z}_2^n$ in that they contain arbitrary real numbers generated by a problem instance $x \in \mathbb{R}^*$. This forces us to generalize ideas used in the area of self-testing functions to larger domains. The paper [7] will be of special importance in our approach. Our results can be seen as well as a generalization of parts of the latter.

It should be pointed out that the classical PCP theorem is stronger than our Main Theorem by characterizing NP as $\mathrm{PCP}(O(\log(n)), O(1))$. Actually, our theorem is the real version of Theorem 5 in [1] which is a major ingredient for proving the full PCP theorem. It remains a challenge to investigate whether we have $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log(n)), O(1))$ over $\mathbb{R}$.

3 The Problem Setting: Outline of Proof Ideas

In order to prove our PCP theorem we start as in the classical setting. We closely follow the description in [3], Chapter 7.1 in order to point out significant differences between the classical and the real number setting. We want to construct a verifier for the following problem.

Definition 3. *The QPS (Quadratic Polynomial Systems) decision problem is:*
INPUT: $n, m \in \mathbb{N}$, *a set of* m *real polynomials* $p_1, \ldots, p_m \in \mathbb{R}[x_1, \ldots, x_n]$ *of degree at most* 2; *moreover, each* p_i *depends on at most* 3 *variables.*
QUESTION: *Is there an* $a \in \mathbb{R}^n$ *such that* $p_1(a) = 0\ ,\ \ldots\ ,\ p_s(a) = 0$?

The QPS problem is $\mathrm{NP}_{\mathbb{R}}$-complete [4]. Moreover, without loss of generality we can assume $m = O(n)$ (by adding dummy variables).

Now consider a random vector $\tilde{r} = (\tilde{r}_1, \ldots, \tilde{r}_m) \in \{0,1\}^m$ ($\tilde{r}$ later on will become a part of the entire string of random bits the verifier generates). Define a polynomial $P(x, \tilde{r})$ by $P(x, \tilde{r}) := \sum_{i=1}^{m} p_i^2(x) \cdot \tilde{r}_i$. Polynomial P has degree 4 in the x-variables and satisfies

- $P(x, \tilde{r}) \geq 0 \ \forall\ x \in \mathbb{R}^n, \tilde{r} \in \{0,1\}^m$;
- $P(a, \tilde{r}) = 0$ for all $\tilde{r} \in \{0,1\}^m$ iff $a \in \mathbb{R}^n$ is a solution of the QPS instance.

If $a \in \mathbb{R}^n$ is not a solution of the QPS instance, then picking uniformly a random $\tilde{r} \in \{0,1\}^m$ gives with probability at least $\frac{1}{2}$ a result $P(a, \tilde{r}) > 0$.

Most important for what follows will be the structure of the polynomial $P(x, \tilde{r})$. Due to the real coefficients present in the p_i's this structure is more complicated than in the classical PCP-proof (compare [3]).
Fix an $a \in \mathbb{R}^n$. Then $P(a, \tilde{r})$ can be written as

$$P(a, \tilde{r}) = E(\tilde{r}) + A \circ L_A(\tilde{r}) + B \circ L_B(\tilde{r}) + C \circ L_C(\tilde{r}) + D \circ L_D(\tilde{r})\ ,$$

where the different terms have the following properties: *A, B, C,* and *D* are linear functions with n, n^2, n^3, and n^4 many inputs, respectively. The coefficient matrices that represent these mappings depend on the chosen a, only. More precisely,

$$A : \mathbb{R}^n \mapsto \mathbb{R}\ ,\ A(x_1, \ldots, x_n) = \sum_{i=1}^{n} a_i \cdot x_i \ \forall\ x \in \mathbb{R}^n;$$
$$B : \mathbb{R}^{n^2} \mapsto \mathbb{R}\ ,\ B(y_1, \ldots, y_{n^2}) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j \cdot y_{ij} \ \forall\ y \in \mathbb{R}^{n^2}$$
(where y_{ij} denotes the argument $(i-1)n + j$);

and similarly for *C* and *D*. The functions $L_A, \ldots, L_D$ are linear. They take as arguments inputs $\tilde{r} \in \mathbb{Z}_2^m$ and give a result in the spaces $\mathbb{R}^n, \mathbb{R}^{n^2}, \mathbb{R}^{n^3}$, and $\mathbb{R}^{n^4}$, respectively. It is important to note that these mappings do only depend on the coefficients of the polynomials $p_1^2, \ldots, p_m^2$ (and thus on those of the p_i's), but

not on a. The mapping $E : \mathbb{Z}_2^m \mapsto \mathbb{R}$ as well is linear and only depends on the coefficients of the p_i's.

The main difference of this set-up in comparison to the classical setting is the presence of the mappings $L_A, \ldots, L_D$. Instead of binary numbers they produce real numbers as results when evaluated for a binary vector $\tilde{r}$. We therefore have to enlarge the set on which linearity has to be checked. There are results generalizing the self-testing and self-correcting algorithms over $\mathbb{Z}_2$ to larger domains. Rubinfeld and Sudan [7] extended a technique by Gemmell et al. [5] to deal with so-called rational domains. Though these ideas turn out to be very useful for us as well, our setting is different in that the real domains that occur in our framework are more general than rational domains. In particular, they are less structured. Thus, the first problem is to find the appropriate real domains. Then, checking linearity will become more involved, too. Whereas over $\mathbb{Z}_2$ only additivity has to be checked (i.e. $f : \mathbb{Z}_2^n \mapsto \mathbb{Z}_2^n$ is linear iff $f(a+b) = f(a)+f(b)$), for our domains also the multiplicativity condition for certain scalars has to be verified.

The particular representation of $P(a, \tilde{r})$ seems to be necessary in order to follow these ideas. Instead of considering $A \circ L_A$ as a single linear function (similarly for B,C,D) it is important to separate the input-depending parts $L_A, \ldots, L_D$ from the "guess" $a \in \mathbb{R}^n$. This makes self-testing more complicated, but gives an easier realization of a consistency test (basically as in the classical proof).

4 Self-Testing and -Correcting over Real Domains

Our goal is in the end to guess with high probability linear functions A,B,C,D on a domain $\mathcal{X}_0$. Guessing is done by means of giving all function values of the corresponding mappings on an enlarged domain $\mathcal{X}_2$. Since the latter has exponential size our guess has exponential length. Next, we self-test the guess in order to figure out with high probability whether it really corresponds to linear functions on $\mathcal{X}_0$. Finally, we also have to guarantee with high probability that the four mappings we guessed all are generated by the same vector $a \in \mathbb{R}^n$ according to the definition of the four mappings given in the previous section.

4.1 The Appropriate Domains

We describe our construction for the linear function A, only. The same works for B, C, and D.

Let $C_0 := \{\lambda_1, \ldots, \lambda_K\} \subset \mathbb{R}$ be the multiset of all non-zero coefficients present in the matrix of the linear function L_A, $\lambda_1 := 1$. Since $m = O(n)$ and since each polynomial p_i depends on at most 3 variables we get as cardinality of C_0 a value $K = O(n)$. Consider the set

$$\mathcal{X}_0 := \left\{ \sum_{i=1}^{K} s_i \cdot \lambda_i \mid s_i \in \{0,1\} \text{ for } 1 \le i \le K \right\}^n \subset \mathbb{R}^n . \qquad (1)$$

It is $\mathbb{Z}_2^n \subset \mathcal{X}_0$ as well as $L_A(\mathbb{Z}_2^m) \subseteq \mathcal{X}_0$ (because we defined C_0 as a multiset). Moreover, all sums of at most K many terms of the form $\lambda \cdot z, z \in \mathbb{Z}_2^n, \lambda \in C_0$ belong to $\mathcal{X}_0$.

We denote the set of values of a particular component j of $\mathcal{X}_0$ by $\mathcal{X}_{0,j}$, that is $\mathcal{X}_0 = \{\mathcal{X}_{0,j}\}^n$.

The set $\mathcal{X}_0$ is the domain on which finally we want to check linearity of the functions we guess. However, due to the difficulties with the (new) probability distributions we get by shifting $\mathcal{X}_0$ with a fixed element (note that the sum of two elements from $\mathcal{X}_0$ does not necessarily belong to $\mathcal{X}_0$), we have to enlarge $\mathcal{X}_0$ significantly. This enlargement follows ideas similar to those in [7].

In a first step we enlarge $\mathcal{X}_0$ to $\mathcal{X}_1$ by defining $\mathcal{X}_1 := \mathcal{X}_0 \oplus \mathcal{X}_0$.

In a second step we enlarge $\mathcal{X}_1$ to a set $\mathcal{X}_2$ given as

$$\mathcal{X}_2 := \left\{ \sum_{i=1}^{K} s_i \cdot \lambda_i \mid s_i \in \{-n^3, -n^3+1, \ldots, n^3\} \text{ for } 1 \le i \le K \right\}^n \subset \mathbb{R}^n . \quad (2)$$

Once more, $\mathcal{X}_{2,j}$ denotes the set of values occuring as j-th component of a point in $\mathcal{X}_2$.

The goal of this construction is to guarantee that for an arbitrary and fixed element $x \in \mathcal{X}_1$ the set $x + \mathcal{X}_2$ still contains a major fraction of the set $\mathcal{X}_2$ itself.

Remark 1. Instead of just counting the different numerical values in $\mathcal{X}_0, \mathcal{X}_1, \mathcal{X}_2$ we consider the construction of elements through the defining formula, i.e. we consider the s_i-coefficients involved in the definition. If in $\mathcal{X}_0$ the same real component is generated by two different sums $\sum s_i \cdot \lambda_i$ and $\sum \tilde{s}_i \cdot \lambda_i$ we count it twice. This results in $|\mathcal{X}_0| = 2^{K \cdot n}$. Whenever we speak about the uniform distribution on $\mathcal{X}_0$ we mean that each formal sum is assigned the uniform probability among the set of all such formal sums (i.e. we take the uniform distribution over the coefficient vectors). Similarly for $\mathcal{X}_2$. For $\mathcal{X}_1$ we count $|\mathcal{X}_1| = |\mathcal{X}_0|^2$. That way of counting the elements makes calculations much easier because we are not forced to analyse algebraic dependencies among the numbers that are produced when applying the mappings L_A, L_B, L_C, L_D to elements from $\mathbb{Z}_2^m$.

Lemma 1. *Let $n \in \mathbb{N}, j \in \{1, \ldots, n\}$. The following cardinality bounds hold:*

a) $|\mathcal{X}_{2,j}| = (2n^3+1)^K$ *and* $|\mathcal{X}_2| = (2n^3+1)^{K \cdot n}$.

b) There is a constant $c_1 > 0$ *such that for each* $x \in \mathcal{X}_0$ *it is*

$$\frac{|x + \mathcal{X}_2 \cap \mathcal{X}_2|}{|\mathcal{X}_2|} \ge 1 - \frac{c_1}{n} .$$

c) There is a constant $c_2 > 0$ *such that for each* $x \in \mathcal{X}_1$ *it is*

$$\frac{|x + \mathcal{X}_2 \cap \mathcal{X}_2|}{|\mathcal{X}_2|} \ge 1 - \frac{c_2}{n} .$$

The following is an outline of the proof idea. Suppose we guess the values of a function $A : \mathbb{R}^n \mapsto \mathbb{R}$ on the set $\mathcal{X}_2 \bigoplus \mathcal{X}_2 \bigoplus \mathcal{X}_2$; following Remark 1 $\oplus$ indicates that we consider an element of this set as the sum of three elements in $\mathcal{X}_2$, not just as a numerical value. Recall that $\mathcal{X}_2$ has exponential size in n.
We want to check by a verifier whether A actually is a linear function at least on the subset $\mathcal{X}_0$ of $\mathcal{X}_2$. As we shall see below this involves the two conditions

$$(Add) \quad A(x+y) = A(x) + A(y) \quad \forall\, x, y \in \mathcal{X}_0 \tag{3}$$

$$(SM) \quad A(\lambda \cdot x) = \lambda \cdot A(x) \quad \forall\, x \in \mathbb{Z}_2^n \subset \mathcal{X}_0, \lambda \in C_0. \tag{4}$$

In a first step we build a verifier that accepts A with probability 1 if *(Add)* holds and rejects A with probability $\geq \frac{1}{2}$ if A is not close (a notion that has to be precised) to a function g_A^+ satisfying *(Add)* on $\mathcal{X}_0$. In a second step we independently construct another verifier that accepts A with probability 1 if (SM) holds and rejects with probability $\geq \frac{1}{2}$ if A is not close (to be precised) to a function g_A^* that satisfies (SM) on $C_0 \times \mathbb{Z}_2^n$. We then combine these two tests and perform them for B,C,D as well. In the third step we face the situation that all tables we guessed are close to linear functions on $\mathcal{X}_0$. Another verifier is constructed that rejects (A, B, C, D) with probability $\geq \frac{1}{2}$ if they are not generated from the same vector $a \in \mathbb{R}^n$ (in the sense of Section 3) or if that vector was not a zero of the given QPS instance.
Whereas the proof structure resembles the classical one, Step 1 above is much more involved due to the use of $\mathcal{X}_0$ instead of $\mathbb{Z}_2^n$ and Step 2 is not necessary over $\mathbb{Z}_2^n$ because there linearity can be defined by additivity, only. Note that it is enough to construct the required verifier for sufficiently large input sizes n.

4.2 Additivity und Multiplicativity

Given the table for the values of A and $0 < \delta_1 \leq 1$ consider the following test (compare Test 7.1 in [3]):

Test 1: For $i = 1$ to $k := \lceil \frac{2}{\delta_1} \rceil$ do

i) pick randomly (according to the uniform distribution defined in Remark 1) elements x, y from $\mathcal{X}_2$;
ii) if $A(x+y) \neq A(x) + A(y)$ reject.

If all test pairs satisfy additivity accept A.

Lemma 2. *(see [3])*

a) For a function A satisfying additivity Test 1 accepts with probability 1.
b) If $\Pr_{x,y \in \mathcal{X}_2} \{A(x+y) \neq A(x) + A(y)\} > \frac{\delta_1}{2}$ *(this implies in particular that A is not linear), then Test 1 rejects with probability at least* $\frac{1}{2}$. *This probability can be pushed arbitrarily close to* 1 *by increasing* k.

The test leaves as most important case the one where A is not linear but

$$\Pr_{x,y\in\mathcal{X}_2}\{A(x+y)\neq A(x)+A(y)\}\leq\frac{\delta_1}{2}. \tag{5}$$

This case involves a major part of the work. The verifier we are going to construct will work as follows: First, it is proven that if A satisfies condition (5) a unique function g_A^+ that satisfies the additivity condition on $\mathcal{X}_0$ can be defined from A.

From now on suppose that for an arbitrary fixed $\delta_1>0$ (to be chosen later) condition (5) is satisfied. The latter can be guaranteed to hold with high probability with respect to Lemma 2 if Test 1 was performed sufficiently (constantly!) many times without failure. We use (5) in order to define a function g_A^+ on $\mathcal{X}_2$: For a fixed $a\in\mathcal{X}_2$ consider the set of values we obtain when evaluating $A(a+x)-A(x)$ for all $x\in\mathcal{X}_2$. Define

$$g_A^+(a):=majority_{x\in\mathcal{X}_2}\{A(a+x)-A(x)\}, \tag{6}$$

i.e. $g_A^+(a)$ is the most frequent value that occurs among $\{A(a+x)-A(x),x\in\mathcal{X}_2\}$ (by breaking ties arbitrarily).

Proposition 1. *Under the above assumptions, for sufficiently large n the function g_A^+ satisfies the additivity condition on $\mathcal{X}_0$, i.e.*

$$g_A^+(a+b)=g_A^+(a)+g_A^+(b)\ \forall\ a,b\in\mathcal{X}_0.$$

Proof. Let $a\in\mathcal{X}_0$ be fixed. According to Lemma 1, b) for any $\epsilon_1>0$ and large enough n the set $\{a+x|x\in\mathcal{X}_2\}$ contains a fraction of at least $1-\frac{\epsilon_1}{2}$ many points of $\mathcal{X}_2$. Furthermore, according to condition (5)

$$\Pr_{x,y\in\mathcal{X}_2}\{A(x)=A(x+y)-A(y)\}\geq 1-\frac{\delta_1}{2} \tag{7}$$

Thus, we get

$$\Pr_{x,y\in\mathcal{X}_2}\{A(a+x)=A(a+x+y)-A(y)\}\geq 1-\frac{\delta_1}{2}-\frac{\epsilon_1}{2}. \tag{8}$$

Similarly

$$\Pr_{x,y\in\mathcal{X}_2}\{A(a+y)=A(a+x+y)-A(x)\}\geq 1-\frac{\delta_1}{2}-\frac{\epsilon_1}{2} \tag{9}$$

which results in

$$\Pr_{x,y\in\mathcal{X}_2}\{A(a+x)-A(x)=A(a+y)-A(y)\}\geq 1-(\delta_1+\epsilon_1). \tag{10}$$

It is easy to see that for small enough δ_1,ϵ_1 (f.e. if their sum is $<\frac{1}{2}$) the latter probability is a lower bound for the probability of obtaining the majority

result among $A(a+x)-A(x)$ with respect to the uniform distribution over $\mathcal{X}_2$, see [7]. This implies

$$\Pr_{x\in\mathcal{X}_2}\{g_A^+(a)=A(a+x)-A(x)\}\geq 1-(\delta_1+\epsilon_1). \tag{11}$$

Next, we consider $g_A^+(b)$. A similar argument as before (by replacing a by b and shifting the randomly chosen elements x and y in (8) with the fixed a and using Lemma 1) results in

$$\Pr_{x\in\mathcal{X}_2}\{g_A^+(b)=A(b+a+x)-A(a+x)\}\geq 1-(\delta_1+3\cdot\epsilon_1). \tag{12}$$

Finally, we use the same argument once more, this time for $a+b\in\mathcal{X}_1$; Lemma 1, c) implies that for an arbitrarily chosen $\epsilon_2>0$ and n large enough a fraction of at least $1-\frac{\epsilon_2}{2}$ points from $\mathcal{X}_2$ occurs in $a+b+\mathcal{X}_2$. Furthermore,

$$\Pr_{x\in\mathcal{X}_2}\{g_A^+(a+b)=A(b+a+x)-A(x)\}\geq 1-(\delta_1+\epsilon_2) \tag{13}$$

and altogether - by combining (11), (12), (13):

$$\Pr_{x\in\mathcal{X}_2}\{g_A^+(a+b)=g_A^+(a)+g_A^+(b)\}\geq 1-(3\cdot\delta_1+4\cdot\epsilon_1+\epsilon_2). \tag{14}$$

The latter is independent of x. For small enough $\delta_1,\epsilon_1,\epsilon_2$, for example $\delta_1:=\epsilon_1:=\epsilon_2:=\frac{1}{16}$ the right-hand side of (14) is strictly positive, which implies the probability on the left-hand side to equal 1. □

Though g_A^+ is additive on $\mathcal{X}_0$ it does not necessarily have to be linear. Therefore, we set up another verification procedure similar to the one in the previous sub-section, but this time tailoring for guaranteeing multiplicativity on a sufficiently large set. Enlarge the multiset C_0 by defining

$$C_1 := \left\{\prod_{i=1}^{K}\lambda_i^{t_i}\ ,\ t_i\in\{-n,\ldots,n\}\right\} \tag{15}$$

We "count" the number of elements in C_1 as the number of ways to choose $(t_1,\ldots,t_K)$.

Lemma 3. *a)* $|C_1|=(2n+1)^K$
b) For fixed $\lambda\in C_0$ *it is*

$$\frac{|\lambda\cdot C_1\ \cap\ C_1|}{|C_1|}\ \geq\ 1-\frac{1}{n}.$$

Suppose A has passed Test 1 and g_A^+ is defined as in the previous subsection. We design a second test in order to guarantee multiplicativity on a sufficiently large set as well.

Let $\delta_2>0$ be fixed.

Test 2: For $i = 1$ to $k := \lceil \frac{2}{\delta_2} \rceil$ do

i) pick random elements $\mu \in C_1, x \in \mathbb{Z}_2^n$;
ii) if $\dfrac{A(\mu \cdot x)}{\mu} \neq A(x)$ reject.

If all test tuples satisfy equality accept A.

Again it easily follows:

Lemma 4. *a) For a linear function A Test 2 accepts with probability 1.*
b) If $\Pr_{\mu \in C_1, x \in \mathbb{Z}_2^n} \left\{ \frac{A(\mu x)}{\mu} \neq A(x) \right\} > \frac{\delta_2}{2}$ *(implying in particular that A is not linear), then Test 2 rejects with probability at least* $\frac{1}{2}$. *This probability can be pushed arbitrarily close to* 1 *by increasing* k.

Once again, we have to analyze the case where A might not be linear but

$$\Pr_{\mu \in C_1, x \in \mathbb{Z}_2^n} \left\{ \frac{A(\mu x)}{\mu} \neq A(x) \right\} \leq \frac{\delta_2}{2} . \tag{16}$$

In case Test 2 accepts we now define a function g_A^* that satisfies multiplicativity on a large enough set. The construction of g_A^* is a bit different than that of g_A^+ in that we define g_A^* by certain majority results, but only if the majority value occurs with sufficiently high probability.

Definition of g_A^*: **Step 1:** Let $0 < \delta_3 < \frac{1}{2}$. For $x \in \mathbb{Z}_2^n$ fixed define the value $g_A^*(x)$ as

$$g_A^*(x) := majority_{\mu \in C_1} \frac{A(\mu \cdot x)}{\mu} \tag{17}$$

but that definition is performed only if the majority result occurs with probability at least $1 - \delta_3$, i.e. we define $g_A^*(x) := t \in \mathbb{R}$ only if

$$\Pr_{\mu \in C_1} \left\{ t = \frac{A(\mu \cdot x)}{\mu} \right\} \geq 1 - \delta_3. \tag{18}$$

Step 2: Let M denote the set of all $x \in \mathbb{Z}_2^n$ for which $g_A^*(x)$ was already defined in Step 1. Then for all $\lambda \in C_0, \lambda \neq 1$ we put

$$g_A^*(\lambda \cdot x) := majority_{\mu \in C_1} \frac{A(\mu \cdot \lambda \cdot x)}{\mu}. \tag{19}$$

Proposition 2. *Let* $0 < \delta_2 < \delta_3 < \frac{1}{8}$ *and* $0 < \epsilon_3 < \frac{1}{8}$. *Choose* n *large enough such that Lemma 3 implies* $\frac{|\lambda \cdot C_1 \cap C_1|}{|C_1|} \geq 1 - \epsilon_3$.

a) If A passes Test 2 with respect to δ_2 *without rejection, then there exists a basis* $M \subseteq \mathbb{Z}_2^n$ *of* $\mathbb{R}^n$ *such that for all* $x \in M$ *and for all* $\lambda \in C_0$ *the values* $g_A^*(\lambda \cdot x)$ *are defined through (17), (18), (19).*

b) For all $\lambda \in C_0, x \in M$ *it is*

$$g_A^*(\lambda \cdot x) = \lambda \cdot g_A^*(x). \tag{20}$$

Proof. a) It suffices to show that for a set M of at least $2^{n-1}+1$ many points $x \in \mathbb{Z}_2^n$ and for all $\lambda \in C_0$ the value $g_A^*(\lambda \cdot x)$ is defined. This clearly implies M to contain a basis of $\mathbb{R}^n$. According to Test 2 we have

$$\Pr_{\mu \in C_1, x \in \mathbb{Z}_2^n} \left\{ \frac{A(\mu \cdot x)}{\mu} = A(x) \right\} \geq 1 - \frac{\delta_2}{2}. \tag{21}$$

Any $x \in \mathbb{Z}_2^n$ for which $g_A^*(x)$ is not defined in Step 1 gives rise to at least $\delta_3 \cdot |C_1|$ many faults among the $\frac{A(\mu x)}{\mu}$. According to (21) there can be at most $\frac{\delta_2}{2\delta_3} \cdot 2^n$ such x. Given that $\frac{\delta_2}{\delta_3} < 1$ we conclude that the set M of points where g_A^* is defined at least contains $2^{n-1}+1$ many elements.
Now for $x \in M, \lambda \in C_0$ Lemma 3 implies that a fraction of at least $1-\epsilon_3$ points $\mu \in C_1$ produces again a result $\mu \cdot \lambda \in C_1$. Therefore, if we take all values $\left\{ \frac{A(\mu\lambda x)}{\mu\lambda}, \mu \in C_1 \right\}$, at least $(1-\delta_3-\epsilon_3) \cdot |C_1|$ choices for μ give the same result. Hence $g_A^*(\lambda x)$ is defined in Step 2. Note that the majority value among the $\frac{A(\mu\lambda x)}{\mu}$ equals $\lambda \cdot g_A^*(x)$.
b) For $x \in M, \lambda \in C_0$ the previous arguments in particular imply:

$$\Pr_{\mu \in C_1} \left\{ \lambda \cdot g_A^*(x) = \frac{A(\mu \cdot \lambda \cdot x)}{\mu} \right\} \geq 1 - \delta_3 - \epsilon_3 \tag{22}$$

as well as

$$\Pr_{\mu \in C_1} \left\{ g_A^*(\lambda \cdot x) = \frac{A(\mu \cdot \lambda \cdot x)}{\mu} \right\} \geq 1 - \delta_3 - \epsilon_3 \ . \tag{23}$$

It follows

$$\Pr_{\mu \in C_1} \{ g_A^*(\lambda \cdot x) = \lambda \cdot g_A^*(x) \} \geq 1 - 2(\delta_3 + \epsilon_3) \geq 1 - \frac{1}{2} > 0. \tag{24}$$

This probability is independent of μ and therefore the latter inequality implies that g_A^* satisfies multiplicativity on the domain $C_0 \times M$. □

Tests 1 and 2 now are combined in the sense that for all arguments $\mu \cdot x$ and x that are randomly picked in Test 2 we check whether we have evaluated A already on one of these arguments in Test 1. We refer to this combined test as **Test 1-2.** We actually have shown

Proposition 3. *Suppose Test 1-2 was performed without rejection for A. If A is a linear function on* $\mathcal{X}_0$ *(with respect to additivity) and* $C_0 \times M$ *(with respect to multiplicativity), then A equals* g_A^+ *and* g_A^* *and both are the same linear function on* $\mathbb{R}^n$. *We denote the latter by* g_A; *similarly for* g_B, g_C, *and* g_D, *respectively.*

4.3 Self-Correcting; Inconsistency; Satisfiability

Next, the verifier is extended so that it detects with high probability inconsistency. Due to the special way we used in Section 3 to represent the polynomial $P(a,\tilde{r})$ it is now possible to closely follow the classical proof in the Turing model. We include the description of how the verifier detects that g_A and g_B do not originate from the same a. The functions g_C, g_D then can be treated similarly.

We want to check whether $g_A : \mathbb{R}^n \mapsto \mathbb{R}$ and $g_B : \mathbb{R}^{n^2} \mapsto \mathbb{R}$ result from a single $a \in \mathbb{R}^n$. This is true iff for all $x, x' \in \mathbb{Z}_2^n$ the equality $g_A(x) \cdot g_A(x') = g_B(x \otimes x')$ holds, where $x \otimes x' := (x_1 \cdot x'_1, x_1 \cdot x'_2, \ldots, x_n \cdot x'_n) \in \mathbb{R}^{n^2}$, i.e. $(x \otimes x')_{ij} = y_{ij}$ in the sense of Section 3. The verifier for random $x, x' \in \mathbb{Z}_2^n$ computes with high probability the values $g_A(x), g_A(x')$ and $g_B(x \otimes x')$ and checks whether $g_A(x) \cdot g_A(x') = g_B(x \otimes x')$. The probability analysis ($x \otimes x'$ is no random element in $\mathbb{Z}_2^{n^2}$) requires to compute g_A and g_B by what is called self-correction of the functions A and B. First, we need

Lemma 5. *Let $a, b \in \mathbb{R}^n$. If $a \neq b$, then $\Pr_{x \in \mathbb{Z}_2^n}\{a^T \cdot x \neq b^T \cdot x\} \geq \frac{1}{2}$, where x is chosen uniformly from $\mathbb{Z}_2^n$.*
Similarly for matrices $M, N \in \mathbb{R}^{n \times n}, M \neq N$: $\Pr_{x \in \mathbb{Z}_2^n}\{M \cdot x \neq N \cdot x\} \geq \frac{1}{2}$.

Definition 4. *The random function SC-A is defined as follows: For $x \in \mathbb{Z}_2^n$ (note that $\mathbb{Z}_2^n \subset \mathcal{X}_0$) pick a random $y \in \mathcal{X}_2$ and return as result the value $A(x+y) - A(y)$. Similarly for SC-B.*

Let $0 < \delta_4 < 1$ be arbitrarily chosen and fixed.

Test 3 (Consistency): For $i = 1$ to $k := \lceil \frac{\log \delta_4}{\log \frac{7}{8}} \rceil$ do

i) pick $x, x' \in \mathbb{Z}_2^n$ randomly according to the uniform distribution on $\mathbb{Z}_2^n$.
ii) Pick $y, y', y'' \in \mathcal{X}_2$ according to the uniform distribution on $\mathcal{X}_2$.
iii) If $SC\text{-}A(x) \cdot SC\text{-}A(x') \neq SC\text{-}B(x \otimes x')$ reject. Here, we compute the results according to Definition 4 with respect to the randomly chosen y, y', y''.

If all test points satisfy equality accept.

Proposition 4. *Suppose that A, B pass Test 1-2 and that the corresponding linear function $g_A : \mathbb{R}^n \mapsto \mathbb{R}$ originates from an $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$ in the above way. If $a \otimes a \neq b$, then Test 3 rejects with a probability of at least $1 - \delta_4$.*

If Test 3 passes we perform a similar one for comparing g_A with g_C and g_D. It follows

Theorem 2. *For sufficiently small chosen probabilities in the Tests 1-2 and 3 it holds: If A,B,C,D pass Test 1-2 but they do not originate in a single vector $a \in \mathbb{R}^n$, then a verifier performing Test 3 for the three function comparisons mentioned above detects a fault with a probability arbitrarily close to* 1.

Finally, consider a problem instance $n, m, (p_1, \ldots, p_m)$ for the QPS problem. For a vector $a \in \mathbb{R}^n$ let $P(a, \tilde{r}), \tilde{r} \in \{0,1\}^m$ be the polynomial constructed from the p_i's in section 3. Let *A, B, C, D* be the corresponding linear functions which are given by tables of their function values on $\mathcal{X}_2$ and $C_1 \times \mathbb{Z}_2^n$.

Let $0 < \delta_5 < 1$ be arbitrarily chosen and fixed.

Test 4 (Satisfiability): For $i = 1$ to $k := \lceil \frac{\log \delta_5}{\log \frac{1}{2}} \rceil$ do

i) pick $\tilde{r} \in \mathbb{Z}_2^m$ randomly according to the uniform distribution on $\mathbb{Z}_2^m$.
ii) Evaluate $P(a, \tilde{r})$; if the result is different from 0 reject.

If $P(a, \tilde{r})$ vanishes for all test points $\tilde{r}$ accept.

Proposition 5. *If a is no solution for the QPS instance, then a verifier that performs Test 4 will figure it out with probability at least* $1 - \delta_5$. *The evaluation of* $P(a, \tilde{r})$ *can be done in polynomially many steps for each* $\tilde{r}$.

Proof. (of Theorem 1) The verifier for QPS performs Test 1-2, Test 3 and Test 4 for appropriately chosen values of the probabilities involved. If one of the tests gives a contradiction the verifier rejects, otherwise it accepts. According to Propositions 3, 4, 5 and Theorem 2 each fault in a verification proof is detected (given the corresponding previous tests passed) with probability arbitrarily close to 1 by inspecting constantly many function values. Finally, the number of random bits used can be estimated as $O(n^4 \cdot \log n)$. □

In this paper we have started the analysis of probabilistically checkable proofs in the model of Blum, Shub and Smale. We presented the first non-trivial PCP theorem for $\mathrm{NP}_{\mathbb{R}}$. The most challenging question to consider next is whether our PCP result can be improved in that we do not any more need proofs of exponential length. We did not work on that so far but believe that an extension of the present work should be possible to prove the

Conjecture: $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), O(1))$.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy: Proof verification and hardness of approximation problems. Proc. 33rd FOCS, 14–23, 1992.
2. S. Arora, S. Safra: Probabilistic checking proofs: A new characterization of NP. Journal of the ACM 45, 70–122, 1998.
3. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Protasi, M.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer (1999).
4. L. Blum, F. Cucker, M. Shub, S. Smale: Complexity and Real Computation. Springer, 1998.
5. P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, A. Widgerson: Self-Testing/Correcting for Polynomials and for Approximate Functions. Proc. of the 23rd STOCS, 32–42, 1991.
6. S. Ivanov, M. de Rougemont: Interactive Protocols on the reals. Computational Complexity 8, 330–345, 1999.
7. R. Rubinfeld, M. Sudan: Self-testing polynomial functions efficiently and over rational domains. Proc. 3rd SODA, 23–32, 1992.

A Time Lower Bound for Satisfiability

Dieter van Melkebeek*[1] and Ran Raz[2]

[1] University of Wisconsin-Madison
[2] Weizmann Institute of Science

Abstract. We show that a deterministic Turing machine with one d-dimensional work tape and random access to the input cannot solve satisfiability in time n^a for $a < \sqrt{(d+2)/(d+1)}$. For conondeterministic machines, we obtain a similar lower bound for any a such that $a^3 < 1 + a/(d+1)$. The same bounds apply to almost all natural NP-complete problems known.

1 Introduction

Proving time lower bounds for natural problems remains the most difficult challenge in computational complexity. We know exponential lower bounds on severely restricted models of computation (e.g., for parity on constant depth circuits) and polynomial lower bounds on somewhat restricted models (e.g., for palindromes on single tape Turing machines) but no nontrivial lower bounds on general random-access machines. In this paper, we exploit the recent time-space lower bounds for satisfiability on general random-access machines to establish new lower bounds of the second type, namely a time lower bound for satisfiability on Turing machines with one multidimensional work tape and random access to the input.

1.1 Lower Bounds for Satisfiability

Satisfiability constitutes the seminal NP-complete problem and is of major practical importance. While we expect the problem to take time $2^{\Omega(n)}$ in the worst case, the sad state of affairs is that we cannot even rule out the existence of a linear-time algorithm on a random-access Turing machine.

We do have nontrivial lower bounds on the running time of random-access Turing machines that solve satisfiability in sublinear space. We have seen considerable progress on such time-space lower bounds in recent years [3,6,4]. The state-of-the-art is a time lower bound of essentially n^ϕ for algorithms using subpolynomial space, where ϕ denotes the golden ratio, about 1.618. More precisely, the following holds:

Theorem 1 (Fortnow-Van Melkebeek [4]). *Let $\phi \doteq (\sqrt{5}+1)/2$ denote the golden ratio. For any constant $a < \phi$ there exists a positive constant b such that satisfiability cannot be solved on a deterministic random-access Turing machine in time n^a and space n^b.*

* Partially supported by NSF Career award CCR-0133693.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 971–982, 2004.

A nice feature of Theorem 1 is its model independence – the proof works for any reasonable model of computation. However, the theorem does not yield any lower bounds for algorithms that use linear space, e.g., algorithms that explicitly store an assignment to the given formula.

An almost quadratic time lower bound for satisfiability on single tape Turing machines immediately follows from the quadratic lower bound for palindromes in that model because of the standard efficient translation of any problem in NP to satisfiability. This result does not rely on the inherent difficulty of satisfiability, though. It rather exploits an artifact of the single tape Turing machine model – that the machine has to waste a lot of time in moving its tape head between both ends of the tape in order to retrieve information about the input. As soon as we include a work tape separate from the input tape, palindromes can be decided in linear time.

1.2 Our Results

We consider models of computation whose power lies between single tape Turing machines and random-access Turing machines, and establish time lower bounds of the form n^a where a is a constant larger than 1. Our proofs rely on the fact that satisfiability captures nondeterministic computation.

The first model we consider is that of a Turing machine with two tapes, namely an input tape and one work tape. The model is known as the single tape off-line Turing machine, and constitutes the strongest model with two-way access to the input on which superlinear time lower bounds for natural decision problems were established. Maass et al. [7] proved a lower bound of $\Omega(n \log n)$ for a problem in P, and Kannan [5] sketched a lower bound of $n^{1.104}$ for satisfiability. We improve Kannan's lower bound to n^a for any constant $a < \sqrt{3/2} \approx 1.224$. In fact, our result also holds if we allow random access to the input.

We generalize our lower bound to the case of Turing machines with a d-dimensional work tape.

Theorem 2 (Main Result). *For any positive integer d and any constant $a < \sqrt{(d+2)/(d+1)}$, satisfiability cannot be solved in time n^a on a deterministic Turing machine with a d-dimensional work tape and random access to the input.*

Dietzfelbinger and Hühne [2] proved a polynomial lower bound in this model but with the additional restriction that the input tape is one-way. Theorem 2 provides the first superlinear time lower bound for Turing machines with a planar or higher dimensional work tape and random-access to the input.

Our approach also applies to conondeterministic algorithms for satisfiability, or equivalently, to nondeterministic algorithms for tautologies.

Theorem 3. *For any positive integer d and any constant a such that $a^3 < 1 + a/(d+1)$, satisfiability cannot be solved in time n^a on a conondeterministic Turing machine with a d-dimensional work tape and random access to the input.*

The bound in Theorem 3 is somewhat weaker than the one in Theorem 2. The solution $a > 1$ of the equation $a^3 = 1 + a/(d+1)$ lies somewhere between $\sqrt[3]{(d+2)/(d+1)}$ and $\sqrt{(d+2)/(d+1)}$.

Time lower bounds for satisfiability immediately imply time lower bounds for problems to which satisfiability efficiently reduces. Almost all known natural NP-complete problems translate to satisfiability in quasilinear $(n \cdot \text{poly} \log n)$ time such that each bit of the translation can be computed in polylogarithmic time on a random-access Turing machine. As a corollary to Theorems 2 and 3, we can extend our lower bounds to all such problems.

Corollary 1. *The lower bounds of Theorems 2 and 3 apply to any problem to which satisfiability Karp-reduces in time $n^{1+o(1)}$ on a random-access Turing machine such that each bit of the reduction can be computed in time $n^{o(1)}$.*

1.3 Our Approach

Our starting point is the recent time-space lower bounds for satisfiability on random-access machines (see [10] for a survey). The high-level structure of these arguments is that of a proof by indirect diagonalization. We start from the assumption that satisfiability has a deterministic algorithm that runs in time t and space s. Since satisfiability captures nondeterministic (quasi-)linear time in a very strong sense, we can roughly view our assumption as the inclusion

$$\mathsf{NTIME}(n) \subseteq \mathsf{DTISP}(t, s), \tag{1}$$

where $\mathsf{DTISP}(t, s)$ denotes the class of problems that can be solved deterministically in time t and space s simultaneously. Then we use (1) to derive more and more unlikely inclusions of complexity classes, up to the point where we reach a contradiction with a diagonalization result.

A crucial step in the proof of Theorem 1 is an inclusion of the form

$$\mathsf{DTISP}(T, S) \subseteq \mathsf{NTIME}(f(T, S)), \tag{2}$$

where $f(T, S) \ll T$, which actually follows from a weaker hypothesis than (1), namely

$$\mathsf{NTIME}(n) \subseteq \mathsf{DTIME}(t). \tag{3}$$

Inclusion (2) describes a speedup of deterministic space bounded computations on nondeterministic machines and is proved by a combination of the following two arguments.

- We can speed up $\mathsf{DTISP}(T, S)$ computations on an alternating machine by breaking up the computation tableau into b blocks, guessing the configurations at the $b-1$ common boundaries of the blocks, and universally verifying the computation on each of the blocks of size T/b. This yields the inclusion

$$\mathsf{DTISP}(T, S) \subseteq \Sigma_2\mathsf{TIME}(b \cdot S + T/b).$$

Applying this idea k times recursively with block numbers $b_1, b_2, \ldots, b_k$, respectively, and exploiting the closure under complementation of deterministic classes to save about half of the alternations [4], we get

$$\mathsf{DTISP}(T,S) \subseteq \Sigma_{k+1}\mathsf{TIME}((\sum_j b_j) \cdot S + T/(\prod_j b_j)). \tag{4}$$

– We can eliminate alternations using the hypothesis (3). If $t(n)$ is of the form n^a for some constant a, (3) allows us to eliminate one alternation from an alternating computation at the cost of raising the running time to the power a. Eliminating all k alternations of the right-hand side of (4) from back to front yields a nondeterministic simulation running in time $f(T,S)$, where the actual form of $f(T,S)$ depends on a and the choice of $b_1, b_2, \ldots, b_k$.

The proof of Theorem 1 then proceeds as follows. For any smooth bound $\tau(n) \geq n$, the hypothesis (1) implies an inclusion of the form $\mathsf{NTIME}(\tau) \subseteq \mathsf{DTISP}(T,S)$. Combining with (2) leads to the conclusion $\mathsf{NTIME}(\tau) \subseteq \mathsf{NTIME}(f(T,S))$, which contradicts the nondeterministic time hierarchy theorem as long as $f(T,S) = o(\tau)$. The rest of the proof of Theorem 1 involves selecting optimal values for the number of alternations k and the block numbers $b_1, b_2, \ldots, b_k$ so as to minimize the function $f(T,S)$.

Now, suppose we try a similar strategy to obtain a *time* lower bound instead of a time-space lower bound. Thus, our aim is to derive a contradiction from the hypothesis $\mathsf{NTIME}(n) \subseteq \mathsf{DTIME}(t)$ where t is of the form $t(n) = n^a$ for as large a constant a as possible. Note that we can still exploit the speedup of space bounded computations by nondeterminism given by (2) since that step only used the hypothesis (3). The problem is to obtain a deterministic simulation of $\mathsf{NTIME}(\tau)$ that runs in small space. Such a simulation immediately follows from the stronger hypothesis (1) but we do not know how to derive it from the weaker hypothesis (3) when the underlying model of computation allows random memory access. In case of sequential memory access, however, we can break up the computation into pieces that each run in small space, and then apply (2) to each of these pieces.

Consider a deterministic computation that takes t steps on a Turing machine with a single work tape and random access to the input. We can simulate such a computation on an alternating random-access machine as follows: Break up the tape into b_0 blocks of size t/b_0 each. Guess the crossing sequences at all the block boundaries. By choosing an appropriate offset for the blocks, we can argue that the total number of crossings we need to guess is no more than b_0. Then switch to a universal mode and verify the computation on each of the b_0 blocks given the crossing sequences for that block. The verification for a given block can be performed in time $T = t$ and space $S = t/b_0$. This gives us the time-space bounded computation that is critical for the argument of Theorem 1. We can speed up (the complement of) that computation as in (4) and obtain a simulation that essentially lives in

$$\Sigma_{k+2}\mathsf{TIME}(b_0 + (\sum_{j\geq 1} b_j) \cdot S + T/(\prod_{j\geq 1} b_j)). \tag{5}$$

Now, suppose there exists a Turing machine with a single work tape and random access to the input that solves satisfiability in time t. Since random-access machines can efficiently simulate sequential machines, we have that, roughly, $\mathsf{NTIME}(n) \subseteq \mathsf{DTIME}(t)$, so we can eliminate alternations at the cost of a small increase in running time as before. Using a similar argument as in the proof of Theorem 1, we obtain a contradiction to the nondeterministic time hierarchy theorem for small t. It turns out that $k = 1$ leads to the strongest results for this approach – we can rule out running times $t(n)$ up to n^a for $a < \sqrt[3]{3/2}$.

We can do better by exploiting the following slack in our argument. We modeled the verification of any given block as a computation that takes time $T = t$ and uses space $S = t/b_0$. We cannot improve the upper bound $T = t$ for all blocks since it is possible for the computation to spend all its time in one particular block. On average, though, the time the computation spends on a block will be much less. We can benefit as follows from the fact that the total time spent on all blocks together is at most t.

Let t_i denote the time spent on block i. At the second existential level of (5), for a given block i, we guess a configuration after each t/b_1 steps the computation spends on block i. Thus, we really only need to guess $b_1 t_i/t$ configurations for block i at that level. The total number of configurations we guess at the second existential level is therefore bounded by $\sum_i b_1 t_i/t = b_1$. We can as well guess all these b_1 configurations at the first existential level. This saves us one alternation, leading to a simulation that lives in

$$\Sigma_{k+1}\mathsf{TIME}(b_0 + (\sum_{j \geq 1} b_j) \cdot S + T/(\prod_{j \geq 1} b_j)).$$

We note that an equivalent simulation can be obtained by viewing the process after guessing the offset and crossing sequences as a single $\mathsf{DTISP}(t, t/b_0)$ computation and applying (4) to the latter [8]. Using this improvement, we manage to rule out running times $t(n)$ up to n^a for $a < \sqrt{3/2}$.

Our arguments carry over to Turing machines with a d-dimensional work tape and random access to the input, as well as to conondeterministic machines.

1.4 Organization

In Section 2, we describe the various machine models we consider in this paper, and provide the required technical details of the known time-space lower bounds for satisfiability. Section 3 contains the derivation of our main result for Turing machines with a one-dimensional work tape and random access to the input. In Section 4, we extend that result to Turing machines with one d-dimensional work tape for arbitrary positive integers d, to conondeterministic Turing machines, and to NP-complete problems other than satisfiability.

2 Preliminaries

2.1 Machine Models

We use two different machine models – one with sequential memory access and one with random memory access. Both have random read access to the input.

Our main result holds for a sequential memory access model with one d-dimensional work tape for some positive integer d. The work tape has one tape head. In each computation step, the memory cell under the tape head can be accessed (read and/or written) and the tape head can be moved to a neighboring memory cell.

Our proofs also make use of machines with random memory access. We model random access using an auxiliary index tape. An index tape acts as a one-dimensional one-way write-only tape. In any given computation step, the machine can decide to access the cell indexed by the contents of the auxiliary index tape, after which the auxiliary index tape is automatically reset.

The random memory access model can simulate the sequential memory access model with a logarithmic overhead in time.

All notation for complexity classes, e.g., $\mathsf{NTIME}(t)$, refers to the random memory access model. We omit explicit constructibility conditions and other smoothness requirements on time and space bounds. Eventually, we only need to consider polynomial bounds, which meet all conditions needed.

2.2 Time-Space Lower Bounds for Satisfiability

We use a number of ingredients from the known time-space lower bounds for satisfiability. First, a reduction capturing the very close relationship between satisfiability and nondeterministic computation.

Lemma 1 (Cook [1]). *There exists a constant c such that for every language $L \in \mathsf{NTIME}(\tau)$ where $\tau(n) \geq n$, there exists a reduction from L to satisfiability that maps an input x of length n to a formula ϕ_x of length $N \leq \tau(n) \cdot (\log \tau(n))^c$. Moreover, given x and an index i, the ith bit of ϕ_x can be computed in time $(\log \tau(n))^c$.*

Second, we exploit the following crucial ingredient, which quantifies a speedup of deterministic time-space bounded computations on (co)nondeterministic machines that follows if we can simulate nondeterminism very efficiently on deterministic machines.

Lemma 2 (Fortnow-van Melkebeek [4]). *Suppose that*

$$\mathsf{NTIME}(n) \subseteq \mathsf{DTIME}(n^a)$$

for some constant $a \geq 1$. Then for any integer $k \geq 0$ and functions $T(n)$ and $S(n)$,

$$\mathsf{DTISP}(T, S) \subseteq \mathsf{coNTIME}((T \cdot S^k)^{c_k} + (n + S)^{a^k}), \tag{6}$$

where $c_0 = 1$ and $c_{k+1} = ac_k/(1 + c_k)$.

Note that the sequence c_k in Lemma 2 converges to $a-1$.

Finally, we also use the nondeterministic time hierarchy theorem.

Lemma 3 (Seiferas-Fischer-Meyer [9]). *Let $\tau_1(n)$ and $\tau_2(n)$ be time bounds. If $\tau_1(n+1) \in o(\tau_2(n))$ then*

$$\mathsf{NTIME}(\tau_2) \not\subseteq \mathsf{NTIME}(\tau_1).$$

In case $\tau_1(n) = n^{e_1}$ and $\tau_2(n) = n^{e_2}$ where e_1 and e_2 are positive constants, Lemma 3 implies that nondeterministic machines can do strictly more in time τ_2 than in time τ_1 if $e_2 > e_1$.

3 Result for One-Dimensional Tapes

In this section, we derive our time lower bound for satisfiability on deterministic machines with a one-dimensional work tape and random access to the input. We refer to Section 1.3 of the introduction for the intuition behind the derivation.

The proof goes by contradiction. We start from the hypothesis that satisfiability can be solved by a machine M with one work tape and random access to the input in time n^a for some constant $a \geq 1$. We then argue that for $a < \sqrt{3/2}$, this hypothesis leads to a contradiction with the nondeterministic time hierarchy theorem.

Let L be a language in $\mathsf{NTIME}(\tau)$ for some smooth function $\tau(n) \geq n$ which we will specify later. Let x be an input of length n and ϕ_x the Boolean formula of length $N \leq \tau(n) \cdot (\log \tau(n))^c$ that captures the membership of x to L, as given by Lemma 1. We decide the membership of x to L by simulating M on input ϕ_x on a random-access machine. Since each bit of ϕ_x can be computed on the fly in time poly $\log \tau$, the running time of the simulation is at most a factor poly $\log \tau$ times the running time of simulating M on ϕ_x when ϕ_x is given as input.

Consider the computation of M on input ϕ_x. Since M runs in time at most $t \doteq N^a$, M cannot access any memory cells outside the initial segment of length t. We break up this initial segment into b_0+1 consecutive blocks of roughly equal size S, and number the blocks 0 through b_0. More precisely, all blocks except possibly blocks 0 and b_0 contain exactly S cells, and blocks 0 and b_0 contain no more than S cells. See Figure 1, where f denotes the number of cells in block 0.

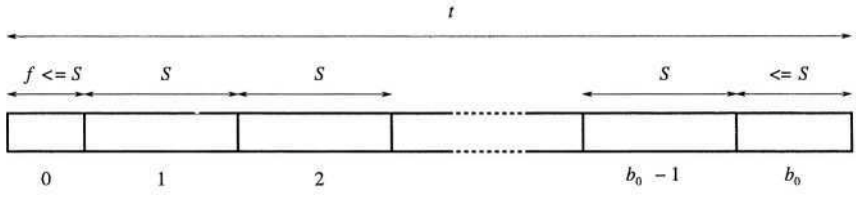


Fig. 1. Breaking up the work tape

Note that f and S fully determine the blocks. The value of b_0 is essentially equal to t/S; more precisely, $t/S - 1 \leq b_0 < t/S + 1$. The parameter S will be set later. We now determine a value for f.

For a given partition into blocks, the computation of M induces a crossing sequence at the boundary of any two consecutive blocks. The crossing sequence at a given boundary consists of a collection of records, one for each time the tape head crosses that boundary. The record corresponding to a particular crossing contains the time step of the crossing, its location on the tape, and the internal state of M as well as the configuration of the index tape for the input at the time of the crossing. Note that each crossing record involves $O(\log t)$ bits of information.

Let X_f denote the collection of all crossings over all the block boundaries for a given value of f (and S). By choosing the offset f appropriately, we can ensure that X_f contains no more than $b_0 + 1$ crossings. This is because the sets X_f, $1 \leq f \leq S$, form a partition of the collection of all crossings (over all boundaries between consecutive memory cells) during the computation. Since there can be at most one crossing per time step, the total number of crossings is no more than t. By averaging, there exists an offset f, $1 \leq f \leq S$, such that X_f contains no more than $t/S \leq b_0 + 1$ crossings.

Once we know f and X_f, we can break up the computation of M into $b_0 + 1$ independent parts, namely one for each block. We can check the correctness of X_f by verifying, for each block i, the consistency of the left end and the right end crossings. Since block i is of size at most S, the verification process P_i for block i can be executed deterministically in space S by simulating the parts of the computation of M that take place on block i.

Let us denote by t_i the time process P_i takes. Note that t_i equals the time M spends on block i. Thus, the sum of the t_i's is bounded by t. We can further break up the process P_i into time intervals of length T each (the last interval can be shorter), where T is another parameter we will determine later. Let $b'_i \doteq \lceil t_i/T \rceil$ denote the number of intervals for process P_i, $C_{i,0}$ the initial configuration of block i, and $C_{i,j}$, $1 \leq j \leq b'_i$, the configuration of block i at the end of the jth time interval of process P_i. A configuration contains a block identifier, the time step, the contents of the block, the internal state of M, and the configuration of the index tape for the input; it can be described using $S + O(\log t)$ bits. Note that the total number of intermediate configurations $C_{i,j}$ (i.e., over all $1 \leq i \leq b_0$ and $0 < j < b'_i$) is bounded by $\sum_i (b'_i - 1) \leq \sum_i t_i/T \leq t/T \doteq b_1$.

We can check the correctness of X_f and the configurations $C_{i,j}$ by verifying, for each block i and interval j, the consistency of the relevant crossings in X_f, the start configuration $C_{i,j-1}$ and the end configuration $C_{i,j}$. We denote the latter predicate by $\mathrm{Cons}_M(x, f, X_f, C_{i,j-1}, C_{i,j})$. It involves a simulation of M that can be executed in deterministic time $T \cdot \log^{O(1)} t$ and space $S + O(\log t)$.

By the above, we can decide membership of x to L by evaluating the following predicate:

$$\left.\begin{array}{l}(\exists \text{ offset } f)(\exists \text{ set of crossings } X)(\exists\, b'_i)(\exists \text{ internal configurations } C_{i,j}) \\ (\forall 0 \leq i \leq b_0)(\forall 1 \leq j < b'_i)\ \mathrm{Cons}_M(x, f, X, C_{i,j-1}, C_{i,j}), \qquad \}\,(\alpha)\end{array}\right\}(\beta)$$

where the existential quantifier for the configurations $C_{i,j}$ ranges over the internal configurations only. Here, we assume without loss of generality that M resets the work and index tape to their initial configuration before accepting, and has a unique accepting state. This implies that the start configurations $C_{i,0}$ and end configurations $C_{i,b_i'}$ for $1 \leq i \leq b_0$ are trivial.

We now analyze how efficiently we can evaluate (β) on a nondeterministic random access machine based on our hypothesis. In order to simplify the expressions, we will neglect multiplicative factors of the form poly $\log t$.

Since $\mathrm{Cons}_M(x, f, X, C_{i,j-1}, C_{i,j})$ involves a $\mathsf{DTISP}(T, S)$ computation on an input of length $O(n + b_0 + S)$, Lemma 2 allows us to transform (α) into a conondeterministic computation running in time

$$O((T \cdot S^k)^{c_k} + (n + b_0 + S)^{a^k})$$

for any integer $k \geq 0$.

Lemma 1 combined with our hypothesis implies that conondeterministic computations running in time $u(n) \geq n$ can be simulated deterministically in time $O(u^a \cdot \mathrm{poly} \log u)$. Applying that transformation to the computation (α), which has an input of size $O(n + b_0 + b_1 \cdot S)$, we can turn (β) into a nondeterministic computation taking time

$$O\left(\left((T \cdot S^k)^{c_k} + (n + b_0 + S)^{a^k} + b_1 \cdot S\right)^a\right). \tag{7}$$

We obtain a contradiction with the nondeterministic time hierarchy theorem provided (7) is $o(\tau(n-1)/\mathrm{poly} \log \tau(n))$. Our goal now is to select the parameters in such a way that we obtain that contradiction for values of a as large as possible.

Recall that $T = t/b_1$, $S = t/b_0$, and $t = \tau^a$. Setting $b_0 = t^{\beta_0}$ and $b_1 = t^{\beta_1}$ for constants $\beta_0, \beta_1 \in (0, 1)$ and letting $\tau(n) = n^e$ for a sufficiently large constant e, we obtain the contradiction we seek as long as

$$a^2 \cdot \max(((1 - \beta_1) + k(1 - \beta_0))c_k, \beta_0 a^k, (1 - \beta_0)a^k, \beta_1 + 1 - \beta_0) < 1. \tag{8}$$

For $k = 0$, the requirement simplifies to

$$a^2 \cdot \max(1 - \beta_1, \beta_0, \beta_1 + 1 - \beta_0) < 1, \tag{9}$$

for which the optimal choice of $\beta_0 = 2/3$ and $\beta_1 = 1/3$ leads to the bound $a < \sqrt{3/2}$.

Further calculations show that positive values of k do not lead to better bounds on a. By dropping all but the second and third components of the maximum expression in (8), the optimal setting of $\beta_0 = 1/2$ leads to the necessary condition that $a^{k+2} < 2$, which is stricter than $a < \sqrt{3/2}$ for $k > 1$. For $k = 1$, dropping the second term of the maximum expression in (8) and setting β_0 and β_1 optimally to $\beta_0 = \beta_1 = 1 - 1/a$ results in the necessary condition $a^2 < 1$. Thus, the result claimed in the statement of Theorem 2 for $d = 1$ is the best we can get using our approach.

4 Extensions

In this section, we extend the result from the previous section for machines with a one-dimensional work tape to machines with a d-dimensional work tape for arbitrary positive integers d. We also derive a similar lower bound for conon-deterministic machines, and argue that all our bounds apply to NP-complete problems other than satisfiability.

4.1 Multi-dimensional Tapes

We follow the proof outline of Section 3. We assume that satisfiability can be solved on a machine M with one d-dimensional tape and random access to the input in time n^a for some constant $a \geq 1$, and argue that this assumption leads to a contradiction with the nondeterministic time hierarchy theorem for $a < \sqrt{(d+2)/(d+1)}$.

We now break up each tape dimension into b_0 parts. The number of blocks becomes b_0^d; the number of possible offsets f as well as the space occupied by a single block becomes $S = (t/b_0)^d$.

There still exists a choice for the offset f such that the set of crossings X_f is of size at most b_0. The averaging argument can be modified as follows. Any given crossing that occurs during the computation of M can appear in up to $(t/b_0)^{d-1}$ of the sets X_f. This is because the crossing fixes the component of f in the dimension of the crossing but leaves the remaining $d-1$ components of f free. Thus, we get the inequality $\sum_f |X_f| \leq t \cdot (t/b_0)^{d-1}$. Since there are $(t/b_0)^d$ possible offsets, this implies that there exists at least one offset f for which $|X_f| \leq b_0$.

With these modified parameters, (β) still holds, as well as the bound (7). Using the same settings as before, condition (8) generalizes to

$$a^2 \cdot \max(((1-\beta_1) + kd(1-\beta_0))c_k, \beta_0 a^k, d(1-\beta_0)a^k, \beta_1 + d(1-\beta_0)) < 1. \quad (10)$$

For $k = 0$, the requirement simplifies to

$$a^2 \cdot \max(1-\beta_1, \beta_0, \beta_1 + d(1-\beta_0)) < 1,$$

for which the optimal choice of $\beta_0 = (d+1)/(d+2)$ and $\beta_1 = 1/(d+2)$ leads to the bound $a < \sqrt{(d+2)/(d+1)}$.

Again, $k = 0$ turns out to give the best results. This can be seen as follows. Dropping all but the second and third terms in the maximum expression in (10) and setting β_0 optimally to $\beta_0 = d/(d+1)$ leads to the necessary condition that $a^{k+2}d/(d+1) < 1$, which is more stringent than $a < \sqrt{(d+2)/(d+1)}$ for $k > 0$ and $d \geq 2$.

Note that a sufficient strengthening of Theorem 2 for general dimension d would imply a time lower bound for satisfiability on multi-tape Turing machines. This follows from the well-known simulation of a k-tape Turing machine running in time t on a Turing machine with one d-dimensional work tape in time $O(t\sqrt[d]{t})$ [11].

4.2 Conondeterministic Machines

The argument used in the proof of Theorem 2 can be modified for conondeterministic instead of deterministic machines.

There are two key modifications. The first one is the use of the following diagonalization result instead of the nondeterministic time hierarchy theorem given in Lemma 3.

Lemma 4. *Let $\tau(n)$ be a time bound.*

$$\mathsf{coNTIME}(\tau) \not\subseteq \mathsf{NTIME}(o(\tau)).$$

Thus, assuming there exists a conondeterministic Turing machine with one d-dimensional work tape and random access to the input that solves satisfiability in time $t(n) = n^a$, we aim for a contradiction to Lemma 4 by showing that an arbitrary language $L \in \mathsf{coNTIME}(\tau)$ can be simulated on a nondeterministic machine in time $o(\tau)$.

We can decide L by evaluating the predicate (β), in which the matrix Cons_M now involves a $\mathsf{NTISP}(T, S)$ computation. The second key modification is that we apply the following lemma instead of Lemma 2 to speed up the computation of Cons_M.

Lemma 5 (Fortnow-van Melkebeek [4]). *Suppose that*

$$\mathsf{NTIME}(n) \subseteq \mathsf{coNTIME}(n^a)$$

for some constant $a \geq 1$. Then for any integer $k \geq 0$ and functions $T(n)$ and $S(n)$,

$$\mathsf{NTISP}(T, S) \subseteq \mathsf{NTIME}((T \cdot S^k)^{g_k} + (n+S)^{a^{2k}}),$$

where $g_0 = 1$ and $g_{k+1} = a^2 g_k/(1 + a g_k)$.

Since the fast simulation of Cons_M entails a nondeterministic computation, we need to eliminate the additional alternation it induces. After eliminating this and the original alternation, we obtain a nondeterministic algorithm for L that runs in time

$$\left(((T \cdot S^k)^{g_k} + (n + b_0 + S)^{a^{2k}})^a + b_1 \cdot S\right)^a \tag{11}$$

times a term of the form $\mathrm{poly}\log t$. We obtain a contradiction with Lemma 4 provided (11) is $o(\tau/\mathrm{poly}\log\tau)$. Setting the parameters as before, we get this contradiction as long as there exists an integer $k \geq 0$ and constants $\beta_0, \beta_1 \in (0,1)$ such that

$$a^2 \cdot \max(a((1-\beta_1) + kd(1-\beta_0))g_k, \beta_0 a^{2k+1}, d(1-\beta_0)a^{2k+1}, \beta_1 + d(1-\beta_0)) < 1.$$

For $k = 0$, the optimal setting of the parameters is $\beta_0 = (d+1)/(a+d+1)$ and $\beta_1 = a/(a+d+1)$, leading to the condition that $a^3 < 1 + a/(d+1)$.

Once again, we do not obtain stronger results for higher values of k, as can be seen as follows. For $k = 1$, dropping the second term in the above maximum

expression and setting β_0 and β_1 optimally to $\beta_0 = 1 - 1/((a^3 + a - 1)d)$ and $\beta_1 = 1 - a/(a^3 + a - 1)$ results in the necessary condition $a^5 < a^3 + a - 1$, which has no solutions $a \geq 1$. For $k \geq 2$, dropping all but the second and third terms in the above maximum expression and setting β_0 optimally to $\beta_0 = d/(d+1)$ leads to the necessary condition $a^{2k+3} < 1 + 1/d$, which is more stringent than $a^3 < 1 + a/(d+1)$ for $d \geq 2$.

4.3 NP-Complete Problems Other Than Satisfiability

We now sketch a proof of Corollary 1.

Let A be a language to which satisfiability reduces under a Karp reduction R that runs in time $n^{1+o(1)}$ on a random access machine such that each bit of the reduction can be computed in time $n^{o(1)}$. The proof of Theorem 2 carries through if we let ϕ_x denote the combination of the reduction from a language L in $\mathsf{NTIME}(\tau)$ to satisfiability as given by Lemma 1, and the reduction R from satisfiability to A. The analysis of the running time remains the same up to multiplicative factors of the form $\tau^{o(1)}$. Since these factors do not affect the condition (8), the conclusion of Theorem 2 also holds if we replace satisfiability by A. The proof of Theorem 3 carries over in a similar way.

Acknowledgments. We would like to thank Ravi Kannan, Rahul Santhanam, and the anonymous referees for helpful discussions, pointers to the literature, and other suggestions.

References

1. S. Cook. Short propositional formulas represent nondeterministic computations. *IPL,* 26:269–270, 1988.
2. M. Dietzfelbinger and M. Hühne. Matching upper and lower bounds for simulations of several tapes on one multidimensional tape. *CC,* 8:371–392, 1999.
3. L. Fortnow. Time-space tradeoffs for satisfiability. *JCSS,* 60:337–353, 2000.
4. L. Fortnow and D. van Melkebeek. Time-space tradeoffs for nondeterministic computation. In *CCC,* pages 2–13. IEEE, 2000.
5. R. Kannan. Alternation and the power of nondeterminism. In *STOC,* pages 344–346. ACM, 1983.
6. R. Lipton and A. Viglas. On the complexity of SAT. In *FOCS,* pages 459–464. IEEE, 1999.
7. W. Maass, G. Schnitger, E. Szemeredi, and G. Turan. Two tapes versus one for off-line Turing machines. *CC,* 3:392–401, 1993.
8. V. Nepomnjaščiĭ. Rudimentary predicates and Turing calculations. *Doklady,* 11:1462–1465, 1970.
9. J. Seiferas, M. Fischer, and A. Meyer. Separating nondeterministic time complexity classes. *JACM,* 25:146–167, 1978.
10. D. van Melkebeek. Time-space lower bounds for satisfiability. *BEATCS,* 73:57–77, 2001.
11. K. Wagner and G. Wechsung. *Computational Complexity.* Reidel Publishing, 1986.

Some Results on Effective Randomness

Wolfgang Merkle[1], Nenad Mihailović[1], and Theodore A. Slaman[2]

[1] Ruprecht-Karls-Universität Heidelberg, Institut für Informatik
{merkle|mihailovic}@math. uni-heidelberg.de
[2] University of California, Department of Mathematics,
slaman@math.berkeley.edu

Abstract. We investigate the characterizations of effective randomness in terms of Martin-Löf covers and martingales. First, we address a question of Ambos-Spies and Kučera [1], who asked for a characterization of computable randomness in terms of tests. We argue that computable randomness can be characterized in term of Martin-Löf tests and effective probability distributions on Cantor space.
Second, we show that the class of Martin-Löf random sets coincides with the class of sets of reals that are random with respect to computable martingale processes. This improves on results of Hitchcock and Lutz [8], who showed that the latter class is contained in the class of Martin-Löf random sets and is a strict superset of the class of rec-random sets.
Third, we analyze the sequence of measures of the components of a universal Martin-Löf test. Kučera and Slaman [12] showed that any component of a universal Martin-Löf test defines a class of Martin-Löf random measure. Further, since the sets in a Martin-Löf test are uniformly computably enumerable, so is the corresponding sequence of measures. We prove an exact converse and hence a characterization. For any uniformly computably enumerable sequence $r_1, r_2, \ldots$ of reals such that each r_i is Martin-Löf random and less than 2^{-i} there is a universal Martin-Löf test $U_1, U_2, \ldots$ such that $U_i\{0,1\}^\infty$ has measure r_i.

1 Introduction

We investigate the characterizations of effective randomness concepts in terms of Martin-Löf tests and martingales. After reviewing concepts of effective randomness and measure in Section 2, we consider in Section 3 a question of Ambos-Spies and Kučera [1], who ask for a characterization of computable randomness in terms of tests. We give such a characterization in terms of Martin-Löf tests such that there is an effective probability distribution μ where for any index k and any word w, the Lebesgue measure of the kth component of the test within the cylinder generated by w is bounded from above by $\mu(w)/k$. An essentially identical characterization has been obtained independently and earlier by Downey, Griffiths, and Laforte [7].

In Section 4, we show that the class of Martin-Löf random sequences coincides with the class of sequences that are random with respect to computable martingale processes. This improves on a result of Hitchcock and Lutz [8], who showed

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 983–995, 2004.

that the latter class is contained in the class of Martin-Löf random sequences and is a strict superclass of the class of rec-random sequences.

Finally, in Section 5, we demonstrate that in the characterization of Martin-Löf randomness by universal Martin-Löf tests the measure of the classes defined by the components of the universal test can be chosen according to any given sequence $r_1, r_2, \ldots$ of uniformly c.e. and Martin-Löf random reals where $r_i < 2^{-i}$, i.e., for any such sequence there is a universal Martin-Löf test such that the measure of the class defined by the ith component of the test is just r_i. This assertion complements the result of Kučera and Slaman [12] that the components of any universal Martin-Löf test define classes of Martin-Löf random measure (where, trivially, these measures form a uniformly c.e. sequence of reals). In summary, reals $r_1, r_2, \ldots$ are random, are uniformly c.e., and satisfy $r_i < 2^{-i}$ if and only if there is a universal Martin-Löf test $U_1, U_2, \ldots$ such that the measure of $U_i\{0,1\}^\infty$ is r_i. The latter has a similar flavor as the characterization of the Martin-Löf random c.e. reals as the reals that are the halting probability of a universal prefix machine; see Calude [5] for further discussion and references.

1.1 Notation

The notation used in the following is mostly standard, for unexplained notation refer to the surveys and textbooks cited in the bibliography [2,3,15].

We consider words over the binary alphabet {0, 1}, the empty word is denoted by λ. If not explicitly stated differently, SETS are sets of words, SEQUENCES are infinite binary sequences and the term CLASS refers to a set of sequences. For any sequence A, let A be equal to $A(0)A(1)\ldots$, i.e., $A(i)$ denotes the $(i+1)$th bit of A. The class of all sequences is referred to as CANTOR SPACE and is denoted by $\{0,1\}^\infty$. The class of all sequences that have a word x as common prefix is called the CYLINDER GENERATED BY x and is denoted by $x\{0,1\}^\infty$. For a set of words W, let $W\{0,1\}^\infty$ be the union of all the cylinders $x\{0,1\}^\infty$ where the word x is in W.

Recall the definition of the LEBESGUE MEASURE (or UNIFORM MEASURE) λ on Cantor space, which describes the distribution obtained by choosing the individual bits of a sequence according to independent tosses of a fair coin.

2 Random Sequences

In this section, we review effectively random sequences and related concepts that are used in the following. For more comprehensive accounts of effectively random sequences and effective measure theory, we refer to the surveys cited in the bibliography [1,2,15].

Imagine a player who successively places bets on the individual bits of the characteristic sequence of an unknown sequence A. The betting proceeds in rounds $i = 1, 2, \ldots$. During round i, the player receives as input the length $i-1$ prefix of A and then, first, decides whether to bet on the ith bit being 0 or 1 and, second, determines the stake that shall be bet. The stake might be any

fraction between 0 and 1 of the capital accumulated so far, i.e., in particular, the player is not allowed to incur debts. Formally, a player can be identified with a BETTING STRATEGY

$$b : \{0,1\}^* \to [-1,1]$$

where on input w the absolute value of $b(w)$ is the fraction of the current capital that shall be at stake and the bet is placed on the next bit being 0 or 1 depending on whether $b(w)$ is negative or nonnegative.

The player starts with strictly positive, finite capital $d_b(\lambda)$. At the end of each round, in case the current guess has been correct, the capital is increased by this round's stake and, otherwise, is decreased by the same amount. So given a betting strategy b and the initial capital, we can inductively determine the corresponding PAYOFF FUNCTION, or MARTINGALE, d_b by applying the equations

$$d_b(w0) = d_b(w) - b(w) \cdot d_b(w), \qquad d_b(w1) = d_b(w) + b(w) \cdot d_b(w) \ .$$

Intuitively speaking, the payoff $d_b(w)$ is the capital the player accumulates until the end of round $|w|$ by betting on a sequence that has the word w as a prefix.

Conversely, any function d from words to nonnegative reals that for all words w satisfies the fairness condition

$$d(w) \ = \ \frac{d(w0) + d(w1)}{2} \tag{1}$$

determines an initial capital $d(\lambda)$ and a betting function b.

Definition 1. *A martingale d* SUCCEEDS *on a sequence A if d is unbounded on the prefixes of A, i.e., if*

$$\limsup_{m\to\infty} \ d(A|\{0,\ldots,m\}) \ = \ \infty \ .$$

A martingale d is COMPUTABLE if it is confined to rational values and there is a Turing machine that on input w outputs an appropriate finite representation of $d(w)$. Computable martingales are considered in recursion-theoretical settings [1,20,21,24], while in connection with complexity classes one considers martingales that in addition are computable within appropriate resource-bounds [2, 14,15,17].

Definition 2. *A sequence is* REC-RANDOM *if no computable martingale succeeds on it.*

Definition 3. *A martingale d has the* EFFECTIVE SAVINGS PROPERTY *if there is a computable function f from words to nonnegative rationals such that*

(i) $f(w) \le d(w)$ *for all words* w,
(ii) f *is nondecreasing, i.e.,* $f(w) \le f(v)$ *whenever* w *is a prefix of* v,
(iii) for any sequence A, d succeeds on A iff f is unbounded on the prefixes of A.

Remark 4 *For every computable martingale d_0 there is a computable martingale d with initial capital 1 that is equivalent to d_0 (i.e., d succeeds on exactly the same sequences as d_0) and has the effective savings property.*

The construction of the martingale d is well-known and works, intuitively speaking, by putting aside one unit of capital every time the capital reaches a certain threshold, while from then on using the remainder of the capital in order to bet according to the initial martingale.

In order to define Martin-Löf random sequences [16], let $\mathrm{W}_0, \mathrm{W}_1, \ldots$ be the standard enumeration of the computably enumerable sets [22].

Definition 5. *A class $\mathcal{N}$ is a* MARTIN-LÖF NULL CLASS *if there exists a computable function g from $\mathbb{N}$ to $\mathbb{N}$ such that for all $i > 0$,*

$$\mathcal{N} \subseteq \mathrm{W}_{g(i)}\{0,1\}^\infty \quad \textit{and} \quad \lambda(\mathrm{W}_{g(i)}\{0,1\}^\infty) \leq \frac{1}{2^i}\,. \tag{2}$$

A sequence is MARTIN-LÖF RANDOM *if it is not contained in any Martin-Löf null class.*

In the situation of Definition 5, we say that the $\mathrm{W}_{g(i)}$ form a MARTIN-LÖF TEST that covers the class $\mathcal{N}$, i.e., a class is covered by a Martin-Löf test if and only if it is a Martin-Löf null class.

By definition, a subclass $\mathcal{N}$ of Cantor space has uniform measure 0 if there is *any* sequence of sets $V_1, V_2, \ldots$ such that (2) is satisfied with $\mathrm{W}_{g(i)}$ replaced by V_i. Thus the concept of a Martin-Löf null class is indeed an effective variant of the classical concept of a class that has uniform measure 0 and, in particular, any Martin-Löf null class has uniform measure 0. It can be shown that the union of all Martin-Löf null classes is again a Martin-Löf null class [5, Section 6.2]. Equivalently, there is a UNIVERSAL MARTIN-LÖF TEST $U_1, U_2, \ldots$ that covers the class of all sequences that are not Martin-Löf random.

3 Characterizing Computable Randomness by Tests

By definition, a sequence is Martin-Löf random if it cannot be covered by a Martin-Löf test $U_1, U_2, \ldots$, i.e., is not contained in the intersection of the classes $U_k\{0,1\}^\infty$. For such a test, for given index k and word w the value

$$m_k^w = \lambda(w\{0,1\}^\infty \cap U_k\{0,1\}^\infty) \tag{3}$$

can be effectively approximated from below by simply enumerating the words in U_k that extend w. If we choose the $U_1, U_2, \ldots$ to be an appropriate universal Martin-Löf test, then by a result of Kučera [10,11, Lemma 8.1 and Lemma 2], we can compute a nontrivial upper bound on m_k^w, i.e., one that is strictly less than $1/2^{|w|}$, in case m_k^w is indeed strictly smaller than the latter number. If one requires that the m_k^w can be effectively approximated to any given precision, one obtains a characterization of the concept of Schnorr randomness, where the

Schnorr random sequences are a strict superclass of the computable random sequences [1,21,25].

We show in this section that the concepts of computable random sequence and of computable null class can be characterized by Martin-Löf tests where the m_k^w can be appropriately bounded from above by means of a computable probability distribution on Cantor space. This gives a positive answer to a question of Ambos-Spies and **Kučera**, who have asked whether computable randomness can be characterized in terms of Martin-Löf tests [1, Open Problem 2.6]. Downey, Griffiths, and Laforte obtained independently and earlier a characterization that is essentially equivalent to ours but is formulated in different terms; for an account of their result see Downey and Hirschfeldt [7].

Definition 6. *A* MASS DISTRIBUTION *on Cantor space is a mapping μ from words to reals such that for any word w holds $\mu(w) = \mu(w0) + \mu(w1)$. A* PROBABILITY DISTRIBUTION *(on Cantor space) is a mass distribution μ where $\mu(\lambda) = 1$. A mass distribution μ is* COMPUTABLE *if μ is rational-valued and there is an effective procedure that on input w computes $\mu(w)$.*

Mass distributions and martingales are essentially the same concept [9] where, in particular, the additivity condition $\mu(w) = \mu(w0) + \mu(w1)$ corresponds to the fairness condition (1). More precisely, given a mass distribution μ, the function $w \mapsto 2^{|w|}\mu(w)$ is a martingale with initial capital $\mu(\lambda)$ and conversely, given a martingale d, the function $w \mapsto d(w)/2^{|w|}$ is a mass distribution.

Proposition 7. *A class $\boldsymbol{C}$ has computable measure 0 if and only if there is a Martin-Löf test $U_1, U_2, \ldots$ and a computable probability distribution μ such that*

(i) $\boldsymbol{C}$ is contained in the intersection of the classes $U_k\{0,1\}^\infty$,

(ii) for any $k > 0$ and any word w, the Lebesgue measure of the intersection of the cylinder generated by w with $U_k\{0,1\}^\infty$ is at most $\mu(w)/k$, i.e.,

$$\lambda(w\{0,1\}^\infty \cap U_k\{0,1\}^\infty) \leq \frac{\mu(w)}{k}. \tag{4}$$

Proof. First, assume that we are given a class $\boldsymbol{C}$ that has computable measure 0. By Remark 4, pick a computable martingale d that succeeds on every sequence in $\boldsymbol{C}$ and has the effective savings property via some computable, nondecreasing function f. In order to obtain a Martin-Löf test $U_1, U_2, \ldots$ and a probability distribution μ as required, let for all $k > 0$

$$U_k = \{w | f(w) \geq k \text{ and } f(v) < k \text{ for all proper prefixes } v \text{ of } w\},\ \mu(w) = \frac{d(w)}{2^{|w|}}.$$

In order to prove assertion (i), fix any sequence X in $\boldsymbol{C}$. Then d succeeds on X and, in particular, f is unbounded on the prefixes of X; hence for all $k > 0$ there is some prefix of X in U_k and X is contained in the intersection of the $U_k\{0,1\}^\infty$.

In order to prove assertion (ii), fix any index $k > 0$ and word w. First assume that w has some prefix w_0 in U_k. In this case assertion (ii) holds because by construction and assumption on f, we have

$$k \leq f(w_0) \leq f(w) \leq d(w), \qquad \text{hence } \frac{1}{2^{|w|}} \leq \frac{\mu(w)}{k}. \tag{5}$$

Next consider any word w that does not have a prefix in U_k and let U_k^w be the set of all words in U_k that extend w. Then assertion (ii) holds because we have

$$\lambda(w\{0,1\}^\infty \cap U_k\{0,1\}^\infty) \leq \sum_{v \in U_k^w} \frac{1}{2^{|v|}} \leq \sum_{v \in U_k^w} \frac{\mu(v)}{k} \leq \frac{\mu(w)}{k},$$

where the inequalities hold, from left to right, because U_k and hence U_k^w are prefix-free, by (5), and by additivity of probability measures.

Next assume that we are given a Martin-Löf test $U_1, U_2, \ldots$ and a probability distribution μ as in the proposition. By the discussion following Definition 6, the function $w \mapsto \mu(w)2^{|w|}$ is a computable martingale, which succeeds on any sequence in $\boldsymbol{C}$ because by assumption any such sequence is contained in the intersection of the $U_k\{0,1\}^\infty$, i.e., has prefixes in all the U_k, where for all words w in U_k we have $\mu(w)2^{|w|} \geq k$ according to (4). □

4 The Power of Martingale Processes

Hitchcock and Lutz [8] remark that the term martingale has different meanings in probability theory and theoretical computer science. In order to compare the two notions, consider Lebesgue measure on Cantor space and for a given function d from words to the real numbers, let ξ_m^d be the random variable defined by $\xi_m^d(X) = d(X(0)\ldots X(m-1))$, i.e., for a a martingale d and a random sequence X chosen uniformly at random, $\xi_0^d(X), \xi_1^d(X), \ldots$ is just the sequence of capital values that are reached on the prefixes of X when betting according to d. The martingale concept from Section 2, which is the one usually considered in theoretical computer science, can then be equivalently characterized by reformulating the fairness condition (1) as follows. A function d from words to reals is a martingale iff for any word w, the conditional expectation of $\xi_{|w|+1}^d$, given that X extends w, is equal to $\xi_{|w|}^d$, i.e.,

$$\mathbf{E}[\xi_{|w|+1}^d \mid w\{0,1\}^\infty] = \xi_{|w|}^d. \tag{6}$$

On the other hand, in probability theory, a sequence $\xi_0, \xi_1, \ldots$ of random variables is called a martingale if for all m, the expectation of ξ_m is finite and

$$\mathbf{E}[\xi_{m+1}^d \mid \xi_0^d = c_0, \ldots, \xi_m^d = c_m] = \xi_m^d \qquad \text{for all reals } c_0, \ldots, c_{m-1}. \tag{7}$$

Hitchcock and Lutz call a function d from words to reals a MARTINGALE PROCESS iff the sequence $\xi_0^d, \xi_1^d, \ldots$ is a martingale in the sense of probability theory; they remark that in the fairness conditions (6) and (7) for martingales and martingale processes, the average is taken over all sequences with the same *bit history* and the same *capital history,* respectively. Two words have the same bit history if they are identical. With a martingale d understood, two words v and w have the same capital history, $v \approx_d w$ for short, if both words have the same length and we have $d(v') = d(w')$ for any two prefixes v' of v and w' of w that have the same length. The relation $\approx_d$ is an equivalence relation on words and will be

called d-equivalence. Condition (8) is an equivalent reformulation of the fairness condition (7) for martingale processes in terms of d-equivalence [8].

$$2 \sum_{\{v:\, v \approx_d w\}} d(v) \quad = \quad \sum_{\{v:\, v \approx_d w\}} [d(v0) + d(v1)] \qquad \text{for all words } w\,. \tag{8}$$

Among other results, Hitchcock and Lutz derive the following facts about martingale processes. Every class that can be covered by a martingale processes is also a Martin-Löf null class. On the other hand, every computable martingale is by definition also a martingale process, and there is a rec-random sequence on which a computable martingale process succeeds. The latter assertion is obtained by proving that the following result of An. A. Muchnik [18, Theorem 9.1] remains true with computable martingale processes in place of nonmonotonic partial computable martingales. If almost all prefixes w of a sequence have Kolmogorov complexity of at most $|w| - \log |w|$, then some nonmonotonic partial computable martingale succeeds on the sequence. These results of Lutz and Hitchcock show that computable martingales are strictly less powerful than computable martingale processes and the latter are in turn at most as powerful as Martin-Löf tests. Accordingly, the concepts of random sequence and of class of measure zero defined in terms of martingale processes are intermediate between the corresponding concepts for computable martingales and Martin-Löf tests. We state in Theorem 9 that martingale processes are in fact as powerful as Martin-Löf tests, hence the corresponding concepts of randomness and measure are the same. In the proof of this theorem, we use a construction of martingale processes that is described in Remark 8.

Remark 8 *Given a computably enumerable set U such that $U\{0,1\}^\infty$ has Lebesgue measure at most 1/2, there is a computable martingale process $\mathbf{d}$ that doubles its initial capital $d(\lambda) = 1$ on every sequence in $U\{0,1\}^\infty$, i.e., every such sequence has a prefix w where $d(w) = 2$. Furthermore, an index for $\mathbf{d}$ can be computed from a c.e. index for U.*

In order to construct a martingale process $\mathbf{d}$ as required, let $w_1, w_2, \ldots$ be an effective enumeration of U and let $\widetilde{U}$ be the set of all words w such that some prefix of w is enumerated into U within at most $|w|$ steps of this enumeration. The set $\widetilde{U}$ is computable and closed under extensions, and $U\{0,1\}^\infty$ coincides with $\widetilde{U}\{0,1\}^\infty$. The martingale process $\mathbf{d}$ is defined inductively for words of length $m = 0, 1, \ldots$, where $d(\lambda) = 1$. For any word w of length m that is in $\widetilde{U}$, let $d(w) = 2$, while all other words w of length m are assigned identical values $d(w)$ in such a way that the fairness condition for martingale processes is not violated. The latter is alway possible because the Lebesgue measure of $U\{0,1\}^\infty$ is at most 1/2, details are left to the reader.

It remains to show that $\mathbf{d}$ is a martingale process, which amounts to show for any given word w that the equation in the fairness condition (7) is satisfied. By construction, first, we have $d(v) = 2$ for all extensions of v of any word w where $d(w) = 2$, hence if $d(w) = 2$, then $d(w0) = d(w1) = 2$ and the same holds for all words v that are d-equivalent to w; the equation in the fairness condition

follows. Second, an easy induction shows that all strings w of the same length such that $d(w)$ differs from 2 are d-equivalent. But for the words w in such an equivalence class, the values $d(w0)$ and $d(w1)$ are simply chosen such that the fairness condition is not violated.

Theorem 9. *A class is a Martin-Löf null class if and only if there is a martingale process that succeeds on any sequence in the class. In particular, a sequence is Martin-Löf random if and only if it is random with respect to martingale processes.*

Proof. The second assertion in the theorem follows from the first one because for both concepts of randomness involved, a sequence R is random iff the singleton class $\{R\}$ can be covered by an admissible martingale. Concerning the first assertion, recall that Lutz and Hitchcock [8] have shown that any class on which a martingale process succeeds is a Martin-Löf null class. So it suffices to show that there is a martingale process d that succeeds on the class covered by some universal Martin-Löf test $U_1, U_2, \ldots$.

For any word w, let

$$V_w = \{u \in U_{|w|+1} : u = wv \text{ for some } v\}\,, \qquad V_w^- = \{v : wv \in V_w\}\,. \tag{9}$$

The Lebesgue measure of $U_{|w|+1}\{0,1\}^\infty$ is at most $1/2^{|w|+1}$, thus the Lebesgue measure of V_w^- is at most 1/2 and similar to the construction in Remark 8 we obtain a martingale process d_w^- that doubles its initial capital 2^k on all words in V_w^-. Then the martingale process $d(w,k)$ defined by

$$d(w,k)(u) = \begin{cases} d_w^-(v) & \text{in case } u = wv, \\ 2^k & \text{in case } w \text{ is not a prefix of } u. \end{cases}$$

is computable, attains on λ, as well as on w the value 2^k, and doubles the latter capital on any sequence in $V_w\{0,1\}^\infty$.

The martingale process d we are looking for can be viewed as working in phases $s = 1, 2, \ldots$, where during phase k it copies the values of some martingale process of the form $d(w,k)$. For a given sequence X, the phases are as follows. During the first phase, d agrees with $d(\lambda, 0)$, this phase lasts up to and including the least prefix w_1 of X such that the latter martingale attains the value 2, i.e., has doubled its initial capital. The second phase starts at w_1, there overlapping with the first phase; the second phase lasts up to and including the least prefix w_2 of X on which $d(w_1, 1)$ has doubled. The further phases are similar, i.e., during phase k, the martingale process d agrees with $d(w_k, k)$, and phase k ends and phase $k+1$ starts as soon as the capital has reached 2^{k+1} at word w_{k+1}. By construction, d is computable and is unbounded on any sequence that is contained in all the $U_k\{0,1\}^\infty$.

It remains to show that d is a martingale process. Any d-equivalence class is the disjoint union of finitely many $d(w_k,k)$-equivalence classes where d agrees on each such class with the corresponding martingale process $d(w_k,k)$; hence d satisfies the fairness condition for martingale processes because all these $d(w_k,k)$ satisfy this condition. □

5 Universal Martin-Löf Tests

Definition 10. *A* COMPUTABLY ENUMERABLE REAL, C.E. REAL *for short, is a real which is the limit of a nondecreasing computable sequence of rational numbers.*

A real is MARTIN-LÖF RANDOM, RANDOM *for short, if its binary expansion has the form* $0.R(0)R(1)\ldots$ *for some Martin-Löf random sequence* R.

A real is a CHAITIN Ω REAL *if it is the halting probability of some universal prefix-free Turing machine.*

Definition 11 (Solovay [23]). *Let* $(a^s : s \geq 0)$ *and* $(b^s : s \geq 0)$ *be computable sequences of rationals which converge nondecreasingly to* α *and* β, *respectively.*

1. $(a^s : s \geq 0)$ DOMINATES $(b^s : s \geq 0)$ *if there is a positive constant* c *such that for all* $s \geq 0$, $c(\alpha - a^s) > (\beta - b^s)$.
2. $(a^s : s \geq 0)$ *is* UNIVERSAL *if it dominates every computable nondecreasing sequence of rationals.*
3. α *is* Ω-LIKE *if it is the limit of a universal nondecreasing computable sequence of rationals.*

By a celebrated result, which Calude [5] attributes to work of Calude, Hertling, Khoussainov, and Wang, of Chaitin, of Kučera and Slaman, and of Solovay, a c.e. real is random iff it is a Chaitin Ω real iff it is Ω-like; for proofs and references see Calude [5].

Among other results, Kučera and Slaman [12] show, first, that for any universal Martin-Löf test $U_1, U_2, \ldots$, the Lebesgue measure of the classes $U_k\{0,1\}^\infty$ are Martin-Löf random reals and, second, that given any c.e. Martin-Löf random real r there is a universal Martin-Löf test such that the sum of the measures $U_k\{0,1\}^\infty$ is r. These results are complemented by Theorem 14 below, the main result of this section, which states that for any sequence $r_1, r_2, \ldots$ of reals that are random and uniformly c.e., there is a universal Martin-Löf test $U_1, U_2, \ldots$ such that the Lebesgue measure of $U_k\{0,1\}^\infty$ is just r_k. Furthermore, the proof of Theorem 14 can be adapted to yield a simplified proof of the second mentioned result by Kučera and Slaman.

Remark 12. In their proof that every c.e. random real is Ω-like, Kučera and Slaman [12, Theorem 2.1] actually show that for any real r that is random and c.e. there is not just some computable universal sequence that converges nondecreasingly to r but in fact any computable sequence that converges nondecreasingly to r is universal.

With an enumeration of a computably enumerable set U understood, we write U^s for the set of words that enter U during the first s steps of the enumeration, and we write u and u^s for the Lebesgue measure of $U\{0,1\}^\infty$ and $U^s\{0,1\}^\infty$, respectively.

Lemma 13. *Let r be a c.e. random real and let $(r^s : s \geq 0)$ be any universal sequence that converges to r. Let $(U_j : j \geq 1)$ be a universal Martin-Löf test with uniform computable enumerations $(U_j^s : j \geq 1)$ such that U_j^s is empty in case $s \leq j$. Then there is an index k such that for every stage s,*

$$r - r^s > \sum_{j \geq k} (u_j - u_j^s).$$

Proof. Choose a computable, nondecreasing, and unbounded sequence $c_1, c_2, \ldots$ of positive rationals such that

$$\sum_{j \geq 1} c_j 2^{-j} < \infty \text{ , hence } \sum_{j \geq 1} c_j u_j < \infty$$

by the measure condition on the components of a Martin-Löf test. We then have that $\widetilde{u} = \sum_j c_j u_j$ is finite and thus is a c.e. real with approximation

$$\widetilde{u}^s = \sum_{j=1}^{s} c_j u_j^s = \sum_{j=1}^{\infty} c_j u_j^s \, .$$

Accordingly, by Remark 12 and assumption on r and the r^s, there is a nonzero constant c such that for every stage s,

$$c(r - r^s) > \widetilde{u} - \widetilde{u}^s = \sum_{j=1}^{\infty} c_j u_j - \sum_{j=1}^{\infty} c_j u_j^s = \sum_{j=1}^{\infty} c_j (u_j - u_j^s).$$

Since the c_j are unbounded, we may let k be the minimal index such that $c < c_k$. Then we have

$$r - r^s > \frac{1}{c} \sum_{j \geq k} c_j (u_j - u_j^s) > \frac{c_k}{c} \sum_{j \geq k} (u_j - u_j^s) > \sum_{j \geq k} (u_j - u_j^s),$$

and the lemma is proved. □

Theorem 14. *Let $(r_n : n \geq 1)$ be a uniformly c.e. sequence of random reals with $r_n < 2^{-n}$ for every n. Then there is a universal Martin-Löf test $(A_n : n \geq 1)$ such that for each n, $\lambda(A_n\{0,1\}^\infty) = r_n$.*

Proof. For fixed n, we describe an enumeration of component A_n of the universal Martin-Löf test $(A_n : n \geq 1)$; the enumeration of A_n is uniform in n and an uniform enumeration of the random reals $(r_n : r \geq 1)$, hence the sets A_n are indeed uniformly computably enumerable. For the sake of notational simplicity, in the following construction of the component A_n we drop the index n and just write A. Also r_n is abbreviated to r.

By Remark 12, the given uniform enumeration of the r_n yields a universal nondecreasing computable sequence $(r^s : s \geq 1)$ of rationals that converges to r.

Fix a universal Martin-Löf test $(U_j : j \geq 1)$, where we may assume that U_j^s is empty in case $s \leq j$.

We enumerate A by recursion on stages s. Let A^s denote the finite set of words that are enumerated into A during the first s stages. At stage s we aim to add, if necessary, certain elements to A such that the measure of $A^s\{0,1\}^\infty$ will be equal to r^s. For a start, we try to enumerate as many words from $U_1^s, \ldots, U_s^s$ into A as possible on the condition that the measure of $A\{0,1\}^\infty$ does not get greater than r^s. More precisely, for $j \leq s$, the words from U_j^s are enumerated into A if all sets $U_{j+1}^s, \ldots, U_s^s$ have already been enumerated into A and if the above measure condition is not violated after *all* words in U_j^s will have entered A. Formally, let m be the minimal index such that

$$l^s := \lambda\left(\left(A^{s-1} \cup \bigcup_{j\geq 0} U_{m+j}^s\right)\{0,1\}^\infty\right) \leq r^s.$$

Since U_{s+j}^s is empty for all $j \geq 0$, we have $m \leq s$ and m is the index such that exactly the sets $U_m^s, U_{m+1}^s, \ldots, U_s^s$ may be enumerated into A without exceeding measure r^s. In case $l^s = r^s$ we are done. Now suppose otherwise. Then we will add further elements to A in order to make the measure of $A^s\{0,1\}^\infty$ equal to r^s.

In case $m = 1$, choose any set $F(s)$ of words that are incomparable with any word already enumerated into A such that the class $F(s)\{0,1\}^\infty$ has measure $r^s - l^s$; enumerate the words in $F(s)$ into A.

In case $m > 1$, as long as there is no word $w \in U_{m-1}^s$ where the measure of the union of $A\{0,1\}^\infty$ and $w\{0,1\}^\infty$ is greater than r^s, enumerate arbitrary words from U_{m-1}^s into A; by choice of m, eventually there will be such a word w. In this situation, let d denote the difference between r^s and the measure of $A\{0,1\}^\infty$. We may suppose that the rational d is given by its dyadic representation, in which there are only finitely many digits different from zero. Now it is easy to see that there is a set E of words of equal length which extend w such that for a suitable subset E' of E, the measure of $(A \cup E')\{0,1\}^\infty$ is equal to r^s. Note that it suffices to choose the set E such that the words contained in it are longer than any word in A and longer than t, where 2^t is the denominator of the dyadic representation of d. Enumerate E' into A. This concludes the construction of the set A, and hence of $(A_n : n \geq 1)$.

By construction, the measure of $A\{0,1\}^\infty$ is increased at every stage s by $r^s - r^{s-1}$, where the r^s converge to r, hence we have

$$\lambda(A\{0,1\}^\infty) = r. \tag{10}$$

Next we argue that there is an index k such that

$$A\{0,1\}^\infty \supseteq \bigcup_{j\geq k} U_j\{0,1\}^\infty, \tag{11}$$

where it suffices to show that for some index k and infinitely many stages s

$$A^s\{0,1\}^\infty \supseteq \bigcup_{j\geq k} U_j^s\{0,1\}^\infty. \tag{12}$$

By choice of the U_j^s, equation (12) is trivially satisfied for any k and $s = k$. So it remains to demonstrate that there is an index k such that for any s for which (12) is true there is $t > s$ such that (12) remains true with s replaced by t.

Since $(U_j : j \geq 0)$ is a universal Martin-Löf test and $(r^s : s \geq 0)$ is a universal sequence of rationals, we may apply Lemma 13. That is, there is an index k such that at every stage s,

$$r - r^s > \sum_{j \geq k} \Big(\lambda(U_j\{0,1\}^\infty) - \lambda(U_j^s\{0,1\}^\infty) \Big). \tag{13}$$

Now fix any s that satisfies (12) and let v be the minimum index strictly larger than s such that

$$r^v - r^s > \sum_{j \geq k} \Big(\lambda(U_j^v\{0,1\}^\infty) - \lambda(U_j^s\{0,1\}^\infty) \Big).$$

Note that such an index v exists by (13) and because the r^s converge to r. In case (12) is satisfied with s replaced by v we are done. Otherwise, by choice of v, there must be a stage t strictly between s and v such that during stage t a word w is enumerated into A such that $w\{0,1\}^\infty$ is not a subclass of the union of the classes $U_k^t\{0,1\}^\infty, U_{k+1}^t\{0,1\}^\infty, \ldots$. By construction, this can only be the case if during stage t all words in the sets $U_k^t, U_{k+1}^t, \ldots$ are enumerated into A, i.e., in this case (12) is satisfied with s replaced by t. This shows that there are infinitely many stages s such that (12) is satisfied, which in turn shows that (11) is true.

In summary, a uniformly c.e. sequence $(A_n : n \geq 1)$ was constructed such that for each set $A = A_n$ and for the corresponding random real $r = r_n$, the conditions (10) and (11) are satisfied. By (11), the intersection of all $U_k\{0,1\}^\infty$ is a subclass of the intersection of all $A_n\{0,1\}^\infty$. Since $(U_j : j \geq 1)$ was chosen as a universal Martin-Löf test, we have that $(A_n : n \geq 1)$ is a universal Martin-Löf test, too. By (10), the measure of each class $A_n\{0,1\}^\infty$ is equal to r_n. □

Remark 15 *From the proof of Theorem 14, we obtain a somewhat simpler proof of the result of Kučera and Slaman [12] that given any c.e. Martin-Löf random real r there is a universal Martin-Löf test such that the sum of the measures $U_k\{0,1\}^\infty$ is r. Such a test can be constructed similar to the construction of the set A in in the proof of Theorem 14, except that now words that correspond to different components U_k are put into different components of the constructed test and that we can assume that we know the index k, hence we never have to add words corresponding to components with an index less than k.*

References

1. K. Ambos-Spies and A. Kučera. Randomness in computability theory. In P. A. Cholak et al. (eds.), *Computability Theory and Its Applications. Current Trends and Open Problems.* Contemporary Mathematics 257:1–14, American Mathematical Society (AMS), 2000.

2. K. Ambos-Spies and E. Mayordomo. Resource-bounded measure and randomness. In A. Sorbi (ed.), *Complexity, Logic, and Recursion Theory,* p. 1-47. Dekker, New York, 1997.
3. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I,* Springer, 1995.
4. C. S. Calude, *Information and Randomness,* Springer-Verlag, 1994.
5. C. S. Calude. *A characterization of c.e. random reals.* Theoretical Computer Science 271:3-14, 2002.
6. R. Durrett. *Essentials of Stochastic Process,* Springer, 1999.
7. R. Downey and D. Hirschfeldt. *Algorithmic Randomness and Complexity.* Manuscript, 2003.
8. J. M. Hitchcock and J. H. Lutz. Why computational complexity requires stricter martingales. Theory of computing systems, to appear. A preliminary version appeared in P. Widmayer et al. (eds.), 29th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 2380:549–560, Springer, 2002.
9. J. M. Hitchcock. Small Spans in Scaled Dimension. IEEE Conference on Computational Complexity 2004, to appear.
10. A. Kučera. Measure, Π^0_1-classes and complete extensions of PA. In: H.-D. Ebbinghaus et al. (eds.), *Recursion Theory Week.* Lecture Notes in Mathematics 1141:245-259, Springer, 1985.
11. A. Kučera. On the use of diagonally nonrecursive functions. In: H.-D. Ebbinghaus et al. (eds.), *Logic Colloquium '87.* Studies in Logic and the Foundations of Mathematics 129:219-239, North-Holland, 1989.
12. A. Kučera and T. A. Slaman. *Randomness and recursive enumerability.* SIAM Journal on Computing, 31(1):199–211, 2001.
13. M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications,* second edition. Springer, 1997.
14. J. H. Lutz. Almost everywhere high nonuniform complexity. *Journal of Computer and System Sciences* 44:220–258, 1992.
15. J. H. Lutz. The quantitative structure of exponential time. In L. A. Hemaspaandra and A. L. Selman (eds.), *Complexity Theory Retrospective* II, p. 225–260, Springer, 1997.
16. P. Martin-Löf. The definition of random sequences. *Information and Control* 9(6):602–619, 1966.
17. E. Mayordomo. *Contributions to the Study of Resource-Bounded Measure.* Doctoral dissertation, Universitat Politècnica de Catalunya, Barcelona, Spam, 1994.
18. An. A. Muchnik, A. L. Semenov, and V. A. Uspensky. Mathematical metaphysics of randomness. *Theoretical Computer Science* 207:263–317, 1990.
19. P. Odifreddi. *Classical Recursion Theory.* North-Holland, Amsterdam, 1989.
20. C.-P. Schnorr. A unified approach to the definition of random sequences. *Mathematical Systems Theory* 5:246–258, 1971.
21. C.-P. Schnorr. *Zufälligkeit und Wahrscheinlichkeit.* Lecture Notes in Mathematics 218, Springer, 1971.
22. R. I. Soare. *Recursively Enumerable Sets and Degrees.* Springer, 1987.
23. R. M. Solovay. *Draft of a Paper (or Series of Papers) on Chaitin's Work....* Manuscript, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1974.
24. S. A. Terwijn. *Computability and Measure.* Doctoral dissertation, Universiteit van Amsterdam, Amsterdam, Netherlands, 1998.
25. Y. Wang. A Separation of two randomness concepts. *Information Processing Letters,* 69:115–118, 1999.

A Polynomial Quantum Query Lower Bound for the Set Equality Problem

Gatis Midrijānis*

University of Latvia, Raiņa bulvāris 19, Riga, Latvia.
Fax: +371-7820153, gatis@zzdats.lv

Abstract. The set equality problem is to tell whether two sets A and B are equal or disjoint under the promise that one of these is the case. This problem is related to the Graph Isomorphism problem. It was an open problem to find any $\omega(1)$ query lower bound when sets A and B are given by quantum oracles. We will show that any error-bounded quantum query algorithm that solves the set equality problem must evaluate oracles $\Omega(\sqrt[5]{\frac{n}{\ln n}})$ times, where $n = |A| = |B|$.

1 Introduction, Motivation, and Results

The amazing integer factoring algorithm of Shor [14] and search algorithm of Grover [7] show that to find quantum lower bounds is more that just a formality. The most popular model of quantum algorithms is the query (oracle) model. Thus, also quantum lower bounds are proved in the query model. There are developed methods that offer tight or nearly tight lower bound for some problems, however for some other problems not. Recently Aaronson [1] found a new method how to get tight quantum query lower bounds for some important problems, for example, the collision problem. This was an open problem since 1997. Aaronson's method uses symmetrization over the input and therefore can be hard to apply to the problems with asymmetric input. The set equality problem is an example of such problem and it remaind unsolved.

In this paper we will find a quantum lower bound for the set equality problem by reduction. We will reduce the collision problem to the set equality problem, therefore getting quantum query lower bound for the set equality problem.

Let assure ourselves that the set equality problem is related with Graph Isomorphism problem. We are given two graphs G_1, G_2 and we want to establish whether there exists permutations p_1, p_2 over vertices of graphs such that permutated graphs $p_1(G_1), p_2(G_2)$ are equivalent (graphs G_1, G_2 are isomorphs). Let P_i denote the set of all graphs gotten by some permutation over graph G_i's vertices ($i \in \{0, 1\}$). It is easy to see that if graphs are isomorphs then $P_1 = P_2$, but if not, then $P_1 \cap P_2 = \emptyset$. Therefore, if one can distinguish between those

* Research supported by Grant No.01.0354 from the Latvian Council of Science, and Contract IST-1999-11234 (QAIP) from the European Commission.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 996–1005, 2004.

cases, then he can solve the Graph Isomorphism problem. Since there are $n!$ permutations for a graph with n vertices , the sizes of P_1, P_2 can be superpolynomial over the number of vertices of graphs G_1, G_2.

Let $[n]$ denote the set $\{1, 2, ..., n\}$.

Definition 1 *Let $a : [n] \mapsto [N]$ and $b : [n] \mapsto [N]$ be the functions. Let A be the set of all $a's$ images $A := a([n]) := \{a(1), a(2), ..., a(n)\}$ and $B := b([n]) := \{b(1), b(2), ..., b(n)\}$. There is the promise that either $A = B$ or $A \cap B = \emptyset$.*

*Let **the general set equality** problem denote the problem to distinguish these two cases, if functions a and b are given by quantum oracles.*

By use of Ambainis' [2] method it is simple ([11]) to prove $\Omega(\sqrt{n})$ lower bound for the general set equality problem. However, this approach works only if every image can have very many preimages. Graph theorists think that the Graph Isomorphism problem, when graphs are promised not to be equal with themselves by any nonidentical permutation, still is very complex task. This limitation lead us to the set equality problem where a and b are one-to-one functions.

Definition 2 *Let **one-to-one set equality** problem denote the general set equality problem under promise that $a(i) \neq a(j)$ and $b(i) \neq b(j)$ for all $i \neq j$.*

Finding $\omega(1)$ quantum query lower bound for the set equality problem was posed an open problem by Shi[13]. Despite $\omega(1)$ lower bound for the one-to-one set equality problem remaining unsolved task, $\Omega(\frac{n^{1/3}}{\log^{1/3} n})$ quantum query lower bound was showed [11] for a problem between these two problems when $|a^{-1}(x)| = O(\log n)$ and $|b^{-1}(x)| = O(\log n)$ for all images $x \in [N]$.

In this paper we will show the polynomial quantum query lower bound for the most challenging task: the one-to-one set equality problem.

Theorem 3 *Any error-bounded quantum query algorithm $\mathcal{A}$ that solves the one-to-one set equality problem must evaluate functions $\Omega(\sqrt[5]{\frac{n}{\ln n}})$ times.*

The rest of the paper will be organized as follows. In the section 2 will be notations and previous results that we will use. In the section 3 we will preview the main idea of the proof of Theorem 3. The section 4 will start the proof, the section 5 will prepare for continuing proof and the section 6 will finish the proof.

2 Preliminaries

2.1 Quantum Query Algorithms

The most popular model of the quantum computing is a query (or oracle, or black box) model where the input is given by the oracle. For more details, see a survey by Ambainis [3] or a textbook by Gruska [8]. In this paper we are able to skip them because our proof will be based on reduction to solved problems.

In this paper we consider only the worst case complexity for error-bounded quantum algorithms. Thus, without loss of generality, we can assume that any

quantum algorithm makes the same number of queries for any input. If we say that algorithm has two input functions a and b then for technical reasons somewhere it can be comprehend with one input function denoted as (a, b).

One of the most amazing quantum algorithms is Grover's search algorithm ([7]). It shows how a given $x_1 \in \{0,1\}, x_2 \in \{0,1\}, ..., x_n \in \{0,1\}$ to find the i such that $x_i = 1$ with $O(\sqrt{n})$ queries under promise that there exists at most one such i.

This algorithm can be considerably generalized to so called amplitude amplification [6]. Using amplitude amplification one can make good quantum algorithms for many problems till the quadratic speed-up over classical algorithms.

By straightforward use of amplitude amplification we get quantum algorithm for the general set equality problem making $O(\sqrt{n})$ queries and quantum algorithm for the one-to-one set equality problem making $O(n^{1/3})$ queries. Therefore our lower bound probably is not tight.

2.2 Quantum Query Lower Bounds

There are two main approaches to get good quantum query lower bounds. The first is Ambainis' [2] quantum adversary method, other is lower bound by polynomials introduced by Beals et al. [5] and substantially generalized by Aaronson [1], Shi [13] and others. Although explicitly we will use only Ambainis' method, the lower bound we will get by the reduction to the problem, solved by polynomials' method.

The basic idea of the adversary method is, if we can construct a relation $R \subseteq X \times X$, where X and Y consist of 0-instances and 1-instances and there is a lot of ways how to get from an instance in X to an instance in Y that is in the relation and back by flipping various variables, then query complexity must be high.

Theorem 4 *[2] Let $f(x_1, ..., x_n)$, be a function of n variables with values from some finite set and X, Y be two sets of inputs such that $f(x) \neq f(y)$ if $x \in X$ and $y \in Y$. Let $R \subset X \times Y$ be such that*

- *For every $x \in X$, there exist at least m different $y \in Y$ such that $(x, y) \in R$.*
- *For every $y \in Y$, there exist at least m' different $x \in X$ such that $(x, y) \in R$.*
- *For every $x \in X$ and $i \in \{1, ..., n\}$, there are at most l different $y \in Y$ such that $(x, y) \in R$ and $x_i \neq y_i$.*
- *For every $y \in Y$ and $i \in \{1, ..., n\}$, there are at most l' different $x \in X$ such that $(x, y) \in R$ and $x_i \neq y_i$.*

Then, any quantum algorithm computing f uses $\Omega(\sqrt{\frac{mm'}{ll'}})$ queries.

Actually, original Ambainis' formulation was about {0, 1}-valued variables but we can use any finite set as it is implied by the next, more general theorem in Ambainis' paper [2].

2.3 The Collision Problem

Finding $\omega(1)$ quantum lower bound for the collision problem was an open problem since 1997. In 2001 Scott Aaronson [1] solved it by showing polynomial lower bound. Later his result was improved by Yaoyun Shi [13]. Recently, Shi's result was extended by Samuel Kutin [10] and by Andris Ambainis [4] in another directions.

Below is an exact formulation of the collision problem due to Shi[13].

Definition 5 *Let $n > 0$ and $r \geq 2$ be integers with $r|n$, and let a function f of domain size n be given as an oracle with the promise that it is either one-to-one or r-to-one. Let* ***r-to-one*** *collision problem denote the problem to distinguishing these two cases.*

Shi [13] showed following quantum lower bound for the r-to-one collision problem.

Theorem 6 *[13] Any error-bounded quantum algorithm that solves r-to-one collision problem must evaluate the function $\Omega((n/r)^{1/3})$ times.*

Kutin [10] and Ambainis [4] extended his result for functions with any range.

2.4 Notations

Let $F^* := F^*(n, N)$ denote the set of all partial functions from $[n]$ to $[N]$. Then any $f^* \in F^*$ can be conveniently represented as a subset of $[n] \times [N]$, i.e., $f^* = \{(i, f^*(i)) : i \in dom(f^*)\}$.

For a finite set $K \subseteq \mathbf{Z}^+$, let $SG(K)$ denote the group of permutations on K. For any integer $k > 0$, $SG(k)$ is a shorthand for $SG([k])$. For each $\sigma \in SG(n)$ and $\tau \in SG(N)$ define $\Gamma_\tau^\sigma : F^* \to F^*$ as

$$\Gamma_\tau^\sigma(f^*) := \{(\sigma(i), \tau(j)) : (i, j) \in f^*\}, \forall f^* \in F^*.$$

3 The Idea Behind the Proof

The rest of the paper is proof of the Theorem 3. In this section we will discuss the main idea behind this proof. The key is to reduce some problem with known quantum query lower bound to the one-to-one set equality problem. Unfortunately, a simple reduction does not work. Therefore we must make a chain of reductions and in the end get the problem, which can be solved by arbitrary methods.

The problem, which we will try to reduce to the one-to-one set equality problem, is the collision problem. All steps of reduction will be probabilistic. One of that steps Midrijanis[11] used to prove quantum query lower bound for modified set equality problem. We will conclude that any quantum query algorithm that solves the one-to-one set equality problem either solves the collision problem or some other problem that will be presented later. For the collision problem we

have a quantum query lower bound and for this other problem we will prove it using Ambainis' adversary method. This implies lower bound for the one-to-one set equality problem.

Unfortunately, since these reductions are probabilistic ones, but Theorem 4 tells about ordinary functions, a lot of technical work must be done to provide the correctness of the last reduction. We will analyze properties of those reductions and show, informally, that they are very similar (in sense of query complexity).

There will be two kinds of reduction from the collision problem to the set equality problem. Let f denote r-to-one function which the collision problem has in the input. From f we will randomly get two functions, a and b. The both reductions will randomly permutate range and domain of f and divide domain into 2 disjoint halves. The first reduction will takes those halves of domain as domains for functions a and b. The second reduction will take only the first half for both functions, just it will make additional permutation over domain for both functions.

Informally, it is clear that both reductions makes "almost" equal pair of functions a and b whenever r is big "enough". We will show that any quantum algorithm that can make distinction between them must make "quite many" queries. On the other hand, every algorithm for the set equality problem that don't make distinction between them can be used to solve the collision problem that is proved to be hard.

4 Framework of the Proof

We have some n and $1 < r < n$, such that $2|n$ and $r|n$. From the conditions of the collision problem we have function $f : [n] \to [N]$ with promise that f is either one-to-one or r-to-one. Let us choose random variables $\sigma \in SG[n], \sigma_1 \in SG[n/2], \sigma_2 \in SG[n/2]$ and $\tau \in SG[N]$.

With **complementary** reduction we will denote the process deriving functions a and b such that $a(i) = \Gamma_\tau^\sigma(f)(i)$ and $b(i) = \Gamma_\tau^\sigma(f)(n/2+i)$ for all $i \leq n/2$.

With **equivalent** reduction we will denote the process deriving functions a and b such that $a(\sigma_1(i)) = \Gamma_\tau^\sigma(f)(i)$ and $b(\sigma_2(i)) = \Gamma_\tau^\sigma(f)(i)$ for all $i \leq n/2$.

Lemma 7 *For any quantum algorithm $\mathcal{A}$ that solves the set equality problem with T queries either there exists quantum algorithm that solves r-to-one collision and makes $O(T)$ queries or there exists quantum algorithm that makes distinction between complimentary and equivalent reduction and makes $O(T)$ queries.*

Proof. This tabular shows the acceptance probability of algorithm $\mathcal{A}$ running on a and b.

reduction's type \ function's f type	one-to-one	r-to-one
complimentary	$p_1^c > 4/5$	p_2^c
equivalent	$p_1^e < 1/5$	p_2^e

There are two possibilities. If $p_2^e \geq 2/5$ or $p_2^c \leq 3/5$ then algorithm $\mathcal{A}$ can be used to solve the collision problem. But if $p_2^e < 2/5$ and $p_2^c > 3/5$ then algorithm

$\mathcal{A}$ can be used to make distinction between complimentary and equivalent reduction. □

In the next sections we will prove the following lemma:

Lemma 8 *Any quantum algorithm $\mathcal{A}$ that makes distinction between complimentary and equivalent reduction makes $\Omega(\sqrt{\frac{r}{\log n}})$ queries.*

Choosing $r = n^{2/5}\log^{3/5} n$ Lemma 7 together with Lemma 8 and Theorem 6 will finish the proof of Theorem 3.

5 The Lower Bound of Distinction, Preparation

In this section we will start to prove Lemma 8. Informally, both reductions, complementary and equivalent, make "quite similar" pairs of functions. So we have to define what means "similar" and to proof exactly how similar. Also, Theorem 4 deals with ordinary input not distributions over inputs, therefore we will need to formulate ordinary problem and reduce it to ours.

In this subsection we will investigate properties of both reductions.

We will speak only about pairs of functions (a, b) that can be result of either complementary or equivalent reduction with nonzero probability $p(a, b) > 0$. We will investigate what pairs (a, b) can appear.

For any function a, which is in some pairs, let $INV(a) := (a_i | 0 \leq i \leq r) := (a_0, a_1, ..., a_{r-1}, a_r)$ denote the tuple where a_i is the number of image's elements $x \in [N]$, such that cardinality of the set of preimages of x is i, formally $a_i := \#x(|a^{-1}(x)| = i)$ for $0 < i \leq r$ and $a_0 := \frac{n}{r} - \sum_{i=1}^{r} a_i$, where $\frac{n}{r}$ is just the total count of images.

Let $INV(a, b)$ denote $(INV(a), INV(b))$. $INV(a, b)$ is quite good way to describe the structure of some pair of functions (a, b) because of many reasons. Firstly, one can see, that $INV(a, b) = INV(\Gamma_\tau^{\sigma_1}(a), \Gamma_\tau^{\sigma_2}(b))$ for any pair of functions (a, b) and any $\sigma_1 \in SG[n/2], \sigma_2 \in SG[n/2]$ and $\tau \in SG[N]$.

Also, the probability for any pair (a, b) to appear after reduction $p(a, b)$ depends only on $INV(a, b)$. Moreover, if there exists pairs of functions (a_1, b_1) and (a_2, b_2) such that $INV(a_1, b_1) = INV(a_2, b_2)$ then there exists variables $\sigma_1 \in SG[n/2], \sigma_2 \in SG[n/2]$ and $\tau \in SG[N]$ such that $(a, b) = (\Gamma_\tau^{\sigma_1}(a), \Gamma_\tau^{\sigma_2}(b))$.

Now we will show, that, for any pair (a, b), $INV(a)$ and $INV(b)$ are closely related. For any $INV(a) = (a_0, a_1, ..., a_{r-1}, a_r)$ let $\overline{INV}(a)$ denote the tuple $(a_{r-i} | 0 \leq i \leq r) := (a_r, a_{r-1}, ..., a_1, a_0)$.

It is evident that for any pair of functions (a, b) that occurs after complementary reduction holds $INV(a) = \overline{INV}(b)$ but for any pair of functions (a, b) that occurs after equivalent reduction holds $INV(a) = INV(b)$.

We will use these facts to show that complementary and equivalent reductions are quite similar, in other words, any functions b_1 and b_2 such that $INV(b_1) = \overline{INV}(b_2)$ differ in many bits only with very small probability. So we will be able to use Ambainis' Theorem 4 about bit's block flip to show lower bound.

Let a be any function that stand in some pair (a,b) with $INV(a) = (a_0, a_1, ..., a_r)$. Let $DISP(INV(a)) := \max_{0 \le i \le r} (|i - r/2| : a_i > 0)$.

Definition 9 *We say that* a *is "bad" (and denote* $BAD(a)$*) if* $DISP(INV(a)) > 15\sqrt{r \ln \frac{n}{r}}$.

Informally, a is bad if there exists image such that after reduction most of its preimages are in domain of either a or b.

It is easy to see that for any pair (a,b) holds: a is bad if and only if b is bad.

This Lemma shows, that the difference between complementary and equivalent reductions is quite small:

Lemma 10 *The sum* $\sum_{(a,b),BAD(a)} p(a,b)$ *is less than small constant if* $r \gg \ln \frac{n}{r}$.

Proof. Let us see only the case when (a,b) occurs after complementary reduction. Let $f : n \to N$ be the function before reduction. Let choose some fixed image j of function f, thus $j \in f([n])$. We say that j is "bad" (denote by $BAD(j)$) if $||a^{-1}(j)| - r/2| > 15\sqrt{r \ln \frac{n}{r}}$. Let p_j denote the probability that j is bad. It is easy to see that for all $j \in f([n])$ p_j is equal with some p. It is easy to see that $BAD(a) \Leftrightarrow BAD(j)$ for some $j \in f([n])$. Therefore probability for a to be bad is less than $n/r * p$ where n/r is the total count of images. Now it remains only to show that $p \ll r/n$.

There are two cases how j can be bad, the first is that $|a^{-1}(j)|$ is too big and the second is that $|a^{-1}(j)|$ is too small. Obviously, that both of these cases holds with similar probability. Let's count the probability that $|a^{-1}(j)|$ is too big. Let enumerate all preimages of j as $x_1, x_2, ..., x_r$. Let χ'_i denote the random variable that is 1 if x_i become a member of domain of a and 0 elsewhere. Let $\chi' := \chi'_1 + \chi'_2 + ... + \chi'_r$. Thus we reach out for $Pr[\chi' > E(\chi') + 15\sqrt{r \ln \frac{n}{r}}]$, since $E(\chi') = r/2$.

Let χ_i denote the random variable that is 1 with probability 1/2 and 0 with probability 1/2. Let $\chi := \chi_1 + ... + \chi_r$. It is easy to see that for all $s \ge r/2 = E(\chi') = E(\chi)$ holds $Pr[\chi' > s] \le Pr[\chi > s]$. Now we can apply Chernoff's inequality

$$Pr[\chi > (1+\epsilon)E(\chi)] \le e^{-\epsilon^2 E(\chi)/3}$$

if $0 \le \epsilon \le 1$. It is easy to see that $E(\chi) = r/2$. Let $\epsilon := 30\sqrt{\frac{\ln \frac{n}{r}}{r}}$. It is easy to see that $\epsilon E(\chi) = 15\sqrt{r \ln \frac{n}{r}}$. It remains to evaluate the probability $e^{-\epsilon^2 E(\chi)/3} \ll r/n$ if $r \gg \ln \frac{n}{r}$. □

6 Proof's Completing

In this section we will reduce the problem to distinguish between complementary and equivalent reduction (with distribution over input) to problem of the ordinary input.

Definition 11 *Let $n > 0$ and $r \geq 2$ be integers such that $n|2$ and $n|r$. Let $a : n/2 \to N$ and $b : n/2 \to N$ be functions given by an oracle, such that the pair (a, b) can occur after complementary or equivalent reduction. $INV(a)$ is known and it is promised that a is not bad. Let ComesFrom problem denote the problem to decide whether the pair (a, b) occurred after complementary or equivalent reduction.*

6.1 Reduction

Lemma 12 *If there exists quantum algorithm $\mathcal{A}$ that makes distinction between complimentary and equivalent reduction with T queries then there exists quantum algorithm $\mathcal{A}'$ that solves ComesFrom problem with O(T) queries.*

Proof. Firstly, we can ignore all pairs (a, b) that have $BAD(a)$ because they appear with very small probability (Lemma 10). If we want to improve the probability we can just repeat $\mathcal{A}$ several times.

Secondly, without loss of generality, we can assume that the accepting probability of $\mathcal{A}$ depends only on $INV(a, b)$. If not, we can modify algorithm $\mathcal{A}$, such that it choose random variables $\sigma_1 \in SG[n/2], \sigma_2 \in SG[n/2]$ and $\tau \in SG[N]$ at the beginning and further just deal with pair of functions $(\Gamma_\tau^{\sigma_1}(a), \Gamma_\tau^{\sigma_2}(b))$.

Thirdly, since $\mathcal{A}$ makes distinction between complimentary and equivalent reduction and for any pair of functions (a, b) depends only on $INV(a, b)$, there exists some $I := INV(a)$ such that $\mathcal{A}$ makes distinction between (a, b_1) such that $INV(b_1) = INV(a) = I$ and (a, b_2) such that $\overline{INV}(b_2) = INV(a) = I$ for any function b_1 and b_2.

It follows, that for this particular I we can solve ComesFrom problem using algorithm $\mathcal{A}$.

□

6.2 Lower Bound for the *ComesFrom* Problem

Lemma 13 *Any quantum algorithm $\mathcal{A}$ that solves the ComesFrom problem makes $\Omega\left(\sqrt{\frac{r}{\log n}}\right)$ queries.*

Proof. Let $I = (a_0, ..., a_r)$ denote the known $INV(a)$. We will use Theorem 4 to prove lower bound quite similarly to Ambainis' [2] proof about lower bound for counting. Let X be the set of all (a, b) such that $INV(b) = INV(a) = I$ and let Y be the set of all (a, b) such that $\overline{INV}(b) = INV(a) = I$.

Let $\Psi := \Psi(I) := \sum_{i:a_i > r/2} a_i - r/2$. Since a is not bad, it implies that $\Psi = O(\frac{n}{r}\sqrt{r \log n})$. 2Ψ is just the number of points that must be changed to to switch from $INV(b)$ to $\overline{INV}(b)$. Let $\Phi := \frac{(2\Psi)!}{\prod_{i:a_i<r/2}(r-2a_i)!}$.

Let R be the set of all $((a, b_1), (a, b_2))$ such that b_1 differs from b_2 exactly in 2Ψ points and $INV(b_1) = \overline{INV}(b_2)$. Then, $m = m' = C_{n/4+\Psi}^{2\Psi}\Phi$ and $l = l' = C_{n/4+\Psi-1}^{2\Psi-1}\Phi$. Therefore,

$$\frac{mm'}{ll'} = \left(\frac{C^{2\Psi}_{n/4+\Psi}\Phi}{C^{2\Psi-1}_{n/4+\Psi-1}\Phi}\right)^2 = \left(\frac{(n/4+\Psi)!(2\Psi-1)!(n/4-\Psi)!}{(2\Psi)!(n/4-\Psi)!(n/4+\Psi-1)!}\right)^2$$

$$= \left(\frac{n/4+\Psi}{2\Psi}\right)^2 = \left(\frac{n}{8\Psi}+\frac{1}{2}\right)^2 = \Omega\left(\left(\frac{n}{\Psi}\right)^2\right) = \Omega\left(\frac{r}{\log n}\right)$$

Now we can apply Theorem 4 and get that any quantum algorithm makes $\Omega\left(\sqrt{\frac{r}{\log n}}\right)$ queries. □

7 Conclusion

We showed a polynomial quantum query lower bound for the set equality problem. It was done by reduction. Arguments that allowed reduction was very specific to the set equality problem. It would be nice to find some more general approach to find quantum query lower bounds for this and other similar problems. Also, it would be fine to make smaller difference between quantum lower and upper bounds for the set equality problem.

Acknowledgments. I am very thankful to Andris Ambainis about introduction and discussions about this problem, quantum lower bounds and quantum computation, and useful comments about this paper, to Inga Černiauskaitė about checking my spelling, to Andrejs Dubrovskis for discussions after my first paper about quantum query lower bound for the set equality problem, to Yufan Zhu about pointing out to mistake in previous version of Lemma's 13 proof and to anonymous referees whose comments helped me to improve presentation of the result.

References

1. S. Aaronson. Quantum lower bound for the collision problem. *In Proceedings Proceedings of ACM STOC'2002, pp. 635-642, 2002.* quant-ph/0111102.
2. A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences,* 64:750-767, 2002. Earlier versions at STOC'00 and quant-ph/0002066.
3. A. Ambainis. Quantum query algorithms and lower bounds. *Proceedings of FOTFS III, to appear.*
4. A. Ambainis. Quantum lower bounds for collision and element distinctness with small range, 2003. quant-ph/0305179
5. R. Beals, H. Buhrman, R. Cleve, M. Mosca, R. de Wolf. Quantum lower bounds by polynomials. *Journal of ACM,* 48: 778-797, 2001. Earlier version at FOCS'98.
6. G. Brassard, P. Høyer, M. Mosca and A. Tapp. Quantum amplitude amplification and estimation, *to appear in AMS Contemporary Mathematics Series Millennium Volume entitled "Quantum Computation & Information".*
7. L. K. Grover. A fast quantum mechanical algorithm for database search. *Journal of ACM,* 2: 212-219, 1996.
8. J. Gruska. Quantum computing. McGraw-Hill, 1999.

9. P. Hoyer, J. Neerbek, Y. Shi. Quantum lower bounds of ordered searching, sorting and element distinctness. *Algorithmica,* 34:429-448, 2002. Earlier versions at ICALP'01 and quant-ph/0102078.
10. S. Kutin. Quantum lower bound for the collision problem, 2003, quant-ph/0304162.
11. G. Midrijanis. Quantum lower bounds of set equality problems, 2003, quant-ph/0309068.
12. Y. Shi. Lower bounds of quantum black-box complexity and degree of approximating polynomials by influence of Boolean variables. *Information Processing Letters* 75:79-83, 2000.
13. Y. Shi. Quantum lower bounds for the collision and the element distinctness problems *Proceedings of the 43rd Annual Symposium on the Foundations of Computer Science,* pp. 513-519, 2002.
14. P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *In proceedings: 35th Annual Symposium on Foundations of Computer Science, November 20-22, 1994, Santa Fe, New Mexico,* pages 124-134, IEEE Computer Society Press, 1994.

Succinct Representations of Functions

J. Ian Munro and S. Srinivasa Rao

School of Computer Science, Univ. of Waterloo, Waterloo ON, Canada N2L 3G1.
{imunro,ssrao}@uwaterloo.ca

Abstract. We investigate the problem of succinctly representing an arbitrary function, $f : [n] \rightarrow [n]$ so that $f^k(i)$ can be computed quickly for any i and any (positive or negative) integer power k. We give a representation that takes $(1+\epsilon)n \lg n + O(1)$ bits and computes arbitrary positive powers in constant time. It can also be used to compute $f^k(i)$, for any negative integer k, in $O(1 + |f^k(i)|)$ time.

1 Introduction

We consider the problem of representing arbitrary functions from $[n]$[1] to $[n]$, so that queries for $f^k(i)$ for any integer k can be answered efficiently, where for all $i \in [n]$, $f^0(i) = i$ and for any $k > 0$, $f^k(i) = f(f^{k-1}(i))$ and $f^{-k}(i) = \{j | f^k(j) = i\}$. The case when f is one-one (or onto), i.e., when f is a permutation on $[n]$, is considered in [10]. Here we consider the more general case.

Since there are n^n functions from $[n]$ to $[n]$, any representation takes at least $\lceil n \lg n \rceil$ bits of space to store an arbitrary function. Our aim is to obtain a representation that takes space close to this information-theoretic lower bound of $n \lg n$ bits and supports queries for arbitrary powers in optimal time.

The most natural representation of a function f is to store the sequence $f(i)$, for $i = 0, \ldots, n-1$. Using this representation one can find $f^k(i)$ in k steps, for $k \geq 1$. To facilitate the computation in constant time, one could store $f^k(i)$ for all i and k ($|k| \leq n$, along with some extra information), but that would require $\Theta(n^2 \lg n)$ bits. The most natural compromise is to retain the values of $f^k(i)$ where $|k| \leq n$ is a power of 2. This $\Theta(n(\lg n)^2)$ bit representation easily yields a logarithmic evaluation scheme. Unfortunately we are a factor of $\lg n$ from the minimal space representation and still have a $\Theta(\lg n)$ time algorithm. Also, this representation does not support queries for the negative powers of f efficiently. Our main result removes this logarithmic factor from both the time and the space terms, giving $f^k(i)$ (for both positive and negative integer values of k) in optimal time while using essentially minimum space.

Along the way, we show that an unlabeled n-node rooted tree can be represented using the optimal $2n + o(n)$ bits of space to answer level-ancestor queries in constant time. This is done by extending the succinct tree representation of Munro and Raman [9].

We assume a standard *word RAM* model with word size $\Theta(\lg n)$ bits, where n is the size of the problem under consideration.

[1] for positive integers n, we define $[n]$ to be the set of integers $\{0, \ldots, n-1\}$

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1006–1015, 2004.

2 Level Ancestor Queries

In this section we consider the problem of supporting level ancestor queries in a static tree. Given a rooted tree T, the *level ancestor problem* is to preprocess T to answer queries of the following form: Given a vertex v and an integer $i > 0$, find the ith vertex on the path from v to the root, if it exists. Solutions with $O(n)$ preprocessing and $O(1)$ query time were given by Dietz [4], Berkman and Vishkin [3] and by Alstrup and Holm [1]. A much simpler solution was given by Bender and Farach-Colton [2]. For a tree on n nodes, all these solutions require $\Theta(n \lg n)$ bits of space to store the additional data structures, apart from the $\Theta(n \lg n)$ bits used to store the tree. Here we give a solution that stores the tree using (optimal) $2n$ bits of space, and uses auxiliary structures of $o(n)$ bits to support the queries in $O(1)$ time. Geary et al. [5] have also given a structure that takes $2n + o(n)$ bits of space and supports level-ancestor queries in $O(1)$ time. Another useful feature of our solution (which we need in our function representation) is that it also supports finding the 'level-successor' of a node, i.e., the node to the right of a given node on the same level (if it exists), in constant time.

A high-level view of our structure and the query algorithm is as follows: we construct a structure, A, that supports finding any ancestor of a node which is within a height of, say $\lg^2 n$ from the given node. We also construct another structure, B, which supports level-ancestor queries on nodes whose depths are multiples of $\lg^2 n$, and whose heights are at least $\lg^2 n$. To support a query, structure A is first used to find the closest ancestor of the given node, whose depth is a multiple of $\lg^2 n$. Then structure B is used to find the ancestor which is the least descendent of the required node, whose depth is a multiple of $\lg^2 n$. Structure A is again used to find the required node from this node. The choice of different powers of $\lg n$ in the structures given below are somewhat arbitrary, and could be fine-tuned to slightly improve the lower-order term.

As in [9], we represent the given n node tree using a balanced parenthesis sequence of length $2n$, by visiting the nodes of the tree in depth first order and writing an open parenthesis whenever a node is first visited, and a closing parenthesis when a node is visited after all its children have been visited. Thus, each node has exactly one open and one closing parenthesis corresponding to it. Hereafter, we also refer a node by the position of either the open or the closing parenthesis corresponding to it in the parenthesis sequence of the tree. We store an auxiliary structure of size $o(n)$ bits that answers the following queries in $O(1)$ time (see [9] for details):

- close(i): find the position of the closing parenthesis that matches the open parenthesis at position i.
- open(i): find the position of the open parenthesis that matches the closing parenthesis at position i.
- excess(i): find the difference between the number of open parentheses and the number of closing parentheses from the beginning up to the position i.

Note that the excess of a position i is simply the depth of the node i in the tree. We also need the following operation to support level-ancestor queries efficiently:

– next-excess(i, k): find the least position $j > i$ such that excess$(j) = k$.

We only support this query for excess$(i) - O(\lg^c n) \leq k \leq$ excess(i) for some fixed constant c, to be chosen later. Observe that next-excess(i, k) gives the pre-order successor of node i whose depth is k, if such a node exists. In other words, it gives – (a) the ancestor of i at depth k, if $k <$ depth(i), and (b) the next node after i in the level-order traversal of the tree, if $k =$ depth(i).

Remark: One can also support the next-excess(i, k) operation for excess$(i) < k \leq$ excess$(i) + O(\lg^c n)$, which can be used to find a descendent of node i whose depth is k, if such a node exists. In our structure, we use a simpler solution for this operation by re-ordering the nodes of the tree.

One important substructure that is used to support all these operations is a bit vector that supports finding the number of 1's before a given position i (rank(i)) and the position of the i-th 1 (select(i)) in it. We refer to such a bit vector as an *indexable bit vector*. It is known [6,8] that a bit vector of length n can be augmented with $o(n)$ bits to support the rank and select operations in constant time.

We now describe the auxiliary structure to support the next-excess query in constant time using $o(n)$ bits of extra space. We split the parenthesis sequence corresponding to the tree into blocks of size $b = \lg^4 n$. For each block, we store the following structure to support the queries within the block (i.e., if the answer lies in the same block as the query element) in $O(1)$ time:

We build a complete tree with branching factor $\sqrt{\lg n}$ (and hence constant height) with the elements of the block at the leaves. Considering an open parenthesis as +1 and a closed parenthesis as –1, we define the weight of an internal node as the sum of the values at all the leaves in the subtree rooted at that node. We store the prefix sums of the weights of all the children at all internal nodes except at the level immediately above the leaves. Since the leaves are labeled either +1 or –1, one can find the prefix sums of any node one level above the leaves, using a lookup table that stores the prefix sums for every possible of sequence of +1s and –1s of length $\sqrt{\lg n}$. (See [11] for the details of a similar structure.) Thus, the size of this structure is $O(b \lg\lg n/\sqrt{\lg n}) = o(b)$ bits. Using this structure, given any position i in the block and a number k, we can find the position next-excess(i, k) in constant time, if it exists within the block.

Since the excess values of two consecutive positions differ only by one, the set containing the excess values of all the positions in a block forms a single range of integers. We store this range information for each block, which requires $o(n)$ bits for the entire sequence. Let $[e_1, e_2]$ be the range of excess values in a block B. Then for each i, $e_1 - \lg^2 n \leq i < e_1$, we store the least position to the right of block B whose excess is i, in an array A_B.

In addition, for each i, $e_1 \leq i \leq e_2$, we store a pointer to the first block B' to the right of block B such that B' has a position with excess i. Then we remove all multiple pointers (thus each pointer corresponds to a range of excesses instead of

just one excess). The graph representing these pointers between blocks is planar. [One way to see this is to draw the graph on the Euclidean plane so that the vertex corresponding to the j-th block B, with excess values in the range $[e_1, e_2]$, is represented as a vertical line with end points (j, e_1) and (j, e_2). Then, there is an edge between two blocks B and B' if and only if the vertices (vertical lines) corresponding to these blocks are 'visible' to each other (i.e., a horizontal line connecting these two vertical lines at some height does not intersect any other vertical lines in the middle).] Since the number of edges in a planar graph on n vertices is $O(n)$, the number of these inter-block pointers (edges) is $O(n/b)$ as there are n/b blocks (vertices). The total space required to store all the pointers and the array A_B is $O(n \lg^3 n/b) = o(n)$ bits.

Thus, each block has a set of pointers associated with a set of ranges of excess values. Given an excess value, we need to find the range containing that value in a given block (if the value belongs to the range of excess values in that block), to find the pointer associated with that range. For this purpose, we store the following auxiliary structure: If a block has more than $\lg n$ ranges associated with it (i.e., if the degree of the node corresponding to a block in the graph representing the inter-block pointers is more than $\lg n$), then we store a bit vector for that block that has a 1 at the position where a range starts, and 0 everywhere else. We also store an auxiliary structure to support rank queries on this bit vector in constant time. Since there are at most $n/(b \lg n)$ blocks containing more than $\lg n$ ranges, the total space used for storing all these bit vectors together with the auxiliary structures is $o(n)$ bits. If a block has at most $\lg n$ ranges associated with it, then we store the lengths of these ranges (from left to right) using the searchable partial sum structure of [11], that supports predecessor queries in constant time. This requires $o(b)$ bits for every such block, and hence $o(n)$ bits overall.

Given a query next-excess(i, k), let B be the block to which the position i belongs. We first check to see if the answer lies within the block B (using the prefix sums tree structure mentioned above), and if so, we output the position. Otherwise, let $[e_1, e_2]$ be the range of excess values in B. If $e_1 - \lg^2 n \leq k < e_1$, then we can find the answer from the array A_B. Otherwise (when $e_1 \leq k \leq e_2$), we first find the pointer associated with the range containing k (using either the bit vector or the partial sum structure, associated with the block) and use this pointer to find block containing the answer. Finding the answer, given the block in which it is contained, is done using the prefix sums tree structure stored for each block.

Thus, using these structures, we can support next-excess(i, k) for any i and excess$(i) - \lg^2 n \leq k \leq$ excess(i) in constant time. In other words, given any node in the tree we can find its k-th ancestor, for $k \leq \lg^2 n$, and also the next node in the level-order traversal of the tree in constant time. To support general level ancestor queries, we do the following:

First, mark all nodes that are at a depth which is a multiple of $\lg^2 n$ and whose height is at least $\lg^2 n$. There are $O(n/\lg^2 n)$ such nodes. We store all these marked nodes as a tree (preserving the ancestor relation among these

nodes) and store a linear space (hence $o(n)$-bit) structure that supports level-ancestor queries in constant time [2]. Note that one level in this tree corresponds to exactly $\lg^2 n$ levels in the original tree. We also store the correspondence between the nodes in the original tree and those in the tree containing only the marked nodes.

A query for level-ancestor(x, k), the ancestor of x at height k from x (i.e., at depth depth$(x) - k$), is answered as follows: If $k \leq \lg^2 n$, we find the answer using a next-excess query. Otherwise, we first find the least ancestor of x which is marked using at most two next-excess queries (the first one to find the least ancestor whose depth is a multiple of $\lg^2 n$, and the next one, if necessary, to find the ancestor whose height is at least $\lg^2 n$). From this we find the highest marked ancestor of x which is a descendent of the answer node, using the level-ancestor structure for the marked nodes. The required ancestor is found from this node using another next-excess query, if necessary. Thus we have:

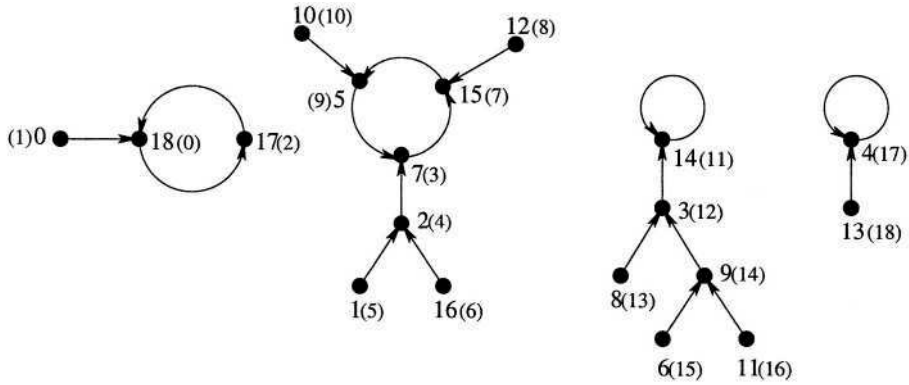
Theorem 1. *Given an unlabeled rooted tree with n nodes, there is a structure that represents the tree using $2n + o(n)$ bits of space and answers* level-ancestor *queries in O(1) time.*

3 Representing Functions

Given a function, we equate it to a digraph in which every node is of outdegree 1, and represent this graph efficiently. We then show how to compute arbitrary powers of the function by translating them into the navigational operations on the digraph. More specifically, given an arbitrary function $f : [n] \rightarrow [n]$, consider the digraph $G_f = (V, E)$ obtained from it, where $V = [n]$ and $E = \{\langle i, j\rangle : f(i) = j\}$. In general this digraph consists of a set of connected components where each component has a directed cycle with each vertex being the root of a (possibly single node) directed tree, with edges directed towards the root. (See Figure 1(a)). We refer to each connected component as a gadget.

The main idea of our representation is to store the structure of the graph G_f that supports the required navigational operations efficiently. In addition, we also need to store the labels of the nodes in G_f. To support the queries for powers of f, we need to find the node corresponding to a label, and also the label corresponding to a node efficiently. For this purpose, we first re-name the nodes of the graph G_f according to the representation we choose to represent its structure, and store the correspondence between these new names and the original names of the vertices. We store this correspondence using the representation of a permutation that supports forward and inverse queries efficiently.

More specifically, let $C_1, C_2, \ldots C_p$ be the gadgets in G_f and let $T_i^1, T_i^2, \ldots T_i^q$ be the trees in the i-th gadget, where we start at an arbitrary tree in a gadget and index them in the increasing order as we move along the cycle edges in the forward direction. All the nodes in the i-th gadget C_i are numbered with numbers in the range from $(\sum_{r=1}^{i-1} |C_r|)$ to $(\sum_{r=1}^{i} |C_r|) - 1$, and all the nodes in the j-th tree of i-th gadget, T_i^j, are numbered with numbers in the range from



(a) Graph representation of the function $f(x) = (x^2 + 2x - 1) \bmod 19$, for $0 \leq x \leq 18$. The vertex labels in the brackets correspond to the function g obtained by renaming the vertices

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	5	4	12	17	9	15	3	13	14	10	16	8	18	11	7	6	2	0

(b) Permutation defining the isomorphism between G_f and G_g

(( ))	( )	((( ) ( )))	(( ))	(( ))	((( ) (( ) ( ))))	(( ))
1 0 0 0	0 0	1 0 0 0 0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0
1 0 0 0	1 0	1 0 0 0 0 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0

(c) Parenthesis representation and the bit vectors indicating the starting positions of the gadgets and the trees (auxiliary structures are not shown)

Fig. 1. Representing a function

$(\sum_{r=1}^{i-1} |C_r| + \sum_{r=1}^{j-1} |T_i^r|)$ to $(\sum_{r=1}^{i-1} |C_r| + \sum_{r=1}^{j} |T_i^r|) - 1$. Within a tree, we number the nodes in the increasing order of their pre-order numbers starting from the root (before numbering the tree nodes, we modify the tree by re-ordering the children, if necessary, as explained later). This numbering gives a re-naming of the nodes from the set $[n]$. See Figure 1 (a) for an example.

This graph with the new names for the nodes corresponds to another function, say $g : [n] \to [n]$. We store the correspondence between the numbering of the nodes in G_g and the actual names of the nodes in G_f (i.e., the isomorphism between G_f and G_g) as a permutation, say π. A query for $f^k(i)$ is answered using the fact that $f^k(i) = \pi^{-1}(g^k(\pi(i)))$. We use a succinct representation of a permutation that supports forward and inverse queries efficiently (see [10]) to store the permutation π. This permutation dominates the cost of our entire

representation, as the rest of the structure only requires $O(n)$ bits of space. We now concentrate on the representation of g that supports $g^k(i)$ queries efficiently.

We represent each tree in G_g using the parenthesis representation of Munro and Raman [9], and store auxiliary structures to support level-ancestor queries. We first re-order the nodes of each tree such that the leftmost path of any subtree is the longest path in that subtree. This helps in finding all the descendents of a given node at a given level in time proportional to the number of such nodes as follows: we first find the leftmost descendent in the subtree at the given level, if it exists, in constant time, as this path will be represented by a sequence of open parentheses in the parenthesis representation of the tree. From this node, we can find all the nodes at this level by using the next-excess operation to find the next node at this level and checking whether the node is a descendent of the given node. (Since all the nodes in a subtree are together in the parenthesis representation, checking whether a node x is a descendent of another node y can be done in constant time by comparing either the open or closing parenthesis position of x with the open and closing parenthesis positions of y.)

For convenience, we pad the representation of an m node tree to cm bits, for some fixed constant $c > 2$ (note that the parenthesis representation takes $2m$ bits and the auxiliary structures take $o(m)$ bits). This enables us to find the size of a tree using the length of its representation and vice versa. We concatenate the tree representations of all the trees in a gadget, starting with the least numbered root, in the increasing order. Then, we concatenate the representations of all the gadgets in the increasing order of their vertex labels. We store indexable bit vectors indicating the starting positions of the representation of each gadget and each tree respectively. These enable us to find the representation of a gadget, and also the roots of the trees in a gadget efficiently. It is easy to see that this entire representation takes $O(n)$ bits of space over all the gadgets.

Given a node in a tree, we can find its k-th successor (i.e., the node reached by traversing k edges in the forward direction), if it exists within the same tree, in constant time using a level-ancestor query. The k-th successor of a node which is the root of a tree can be found efficiently using the indexable bit vector representing the tree roots. By combining these two, we can find the k-th successor of an arbitrary node in a gadget in constant time.

Given a node x in a gadget, if it is not the root of any tree, then we can find all its k-th predecessors (i.e., all the nodes reachable by traversing k edges in the reverse direction) in optimal time using the tree structure by finding all the descendent nodes of x that are k levels below, as explained earlier. Otherwise, if it is the root of a tree, we first identify all the trees which have at least one answer and then find all the answers in each tree (that has at least one answer) using the tree structure.

We now concentrate on the problem of identifying all the trees in the gadget which have at least one answer. For this purpose we store the following structure for each gadget. Let $r_1, r_2, \ldots r_l$ be the roots on the trees in the gadget in the increasing order of their vertex labels (in G_g). Then $\langle r_i, r_{(i \bmod l)+1} \rangle$ is an edge on the cycle. We first remove the nodes in all the trees except the leftmost

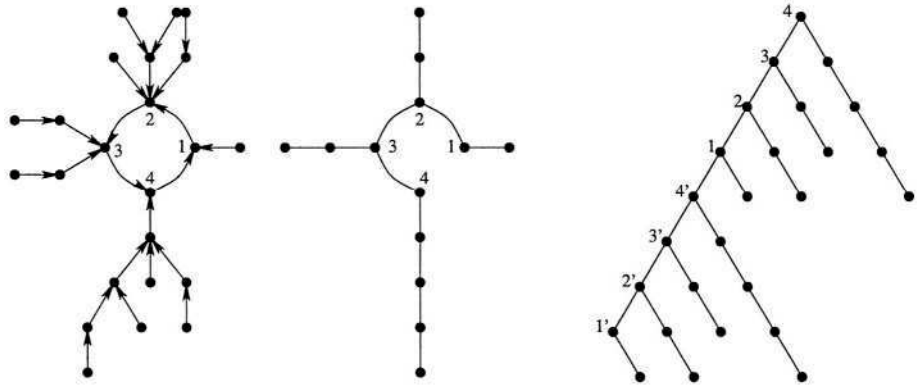


Fig. 2. Construction of the tree T'' for a gadget

(longest) path (paths which are longer than l can be shortened to l), and also remove the edge $\langle r_l, r_1 \rangle$. Consider the rooted tree T obtained by removing the directions of all the remaining edges and making r_l the root. We take another copy T' of T and construct a new tree T'' by making the root r'_l of T', a child of r_1 in T (see Figure 2 for an example).

We store a succinct representation of the tree T'', and store auxiliary structures to support next-excess queries in constant time. Note that each of the trees rooted at the nodes r_j and r'_j, for $1 \leq j \leq l$, are simply paths. Thus, given a node in any of these trees, we can find the respective root in constant time, as these paths will be represented as a sequence of open parentheses followed by a sequence of closing parentheses. (Since we only use this structure to identify all the trees that have an answer for the inverse query, we don't need to store the correspondence between the nodes in this pruned tree and the original graph representing the function.)

Given a node r_i on the cycle, to find all the nodes that are at a distance k from r_i in the backward direction in the graph G_g, we first find all the trees that have at least one answer and then use the tree structure to find all the required nodes in each of the trees. To find all the trees that have at least one answer, we use the above constructed tree T''. The main idea is that if a tree has at least one answer, it has one on its leftmost (longest) path. We start at node r_i in T'' and find all the descendents of it which are depth k from it. For each of these nodes, we output the root (r_j or r'_j) which contains that node. It is easy to see that this gives all the nodes that have at least one answer.

Combining all these, we have:

Theorem 2. *If there is a representation of a permutation on $[n]$ that takes $P(n)$ space to represent a permutation on $[n]$ and supports forward in t_1 time and inverse in t_2 time, then there is a representation of a function from $[n]$ to $[n]$ that takes $P(n)+O(n)$ bits of space and supports $f^k(i)$ in $O(t_1+t_2+|f^k(i)|)$ time, for any integer k and for any $i \in [n]$.*

Using the succinct permutation representation of Munro et al. [10] that takes $(1+\epsilon)n \lg n + O(1)$ bits, for any fixed positive constant ϵ, and supports arbitrary powers (in particular, forward and inverse) in $O(1)$ time, we get:

Corollary 1. *There is a representation of a function $f : [n] \to [n]$ that takes $(1+\epsilon)n \lg n + O(1)$ bits of space for any fixed positive constant ϵ, and supports $f^k(i)$ in $O(1 + |f^k(i)|)$ time, for any integer k and for any $i \in [n]$.*

3.1 Functions with Arbitrary Ranges

So far we considered functions whose domain and range are the same set $[n]$. We now consider functions $f : [n] \to [m]$, and deal with the two cases: (i) $n \geq m$ and (ii) $n < m$. These results can be easily extended to the case when neither the domain nor the range is a subset of the other. We only consider the queries for positive powers.

Case (i) $n \geq m$**:** A function $f : [n] \to [m]$, for $n \geq m$ can be represented by storing the restriction of f on $[m]$ using the representation mentioned in the previous section, together with the sequence $f(m+1), f(m+2), \ldots, f(n)$ stored in an array. Thus we have:

Theorem 3. *If there is a representation of a permutation on $[n]$ that takes $P(n)$ space to represent a permutation on $[n]$ and supports forward in t_1 time and inverse in t_2 time, then there is a representation of a function $f : [n] \to [m]$, $n \geq m$ that takes $(n-m)\lceil \lg m \rceil + P(m) + O(m)$ bits of space and supports $f^k(i)$ in $O(t_1 + t_2)$ time, for any positive integer k and for any $i \in [n]$.*

Case(ii) $n < m$**:** For a function $f : [n] \to [m]$, $n < m$, larger powers (i.e., $f^k(i)$ for $k \geq 2$) are not defined in general (as we might go out of the domain after one application of the function). Let $R = \{i : f(i) \in [n] \text{ for } i \in [n]\}$, $r = |R|$ and $S = [n] \setminus R$. We store the sets R and S using an 'indexable bit vector' (of size $n + o(n)$ bits). Let $R = \{x_1, x_2, \ldots, x_r\}$, where $x_1 < x_2 < \ldots < x_r$. We define a new function $g : [r+1] \to [r+1]$ as follows: $g(r) = r$, and for $0 \leq i < r$,

$$g(i) = \begin{cases} j, & \text{if } f(x_i) = x_j \\ r, & \text{if } f(x_i) \notin R. \end{cases}$$

The function g is stored using the representation of previous section. For $i \notin R$, we store the sequence of $f(i)$ values in the increasing order of i, using $(n-r)\lceil \lg m \rceil$ bits.

A query for $f^k(i)$, for $k \geq 1$ is answered as follows: if $i \notin R$ and $k = 1$, then we look at the answer in the sequence of $f(i)$ values. We use the indexable bit vectors for R and S to index into this sequence. For $i \notin R$ and $k > 1$, $f^k(i)$ is not defined. If $i \in R$ and $k = 1$, we answer the query using the structure for function g. Finally, if $i \in R$ and $k > 1$, then first we find $j = g^{k-1}(i)$. If $j = r$, then the answer is undefined. Otherwise (if $j < r$), the answer is $f(x_j)$.

Thus we have:

Theorem 4. *If there is a representation of a permutation on $[n]$ that takes $P(n)$ space to represent a permutation on $[n]$ and supports forward in t_1 time*

and inverse in t_2 time, then there is a representation of a function $f : [n] \to [m]$, $n < m$ that takes $n \lg m + P(n) + O(n)$ bits of space and supports the queries for $f^k(i)$ (returns the power if defined and –1 otherwise) in $O(t_1 + t_2)$ time, for any positive integer k and for any $i \in [n]$.

4 Open Problems

It is an interesting open problem to design a structure that takes $n \lg n + O(n)$ bits of space to represent a function $f : [n] \to [n]$ and supports arbitrary powers in $O(1)$ time. Note that such a structure is not known even for the special case when f is a permutation. It is also interesting to see if the lower bound for the permutation representation [10] can be strengthened for the function representation.

References

1. S. Alstrup and J. Holm. Improved algorithms for finding level-ancestors in dynamic trees. In *Proceedings of the 27th International Conference on Automata, Language and Programming,* LNCS 1853, 73–84, 2000.
2. M. A. Bender and M. Farach-Colton. The level ancestor problem simplified. In *Proceedings of LATIN,* LNCS 2286, 508–515, 2002.
3. O. Berkman and U. Vishkin. Finding level-ancestors in trees. Journal of Computer and System Sciences **48(2)** 214–230 (1994).
4. P. F. Dietz. Finding level-ancestors in dynamic trees. In *Proceedings of the 2nd Workshop on Algorithms and Data Structures,* LNCS 519, 32–40, 1991.
5. R. Geary, R. Raman and V. Raman. Succinct Ordinal Trees with Level-ancestor Queries. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms,* 1–10, 2004.
6. G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science,* 549–554, 1989.
7. D. E. Knuth. Efficient representation of perm groups. *Combinatorica* **11** 33–43 (1991).
8. J. I. Munro. Tables. In *Proceedings of the Conference on Foundations of Software Technology and Theoretical Computer Science,* LNCS 1180, 37–42, 1996.
9. J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing,* **31** (3):762-776, 2002.
10. J. Ian Munro, R. Raman, V. Raman and S. S. Rao. Succinct representations of permutations. In *Proceedings of the International Conference on Automata, Language and Programming,* LNCS 2719: 345–356, 2003.
11. R. Raman, V. Raman and S. S. Rao. Succinct dynamic data structures. In *Proceedings of the Workshop on Algorithms and Data Structures,* LNCS 2125: 426–437, 2001.

A Note on Karr's Algorithm

Markus Müller-Olm[1]* and Helmut Seidl[2]

[1] FernUniversität Hagen, FB Informatik, LG PI 5, Universitätsstr. 1, 58097 Hagen, Germany
mmo@ls5.informatik.uni-dortmund.de
[2] TU München, Informatik, I2, 85748 München, Germany
seidl@informatik.tu-muenchen.de

Abstract. We give a simple formulation of Karr's algorithm for computing all affine relationships in affine programs. This simplified algorithm runs in time $\mathcal{O}(nk^3)$ where n is the program size and k is the number of program variables assuming unit cost for arithmetic operations. This improves upon the original formulation by a factor of k. Moreover, our re-formulation avoids exponential growth of the lengths of intermediately occurring numbers (in binary representation) and uses less complicated elementary operations. We also describe a generalization that determines all polynomial relations up to degree d in time $\mathcal{O}(nk^{3d})$.

1 Introduction

In 1976, Michael Karr came up with an ingenious algorithm that computes for each program point in a flow graph a vector space of affine relations that hold among the program variables whenever control reaches the program point [6].[1] His algorithm is an iterative fixpoint algorithm that propagates affine spaces through the flow graph and computes for each program point u an affine space that over-approximates the set of run-time states that occur at u, i.e., contains all those run-time states. Hence, affine relationships valid for all states of the computed affine space are also valid for all possible run-time states. Karr represents affine spaces by kernels of affine transformations, i.e., as sets of solutions of linear equation systems. From this representation the affine relations valid for all states in a given affine space can be read off easily.

Finding valid affine relations has many applications. Many classical data flow analysis problems can be conceived as problems about affine relations such as *definite equalities among variables* like $\mathbf{x} = \mathbf{y}$ and *constant propagation.* More general affine relations (such as $2\mathbf{x} + 3\mathbf{y} = 0$) found by automatic analysis routines can also be used as valid assertions in program verification. Leroux uses affine relations for the analysis of counter systems [7]. More applications are discussed in [6,11].

In recent related work [4,8,11] a number of difficulties associated with Karr's algorithm have been observed. Firstly, Karr's algorithm uses quite complicated operations like the transfer function for ("non-invertible") assignments and the union of affine spaces. Secondly, due to the complexity of these operations a straightforward implementation of Karr's algorithm performs $\mathcal{O}(nk^4)$ arithmetic operations in the worst-case

* On leave from Universität Dortmund.

[1] An *affine relation* is a property of the form $a_0 + \sum_{i=1}^{k} a_i \mathbf{x}_i = 0$, where $\mathbf{x}_1, \ldots, \mathbf{x}_k$ are program variables and $a_0, \ldots, a_k$ are elements of the underlying field of values.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1016–1028, 2004.

(where n is the size of the flow graph and k is the number of program variables) and it is not obvious to improve upon this complexity by using standard tricks like semi-naïve fixpoint iteration. Thirdly, the algorithm can lead to exponentially large numbers.

The main contribution of this paper is an extremely simple formulation of Karr's algorithm which solves all three above problems. By using a different representation of affine spaces – we represent an affine space A of dimension l by $l+1$ affine independent points of A – the union operation and the transfer functions become virtually trivial; by using semi-naïve iteration, the complexity goes down to $\mathcal{O}(nk^3)$; and the involved numbers remain of polynomial length. We also show how to generalize our version of Karr's algorithm to determine *polynomial relations,* i.e., properties of the form $p = 0$, where p is a multi-variate polynomial in the program variables $\mathbf{x}_i$.

In this paper we study *affine programs* [11] which differ from ordinary programs in that they have non-deterministic (instead of conditional) branching, and contain only assignments where the right-hand sides either are affine expressions like in $\mathbf{x}_3 := \mathbf{x}_1 - 3\mathbf{x}_2 + 7$ or equal "?" denoting an unknown value. Clearly, our analysis can be applied to arbitrary programs by ignoring the conditions at branchings and simulating input operations and non-affine right-hand sides in assignments through assignments of unknown values. As a byproduct of our considerations we show that Karr's algorithm is *precise* for affine programs, i.e., computes not just some but *all* valid affine relations. While this is kind of folklore knowledge in the field, it has (up to our knowledge) not been formally stated and proved before. Similarly, we show that our extension determines all valid polynomial relations up to a given degree in an affine program.

Related Work. Karr's algorithm has been generalized in different directions. A prominent generalization is the use of polyhedra instead of affine spaces for approximation of sets of program states; the classic reference is Cousot's and Halbwachs' paper [3]. Polyhedra allow us to determine also valid affine inequalities like $3\mathbf{x}_1 + 5\mathbf{x}_2 \leq 7\mathbf{x}_3$. Since the lattice of polyhedra has infinite height, widening must be used to ensure termination of the analysis (see [1] for a recent discussion) – making it unsuitable for precise analyses. Like Karr's original algorithm, analyses using polyhedra suffer from the problem of potentially large numbers.

More recently, we have described an analysis that determines all valid polynomial relations of bounded degree in *polynomial programs* [10,9] with techniques from computable algebra. (In polynomial programs deterministic assignments with polynomial right hand side as well as polynomial disequality guards are allowed.) However, while we can show termination of the analysis we do not know an upper complexity bound.

Gulwani and Necula [4] present a probabilistic analysis for finding affine relations that with a (small) probability yields non-valid affine relations. Unlike the algorithms described so far, however, their algorithm assumes that variables take values in the finite field $\mathbb{Z}_p = \mathbb{Z}/(p\mathbb{Z})$ of natural numbers modulo p, where p is a (large) prime number, instead of natural or rational numbers. This assumption is introduced for two reasons. Firstly, it is needed for the estimation of the error probability. Secondly, it avoids problems with exponentially large numbers. In comparison our version of Karr's algorithm guarantees to yield only valid affine relations and to use only polynomially large numbers despite of working with rational numbers.

Like Karr's algorithm the analyses described so far are intraprocedural algorithms, i.e., they do not treat procedures. Precise *interprocedural* algorithms for affine programs that compute all valid affine or polynomial relations of bounded degree, respectively, are presented in [11]. While these algorithms run in polynomial time, they are asymptotically slower than Karr's, even if we specialize them to the intraprocedural case.

2 Affine Programs

We use a similar notation as in [11]. Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ be the set of variables the program operates on and let $\mathbf{x}$ denote the vector of variables $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_k)$. We assume that the variables take values in $\mathbb{Q}$, the field of rational numbers. Then a *state* assigning values to the variables is conveniently modeled by a **k-dimensional** vector $x = (x_1, \ldots, x_k) \in \mathbb{Q}^k$; x_i is the value assigned to variable $\mathbf{x}_i$. Note that we distinguish variables and their values by using a different font.

For the moment, we assume that the basic statements in the program are *affine assignments of the form* $\mathbf{x}_j := t_0 + \sum_{i=1}^{k} t_i \mathbf{x}_i$ (with $t_i \in \mathbb{Q}$ for $i = 0, \ldots, k$ and $\mathbf{x}_j \in \mathbf{X}$) and that branching is non-deterministic. We show in Sect. 4 how to extend the basic algorithm to non-deterministic assignments $x_i :=?$ and discuss guards in Sect. 7. Let Stmt be the set of affine assignments. Each affine assignment $s \equiv \mathbf{x}_j := t_0 + \sum_{i=1}^{k} t_i \mathbf{x}_i$ induces a transformation, $[\![s]\!]$, on the program state given by $[\![s]\!]x = (x_1, \ldots, x_{j-1}, t_0 + \sum_{i=1}^{k} t_i x_i, x_{j+1}, \ldots, x_k)$. It is easy to see that $[\![s]\!]$ is an affine transformation, i.e., it can be written in the form $[\![s]\!]x = Ax + b$ for a matrix $A \in \mathbb{Q}^{k \times k}$ and a vector $b \in \mathbb{Q}^k$.

An *affine program* is given by a *control flow graph* $G = (N, E, \mathsf{st})$ that consists of: a set N *of program points;* a set of edges $E \subseteq N \times \mathsf{Stmt} \times N$; and a special *entry (or start) point* $\mathsf{st} \in N$.

As common in flow analysis, we use the program's collecting semantics [2] as a reference point for judging the soundness and completeness of Karr's algorithm. The collecting semantics assigns to each program point $u \in N$ the set of all those states that occur at u in some execution of the program. It can be characterized as the least solution of the following constraint system, V, on sets of states, i.e., subsets of $\mathbb{Q}^k$:

$$\begin{aligned} &[\mathrm{V1}] \quad \mathbf{V}[\mathsf{st}] \supseteq \mathbb{Q}^k \\ &[\mathrm{V2}] \quad \mathbf{V}[v] \supseteq f_s(\mathbf{V}[u]), \text{ for each } (u, s, v) \in E, \end{aligned}$$

where the transfer functions f_s are defined by $f_s(X) = \{[\![s]\!]x \mid x \in X\}$. We denote the components of the least solution of the constraint system V (which exists by Knaster-Tarski fix point theorem) by $V[v]$, $v \in N$.

3 The Algorithm

The *affine hull* of a subset $G \subseteq \mathbb{Q}^k$ is the set

$$\mathsf{aff}(G) = \{\sum_{j=0}^{m} \lambda_j x_j \mid m \geq 0, x_j \in G, \lambda_j \in \mathbb{Q}, \sum_{j=0}^{m} \lambda_j = 1\}.$$

In particular, $\mathsf{aff}(G) = G$ whenever G contains at most one element. Whenever $X = \mathsf{aff}(G)$ for some G, we call X an *affine space* and G a set of *generators* for X. If G is a minimal set with $X = \mathsf{aff}(G)$ we call G an *affine basis* of X. The goal of our algorithm is easily stated in terms of the collecting semantics: compute for each program point u the affine hull of the collecting semantics for u, $\mathsf{aff}(V[u])$.

Obviously, aff is a *closure operator,* i.e., it is monotonic and we have, $\mathsf{aff}(X) \supseteq X$ and $\mathsf{aff}(\mathsf{aff}(X)) = \mathsf{aff}(X)$ for all $X \subseteq \mathbb{Q}^k$. It is well-known in abstract interpretation, that the image of a closure operator on a complete lattice is a complete lattice as well (cf., e.g., [2]). By definition, the image of aff consists of the affine subspaces of $\mathbb{Q}^k$. Let us denote this complete lattice by $(\mathbb{D}, \sqsubseteq) = (\{X \subseteq \mathbb{Q}^k \mid X = \mathsf{aff}(X)\}, \subseteq)$. The least element of $\mathbb{D}$ is $\emptyset$ and its greatest element is $\mathbb{Q}^k$. It is well-known that affine spaces are closed under intersection but not under union. Correspondingly, the meet and join operations of the lattice $\mathbb{D}$ are given by the following equations: $\sqcap\mathcal{X} = \cap\mathcal{X}$ and $\sqcup\mathcal{X} = \mathsf{aff}(\cup\mathcal{X})$ for $\mathcal{X} \subseteq \mathbb{D}$. In particular, we have:

Lemma 1. *For all sets $\mathcal{X} \subseteq 2^{\mathbb{Q}^k}$ of subsets of states,* $\mathsf{aff}(\cup\mathcal{X}) = \sqcup\{\mathsf{aff}(X) \mid X \in \mathcal{X}\}$.

The height of $\mathbb{D}$ is $k+1$ as in any strictly increasing chain $A_0 \subset A_1 \subset \ldots$ the dimensions must strictly increase: $\dim(A_0) < \dim(A_1) < \ldots$. Here, the dimension of $\emptyset$ is -1, and the dimension of a non-empty affine space X is the dimension of the linear space $L = \{x - x_0 \mid x_0, x \in X\}$. Thus, the dimensions are bounded by -1 from below and by k from above. (It is easy to construct a strictly increasing chain of length $k+1$.)

Recall that every statement s defines an affine transformation $[\![s]\!]$. Therefore:

Lemma 2. *For all statements s and $X \subseteq \mathbb{Q}^k$,* $\mathsf{aff}(f_s(X)) = f_s(\mathsf{aff}(X))$.

Let $V^\sharp$ be the following constraint system obtained from V by replacing "$\supseteq$" with "$\sqsupseteq$", i.e., switching from the complete lattice of subsets of states to the lattice of affine spaces.

$$[\mathrm{V1}^\sharp] \quad \mathbf{V}^\sharp[\mathsf{st}] \sqsupseteq \mathbb{Q}^k$$
$$[\mathrm{V2}^\sharp] \quad \mathbf{V}^\sharp[v] \sqsupseteq f_s(\mathbf{V}^\sharp[u]), \text{ for each } (u, s, v) \in E.$$

We denote the components of the least solution of $V^\sharp$ over the domain $(\mathbb{D}, \sqsubseteq)$ by $V^\sharp[v]$, $v \in N$. This solution again exists by Knaster-Tarski fixpoint theorem. Lemmas 1 and 2 together with the fact that $\mathsf{aff}(\mathbb{Q}^k) = \mathbb{Q}^k$ imply by standard argumentation from abstract interpretation that the least solution of the abstract constraint system $V^\sharp$ is the precise abstraction of the least solution of the concrete constraint system V, i.e.:

Lemma 3. *For all program points v, $V^\sharp[v] = \mathsf{aff}(V[v])$.*

In order to obtain an effective algorithm we must choose a finitary representation of affine spaces. As mentioned, Karr represents affine spaces by kernels of affine transformation. Instead, we represent an affine space $X \subseteq \mathbb{Q}^k$ by an affine basis of X. This enables us to use semi-naïve fixpoint iteration for computing the solution of constraint system $V^\sharp$. A corresponding algorithm is given in Fig. 1. The algorithm uses an array G indexed by the program points $u \in N$ to store the sets of vectors to become generating sets for $V^\sharp[u]$. Moreover, it uses a workset W in which it holds pairs of the form $(u, x) \in N \times \mathbb{Q}^k$; each pair (u, x) stored in W records that vector x has still to be

```
forall (v ∈ N) G[v] = ∅;
G[st] = {0, e_1, ..., e_k};
W = {(st, 0), (st, e_1), ..., (st, e_k)};
while (W ≠ ∅) {
    (u, x) = Extract(W);
    forall (s, v with (u, s, v) ∈ E) {
        t = [[s]]x;
        if (t ∉ aff(G[v])) {
            G[v] = G[v] ∪ {t};
            W = W ∪ {(v, t)};
        }
    }
}
```

Fig. 1. The base algorithm.

propagated from program point u. We write $\mathbf{0}$ for the zero vector and $e_1, \ldots, e_k$ for the standard basis of the vector space $\mathbb{Q}^k$. The function *Extract(W)* returns an arbitrary element of W and removes it from W.

The idea of semi-naïve fixpoint iteration is to propagate just "increments" instead of full abstract values via the edges of the flow graph. Thus it avoids full re-computation of the transfer functions for new abstract values. In our case a full abstract value is an affine subspace of $\mathbb{Q}^k$ and an "increment" amounts to a new affine independent vector x that is added to a generating set stored for some program point u. The propagation of x via an outgoing edge (u, s, v) is done by applying the concrete semantics of statement s, $[\![s]\!]$, to the vector x, and adding the result to the set of generators stored for the target program point of this edge, v, if it is not already in the affine hull of $G[v]$. Intuitively, this is sufficient because, by Lemma 2, $G' = \{[\![s]\!]x \mid x \in G\}$ is a generating set for $f_s(X)$ if $X = \mathsf{aff}(G)$. Sect. 3.1 contains a more formal correctness argument.

3.1 Correctness

We claim that the algorithm in Fig. 1 computes sets of generators for the affine spaces $V^\sharp[v]$. The proof of this claim is based on two invariants of the **while**-loop:

I1: for all $v \in N$, $G[v] \subseteq V[v]$, and for all $(u, x) \in W$, $x \in V[u]$.
I2: for all $(u, s, v) \in E$, $\mathsf{aff}(G[v] \cup \{[\![s]\!]x \mid (u, x) \in W\}) \sqsupseteq f_s(\mathsf{aff}(G[u]))$.

Both invariants can be easily verified by inspection of the initialization code and body of the **while**-loop. We thus obtain:

Theorem 1. *a) The above algorithm terminates after at most $nk + n$ iterations of the loop (where $n = |N|$ and k is the number of variables).*
b) For all $v \in N$, we have $\mathsf{aff}(G_{\mathrm{fin}}[v]) = V^\sharp[v]$, where $G_{\mathrm{fin}}[v]$ is the value of $G[v]$ upon termination of the algorithm.

Proof. **a)** In each iteration of the loop an entry is extracted from the workset W until the workset is empty. Therefore, the number of loop iterations equals the number of elements that are put to the workset. We observe that a new pair (u, x) is put to the workset only when the affine space $\mathsf{aff}(G[u])$ has been enlarged. In summary, this is also true for the initialization of G and W. Since each strictly ascending chain of affine spaces has length at most $k+1$, we conclude that for every program point u, there are at most $(k+1)$ insertions into W. Since there are at most n program points, the algorithm terminates after at most $n \cdot (k+1)$ iterations of the **while**-loop.

b) In order to show the inclusion $\mathsf{aff}(G_{\mathrm{fin}}[v]) \sqsubseteq V^\sharp[v]$ we note that the loop invariant **I1** implies in particular that $G_{\mathrm{fin}}[v] \subseteq V[v]$ for each $v \in N$. Hence, $\mathsf{aff}(G_{\mathrm{fin}}[v]) \sqsubseteq \mathsf{aff}(V[v]) = V^\sharp[v]$ for each $v \in N$.

In order to prove the reverse inclusion, $\mathsf{aff}(G_{\mathrm{fin}}[v]) \sqsupseteq V^\sharp[v]$, we observe that the invariant **I2** implies that upon termination when the workset W is empty, we have

$$\mathsf{aff}(G_{\mathrm{fin}}[v]) \sqsupseteq f_s(\mathsf{aff}(G_{\mathrm{fin}}[u]))$$

for all $(u, s, v) \in E$. We also have $\mathsf{aff}(G_{\mathrm{fin}}[\mathsf{st}]) \sqsupseteq \mathsf{aff}(\{\mathbf{0}, e_1, \ldots, e_k\}) = \mathbb{Q}^k$ because the elements $\mathbf{0}, e_1, \ldots, e_k$ assigned to $G[\mathsf{st}]$ by the initialization are never removed. Hence the family of values $(\mathsf{aff}(G_{\mathrm{fin}}[v]))_{v \in N}$ satisfies all the constraints of the constraint system $V^\sharp$. As the values $V^\sharp[v]$ are the components of the *least* solution of $V^\sharp$, this implies $\mathsf{aff}(G_{\mathrm{fin}}[v]) \sqsupseteq V^\sharp[v]$ for all $v \in N$. □

3.2 Complexity

In order to reason about the complexity of the algorithm, we consider a uniform cost measure, i.e., we count each arithmetic operation for 1. Moreover, we assume that the affine assignments at control flow edges are of *constant size,* meaning that all occurring coefficients are of constant size, and that each assignment s may contain only a constant number of variables with non-zero coefficients. Note that this assumption does not impose any restriction on the expressiveness of programs since more complicated assignments can easily be simulated by sequences of simpler ones. As a consequence, the size of the control flow graph, $n = |N| + |E|$, can be considered as a fair measure of the size of the input to the analysis algorithm.

Taking a closer look at the algorithm, we notice that each iteration of the **while**-loop consists in processing one pair (u, x) by inspecting each outgoing edge (u, s, v) of u. Thus, its time complexity is proportional to $1 + \mathsf{out}(u) \cdot C$ where $\mathsf{out}(u)$ is the out-degree of u and C is the complexity of checking whether a vector t is contained in $\mathsf{aff}(G[v])$ for some program point v. Since the sum $\sum_{u \in N} \mathsf{out}(u)$ equals the number of edges of the control flow graph, the complexity of the algorithm is proportional to

$$(k+1) \cdot \sum_{u \in N} (1 + \mathsf{out}(u) \cdot C) \;\leq\; (k+1) \cdot (n + n \cdot C) \;=\; (k+1) \cdot n \cdot (C+1)\,.$$

It remains to determine the complexity C of testing whether a vector t is contained in the affine hull of $G[v]$ for some program point v. If $G[v]$ is empty, the test will always return false. Otherwise, $G[v]$ consists of vectors $x_0, \ldots, x_m$, $0 \leq m \leq k$. Then $t \in \mathsf{aff}(G[v])$ iff the vector $t - x_0$ is contained in the *linear* vector space generated from $B = \{x_1 - x_0, \ldots, x_m - x_0\}$. This can be decided by means of Gaussian elimination – resulting in an $\mathcal{O}(k^3)$ upper bound on the complexity C of the element test.

We can do better, though. The key idea is to avoid repeated Gaussian elimination on larger and larger subsets of vectors. Instead, we maintain for v with $G[v] \neq \emptyset$ a *diagonal* basis $B' = \{x'_1, \ldots, x'_m\}$ spanning the same linear vector space as B. This means: if l_i is the index of the first non-zero component of x'_i for $i = 1, \ldots, m$, then the l_i'th component of all other basis vectors x'_j, $j \neq i$ is zero. *Reduction* of a vector $x = t - x_0$

w.r.t. the diagonal basis B' then amounts to successively subtracting suitable multiples of the vectors x'_i from x in order to make the l_i'th components of x zero. Let x' denote the vector obtained by reduction of $t - x_0$. Then $x' = \mathbf{0}$ iff $t - x_0$ is contained in L or, equivalently, $t \in \mathsf{aff}(\{x_0, \ldots, x_m\})$. If $x' \neq \mathbf{0}$, the algorithm inserts t into the set $G[v]$. Therefore, we must extend B' to a diagonal basis for $\mathbf{Span}(B \cup \{t - x_0\})$ in this case. Indeed, this is very simple: we only need to subtract suitable multiples of x' from the vectors $x'_1, \ldots, x'_m$ in order to make the l'th component of these vectors zero, where l is the index of the first non-zero component of x'. Afterwards, we add x' to the set consisting of the resulting vectors. In summary, we have replaced a full Gaussian elimination for each test $t \in \mathsf{aff}(G[u])$ by the reduction of $t - x_0$ possibly followed by the reduction of the vectors in B' by x'. Subtraction of a multiple of one x'_i from t and of a multiple of x' from x'_i uses $\mathcal{O}(k)$ operations. Since $m \leq k$, reduction of $t - x_0$ as well as reduction of B' can thus be done in time $\mathcal{O}(k^2)$. Therefore we obtain:

Theorem 2. *The affine hulls $V^\sharp[u] = \mathsf{aff}(V[u])$ of the sets of program states reaching $u, u \in N$, can be computed in time $\mathcal{O}(nk^3)$ where n is the size of the program and k the number of program variables.*

Moreover this computation performs arithmetic operations only on numbers upto bit length $\mathcal{O}(nk^2)$.

Proof. It only remains to estimate the lengths of numbers used by the algorithm. First, we observe that the algorithm performs at most $n \cdot (k+1)$ evaluations of assignment statements s. Each assignment may increase the maximal absolute value of entries of a vector x at most by a constant factor $d > 0$. Therefore, the absolute values of entries of all vectors in $G_{\mathrm{fin}}[u], u \in N$, are bounded by $d^{n \cdot (k+1)}$. Now for each set $G_{\mathrm{fin}}[u] = \{x_0, \ldots, x_m\}$ with $m > 0$, the algorithm successively applies reduction to construct a diagonal basis for the vectors $x_j - x_0, j = 1, \ldots, m$. Altogether these reduction steps perform one Gaussian elimination on all m vectors. It is well-known that Gaussian elimination introduces rational numbers whose numerators and denumerators are determinants of minors of the original coefficient matrix [12, Problem 11.5.3]. In our application, the original entries have absolute values at most $2 \cdot d^{n \cdot (k+1)}$. At most k-fold products therefore have absolute values at most $2^k \cdot d^{n \cdot (k+1)k}$. Finally, determinants are at most $(k!)$-fold sums of such products. Therefore, their absolute values are bounded by $k! \cdot 2^k \cdot d^{nk(k+1)} = 2^{\mathcal{O}(n \cdot k^2)}$ – which completes the proof. □

4 Non-deterministic Assignments

Let us now extend affine programs as defined in Section 2 with non-deterministic assignments $\mathbf{x}_i :=?$. Such assignments are necessary to model input routines returning unknown values or variable assignments whose right-hand sides are not affine expressions. The semantics of such a statement may update $\mathbf{x}_i$ in the current state with any possible value. Therefore, the transfer function $f_{\mathbf{x}_i:=?}$ is given by $f_{\mathbf{x}_i:=?}(X) = \bigsqcup\{f_{\mathbf{x}_i:=c}(X) \mid c \in \mathbb{Q}\}$. Unfortunately, this is not a finitary definition no matter whether X is an affine space or not. Fortunately, we have:

Lemma 4. $f_{x_i:=?}(\mathsf{aff}(G)) = (f_{x_i:=0}(\mathsf{aff}(G))) \sqcup (f_{x_i:=1}(\mathsf{aff}(G)))$.

Thus for affine X, the infinite union in the definition of $f_{x_i:=?}$ can be simplified to the least upper bound of two affine spaces. Lemma 4 implies that we can treat unknown assignments in flow graphs by replacing each edge (u, s, v) that is annotated with an unknown assignment, $s \equiv x_i :=?$, by the two edges $(u, x_i := 0, v)$ and $(u, x_i := 1, v)$ labeled by affine assignments prior to the analysis.

5 Affine Relations

An equation $a_0 + a_1 x_1 + \ldots + a_k x_k = 0$ is called an *affine relation.* Clearly, such a relation can be uniquely represented by its coefficient vector $a = (a_0, \ldots, a_k) \in \mathbb{Q}^{k+1}$. The affine relation a is *valid* for set $X \subseteq \mathbb{Q}^k$ iff a is satisfied by all $x \in X$, i.e.,

$$a_0 + \sum_{i=1}^{k} a_i \cdot x_i = 0 \qquad \text{for all} \quad (x_1, \ldots, x_k) \in X\,.$$

Accordingly, the relation a is *valid* at a program point u iff it is valid for the set $V[u]$ of all program states reaching u. The key objective, now of Karr's algorithm was *not* to determine (an approximation of) the collecting semantics of the program but to determine, for every program point u, the set $V^{\mathsf{T}}[u]$ of all affine relations valid at u. Here we show that this task is easy — once we have computed the affine hull $V^{\sharp}[u]$ of the sets of program states reaching u. First we recall from linear algebra that the set:

$$A(X) = \{a \in \mathbb{Q}^{k+1} \mid a \text{ is valid for } X\}$$

is a *linear* vector space. Moreover, we have for every affine relation a:

Lemma 5. *For every $X \subseteq \mathbb{Q}^k$, a is valid for X iff a is valid for* $\mathsf{aff}(X)$.

Thus, given a set $\{x_0, \ldots, x_m\}$ of vectors generating $\mathsf{aff}(X)$, we can determine the set $A(X)$ as the set of solutions of the linear equation system:

$$\mathbf{a}_0 + \mathbf{a}_1 \cdot x_{i1} + \ldots + \mathbf{a}_k \cdot x_{ik} = 0 \qquad i = 0, \ldots, m$$

if $x_i = (x_{i1}, \ldots, x_{ik})$. Determining a basis for the vector space of solutions can again be done, e.g., by Gaussian elimination. Thus, we obtain:

Theorem 3. *Assume p is an affine program of size n with k program variables. Then the sets of all relations valid at program points u can be computed in time $\mathcal{O}(nk^3)$.*

The computation requires algebraic operations only for integers of lengths bounded by $\mathcal{O}(nk^2)$.

Recall, moreover, that our algorithm not only provides us, for every program point u, with a finite set of generators of $\mathsf{aff}(V[u])$. Whenever $\mathsf{aff}(V[u]) \neq \emptyset$, it also returns a pair (x_0, B) where x_0 is an element of $V[u]$ and B is a diagonal basis of a linear vector space L such that $x \in \mathsf{aff}(V[u])$ iff $x = x_0 + x'$ for some $x' \in L$.

Lemma 6. *Assume a non-empty affine space X is given by a vector* $x_0 \in X$ *together with a basis B for the linear vector space* $L = \{x - x_0 \mid x \in X\}$. *Then the set of affine relations valid for X is the set of all solutions of the equation system:*

$$\begin{aligned} \mathbf{a}_0 + \mathbf{a}_1 \cdot x_{01} + \ldots + \mathbf{a}_k \cdot x_{0k} &= 0 \\ \mathbf{a}_1 \cdot x'_{i1} + \ldots + \mathbf{a}_k \cdot x'_{ik} &= 0 \quad \textit{for } i = 1, \ldots, m, \end{aligned}$$

where $x_0 = (x_{01}, \ldots, x_{0k})$ *and* $B = \{x'_1, \ldots, x'_m\}$ *with* $x'_i = (x'_{i1}, \ldots, x'_{ik})$.

Moreover, if the basis B is already in diagonal form, we directly can read off a basis for $A(X)$. From a practical point of view, we therefore can be even more efficient and avoid the extra post-processing round of Gaussian elimination.

6 Polynomial Relations

In [11], an interprocedural algorithm is presented which not only computes, for every program point u of an affine program, the set of valid affine relations but the set of all *polynomial relations* of degree at most d in time $\mathcal{O}(nk^{8d})$. Here we show how our version of Karr's algorithm can be extended to compute polynomial relations intraprocedurally much faster.

A *polynomial relation* is an equation $p = 0$ for a polynomial $p \in \mathbb{Q}^{\leq d}[\mathbf{X}]$, i.e., a polynomial in the unknowns $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ with coefficients from $\mathbb{Q}$ and degree bounded by d. Recall that any such polynomial can be represented as its coefficient vector $a = (a_I)_{I \in \mathcal{I}_d}$ where the index set $\mathcal{I}_d$ is given by

$$\mathcal{I}_d = \{(i_1, \ldots, i_k) \mid i_1 + \ldots + i_k \leq d\}.$$

Recall that $|\mathcal{I}_d| = \binom{k+d}{d}$. The polynomial relation $p = 0$ is *valid* for a set $X \subseteq \mathbb{Q}^k$ iff p is satisfied by all $x \in X$, i.e., $p[x/\mathbf{x}] = 0$ for all $(x_1, \ldots, x_k) \in X$. Accordingly, the relation $p = 0$ is *valid at a program point* u iff it is valid for the set $V[u]$ of all program states reaching u. Our goal is to determine, for every program point u, the set of all polynomial relations of degree up to d valid at u. Note that the set:

$$P_d(X) = \{p \in \mathbb{Q}^{\leq d}[\mathbf{X}] \mid p \text{ is valid for } X\}$$

is still a *linear* vector space of dimension less or equal $\binom{k+d}{d} = \mathcal{O}(k^d)$. This vector space, however, can no longer be determined from the *affine hull* of X.

As a simple example consider the two flow graphs in Fig. 2. In G_1, we have $V[1] = V^\sharp[1] = \{(x_1, x_2) \in \mathbb{Q}^2 \mid x_1 = x_2\}$. In G_2, we have $V[5] = \{(0,0), (1,1)\}$. Hence $V^\sharp[5] = \mathsf{aff}(V[5]) = \{(x_1, x_2) \in \mathbb{Q}^2 \mid x_1 = x_2\} = V^\sharp[1]$. It is easy to see, however, that at node 5 the polynomial relation $x_1^2 - x_2 = 0$ holds for all run-time states in contrast to node 1.

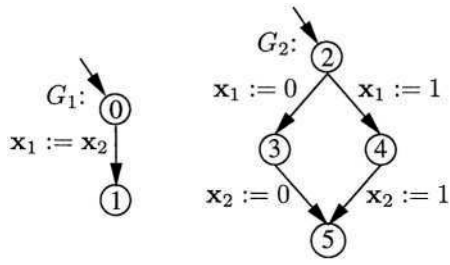


Fig. 2. Polynomial relations and affine hull.

Therefore, we define the *polynomial hull* $\mathsf{pol}_d(X)$. We do this in two steps. For a vector $x = (x_1, \ldots, x_k) \in \mathbb{Q}^k$, we define its *polynomial extension* $\eta_d(x) = (x^I)_{I \in \mathcal{I}_d}$ of degree d by: $x^{(i_1,\ldots,i_k)} = x_1^{i_1} \cdot \ldots \cdot x_k^{i_k}$, where, in particular, $x^{(0,\ldots,0)} = 1$. Thus, the polynomial extension of x has exactly $\binom{k+d}{d}$ components. Let $\eta_d(X) = \{\eta_d(x) \mid x \in X\}$. We call a vector x *polynomially implied* (up to degree d) by $X \subseteq \mathbb{Q}^k$ iff $\eta_d(x) \in \mathbf{Span}(\eta_d(X))$, i.e., iff the polynomial extension $\eta_d(x)$ is contained in the *linear hull* of the polynomial extensions of the vectors in X. The *polynomial hull* of degree d, $\mathsf{pol}_d(X)$, then consists of all vectors which are polynomially implied by X :

$$\mathsf{pol}_d(X) = \{x \in \mathbb{Q}^k \mid \eta_d(x) \in \mathbf{Span}(\eta_d(X))\}\,.$$

It is easily verified that the polynomial hull of X of degree 1 coincides with the affine hull of X. Moreover, we show for every polynomial p of degree at most d:

Lemma 7. *For every $X \subseteq \mathbb{Q}^k$, $p = 0$ is valid for X iff $p = 0$ is valid for $\mathsf{pol}_d(X)$.*

Thus, given a set $\{x_0, \ldots, x_m\}$ of vectors whose extensions $\eta_d(x_i) = (z_{iI})_{I \in \mathcal{I}_d}$ generate the linear vector space $\mathbf{Span}(\eta_d(X))$, we can determine the set $P_d(X)$ as the set of solutions of the linear equation system:

$$\sum_{I \in \mathcal{I}_d} \mathbf{a}_I \cdot z_{iI} = 0 \quad , \qquad i = 0, \ldots, m$$

Determining a basis for the vector space of solutions can again be done, e.g., by Gaussian elimination — now with $\mathcal{O}(k^d)$ variables. Thus, in order to compute the sets $P_d(V[u])$, we modify our base fixpoint algorithm to compute, instead of a finite generating set of $\mathsf{aff}(V[u])$, a finite set $G_d[u]$ generating the polynomial hull of $V[u]$. It is easily verified that pol_d is again a closure operator. Also Lemma 2 remains valid for the polynomial hull, i.e., $\mathsf{pol}_d(f_s(X)) = f_s(\mathsf{pol}_d(X))$ for all statements s and $X \subseteq \mathbb{Q}^k$. A suitable set of vectors that represents $\mathbb{Q}^k$ up to pol_d is given by the following lemma:

Lemma 8. $\mathsf{pol}_d(\mathcal{I}_d) = \mathbb{Q}^k$.

Sketch of proof. The vector space spanned by $\eta_d(\mathbb{Q}^k)$ is contained in the vector space $\mathbb{Q}^{d'}$ for $d' = \binom{d+k}{d}$. It trivially subsumes the span of $\eta_d(\mathcal{I}_d)$, i.e., $\mathbf{Span}(\eta_d(\mathcal{I}_d)) \subseteq \mathbf{Span}(\eta_d(\mathbb{Q}^k)) \subseteq \mathbb{Q}^{d'}$. We prove by induction on $k + d$ that, for all $p \in \mathbb{Q}^{\leq d}[X]$: $p(x) = 0$ for all $x \in \mathcal{I}_d$ implies $p \equiv \mathbf{0}$. From this we conclude that the set of polynomial extensions $\eta_d(x), x \in \mathcal{I}_d$, is in fact linearly independent. Therefore, their span, $\mathbf{Span}(\eta_d(\mathcal{I}_d))$, has dimension d' and thus equals $\mathbb{Q}^{d'}$. This implies $\mathsf{pol}_d(\mathcal{I}_d) = \mathbb{Q}^k$. □

By arguing similarly to Sect. 3, we obtain an algorithm that computes a finite generating set of $\mathsf{pol}_d(V[u])$ by modifying the algorithm in Fig. 1 as follows. We replace the test "$t \notin \mathsf{aff}(G[u])$" with "$t \notin \mathsf{pol}_d(G[u])$" and the initialization of $G[\mathsf{st}]$ and W with

$$G[\mathsf{st}] = \mathcal{I}_d\,; \quad W = \{(\mathsf{st}, I) \mid I \in \mathcal{I}_d\}\,;$$

In order to avoid replicated Gaussian elimination, we may maintain a diagonal basis B_d for the current vector space $\mathbf{Span}(\eta_d(G_d[u]))$. This simplifies the element test for every newly encountered $x \in \mathbb{Q}^k$ to the reduction of the extension $\eta_d(x)$ of x w.r.t. B_d possibly followed by reduction of the vectors in B_d with the reduced vector. We obtain:

Theorem 4. *Assume p is an affine program of size n with k program variables. Then the sets of all polynomial relations of degree at most d which are valid at program points u can be computed in time $\mathcal{O}(nk^{3d})$.*

The computation requires algebraic operations only for integers of lengths bounded by $\mathcal{O}(nk^{2d})$.

Similarly to [11] we can treat non-deterministic assignments $x_i :=?$ by replacing each edge $(u, x_i :=?, v)$ by $d+1$ edges $(u, x_i := l, v)$ for $l = 0, \ldots, d$. Note that the complexity of the resulting intraprocedural algorithm improves upon the complexity of our interprocedural algorithm in [11] by a factor of k^{5d}.

7 Positive Guards

In this paper we restricted attention to affine programs for which we have shown our algorithms to be precise. In Karr's paper, one can also find a non-trivial treatment of branching nodes with affine guards. The main idea is to intersect in the "true" branch the propagated affine space with the hyperplane described by the guard. While this leads to more precise results than ignoring guards totally, it is not a complete treatment of positive affine guards. Indeed, as we show next it is undecidable to decide in affine programs with positive affine guards (or even with only equality guards) whether a given affine relation holds at a program point or not. This implies that a complete algorithmic treatment of positive affine guards is impossible.

We exhibit a reduction of the Post correspondence problem (PCP) inspired by Hecht [5,8]. A Post correspondence system is a finite set of pairs $(u_1, v_1), \ldots, (u_m, v_m)$ with $u_i, v_i \in \{0,1\}^*$. The correspondence system has a solution, if and only if there is a non-empty sequence $i_1, \ldots, i_n$ such that $u_{i_1} \cdot \ldots \cdot u_{i_n} = v_{i_1} \cdot \ldots \cdot v_{i_n}$. From a given Post correspondence system we construct an affine program with an equality guard as indicated in Fig. 3. We write $|u|$ for the length of a string $u \in \{0,1\}^*$ and $\langle u \rangle_2$ for the number represented by u in standard binary number representation. The variables **x** and **y** hold binary numbers that represent strings in $\{0,1\}^*$. For each pair $(u_i, v_i) \in S$ there is an edge from program point 1 to 2 that appends the strings u_i and v_i to x and y, respectively, by appropriate affine computations. The program can loop back from program point 2 to 1 by a skip-edge. The initialization of x and y with 1 avoids a problem with leading zeros. It is not hard to see that there is an execution in which **x** = **y** is true at program point 2 if and only if the Post correspondence system admits a solution. Only in this case the path from program point 2 via 3 to 4 can be executed. We conclude

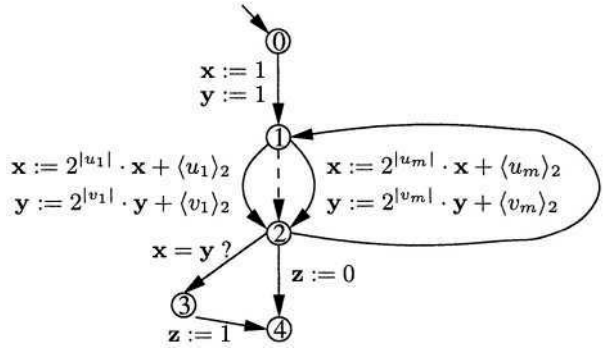


Fig. 3. A PCP reduction with affine guards.

that the affine relation $\mathbf{z} = 0$ is valid at program point 4 if and only if the given Post correspondence system S does not admit a solution.

8 Discussion and Perspective

We have presented a variant of Karr's algorithm for computing valid affine relationships among the variables in a program that has a better worst-case complexity than Karr's original formulation, avoids exponentially large numbers, and is easy to implement. We also showed how to generalize this algorithm to determine polynomial relationships.

Instrumental for our results is that we represent affine spaces by affine bases instead of kernels of affine transformations. Ironically, Karr discards a closely related representation early in his paper [6, p. 135] by remarking that the number of valid affine relationships typically will be small and hence the dimension of the affine spaces will be large, such that many basis vector but few relations are required for representation. This leads to the question whether our representation can compete with Karr's as far as memory consumption is concerned. Clearly, we need more memory for representing an affine space A of high dimension, if we store all the vectors in an affine basis $\{x_0, \ldots, x_m\}$ of A explicitly. Fortunately, instead of storing the affine basis, it suffices to store one vector, x_0, together with the diagonal basis of $\mathbf{Span}(\{x_1 - x_0, \ldots, x_m - x_0\})$ that is computed for the membership tests. The other vectors $x_1, \ldots, x_m$ need not be stored because they are neither needed for the membership tests nor for extraction of the final result. The vectors in the diagonal basis, however, can be stored sparsely such that only the non-zero components (together with their index) are stored. Then we need for representing an affine space of dimension $m, 0 \leq m \leq k$, at most $k + m + m(k - m)$ entries compared to at most $2k - 2m + m(k - m)$ in a (sparse) representation by affine relations. Surprisingly, the maximal difference is just $2k$. Insights into the practical behavior of these two representations require experiments for real-world programs which we leave for future work.

References

1. R. Bagnara, P. Hill, E. Ricci, and E. Zaffanella. Precise Widening Operators for Convex Polyhedra. In *10th Int. Static Analysis Symp. (SAS),* 337–354. LNCS 2694, Springer, 2003.
2. P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *4th POPL,* 1977.
3. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *5th POPL,* 84–97,1978.
4. S. Gulwani and G. Necula. Discovering Affine Equalities Using Random Interpretation. In *30th POPL,* 74–84, 2003.
5. M. S. Hecht. *Flow analysis of computer programs.* Elsevier North-Holland, 1977.
6. M. Karr. Affine Relationships Among Variables of a Program. *Acta Inf.,* 6:133–151,1976.
7. J. Leroux. *Algorithmique de la Vérification des Systèmes à Compteurs: Approximation et Accélération.* PhD thesis, Ecole Normale Supérieure de Cachan, 2003.
8. M. Müller-Olm and O. Rüthing. The Complexity of Constant Propagation. In *10th European Symposium on Programming (ESOP),* 190–205. LNCS 2028, Springer, 2001.
9. M. Müller-Olm and H. Seidl. Computing Polynomial Program Invariants. Submitted, 2003.

10. M. Müller-Olm and H. Seidl. Polynomial Constants are Decidable. In *9th Static Analysis Symposium (SAS),* 4–19. LNCS 2477, Springer, 2002.
11. M. Müller-Olm and H. Seidl. Precise Interprocedural Analysis through Linear Algebra. In *31st POPL,* 330–341, 2004.
12. C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

The Existence and Efficient Construction of Large Independent Sets in General Random Intersection Graphs*

S. Nikoletseas[1,2], C. Raptopoulos[1,2], and P. Spirakis[1,2]

[1] Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
{nikole, spirakis}@cti.gr, raptopox@ceid.upatras.gr
[2] University of Patras, 26500 Patras, Greece

Abstract. We investigate the existence and efficient algorithmic construction of close to optimal independent sets in random models of intersection graphs. In particular, (a) we propose *a new model* for random intersection graphs ($G_{n,m,p}$) which includes the model of [10] (the "uniform" random intersection graphs model) as an important special case. We also define an interesting variation of the model of random intersection graphs, similar in spirit to random regular graphs. (b) For this model we derive *exact formulae* for the mean and variance of the number of independent sets of size k (for any k) in the graph. (c) We then propose and analyse *three algorithms* for the efficient construction of large independent sets in this model. The first two are variations of the greedy technique while the third is a totally new algorithm. Our algorithms are analysed for the special case of uniform random intersection graphs. Our analyses show that these algorithms succeed in finding *close to optimal* independent sets for an interesting range of graph parameters.

1 Introduction

Random graphs, introduced by P. Erdös and A. Rényi, still continue to attract a huge amount of research and interest in the communities of Theoretical Computer Science, Graph Theory and Discrete Mathematics.

There exist various models of random graphs. The most famous is the $G_{n,p}$ random graph, a sample space whose points are graphs produced by randomly sampling the edges of a graph on n vertices independently, with the same probability p. Other models have also been quite a lot investigated: $G_{n,r}$ (the "random regular graphs", produced by randomly and equiprobably sampling a graph from all regular graphs of n vertices and vertex degree r) and $G_{n,M}$ (produced by randomly and equiprobably selecting an element of the class of graphs on n vertices having M edges). For an excellent survey of these models, see [1,3].

In this work we investigate, both combinatorially and algorithmically, a *new model* of random graphs. We nontrivially extend the $G_{n,m,p}$ model ("random intersection graphs") introduced by M. Karoński, E.R. Sheinerman and

* This work has been partially supported by the IST Programme of the European Union under contract numbers IST-2001-33116 (FLAGS) and 001907 (DELIS).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1029–1040, 2004.

K.B. Singer-Cohen [10] and K.B. Singer-Cohen [15]. Also, Godehardt and Jaworski [9] considered similar models. In the $G_{n,m,p}$ model, to each of the n vertices of the graph, a random subset of a universal set of m elements is assigned, by independently choosing elements with the same probability p. Two vertices u, v are then adjacent in the $G_{n,m,p}$ graph if and only if their assigned sets of elements have at least one element in common. We extend this model (which we call hereafter "uniform", because of the same probability of selecting elements) by proposing *two new models* which we define below.

Definition 1 (General random intersection graph). *Let us consider a universe $M = \{1, 2, \ldots, m\}$ of elements and a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$. If we assign independently to each vertex v_j, $j = 1, 2, \ldots, n$, a subset S_{v_j} of M by choosing each element $i \in M$ independently with probability p_i, $i = 1, 2, \ldots, m$, and put an edge between two vertices v_{j_1}, v_{j_2} if and only if $S_{v_{j_1}} \cap S_{v_{j_2}} \neq \emptyset$, then the resulting graph is an instance of the general random intersection graph $G_{n,m,\boldsymbol{p}}$, where $\boldsymbol{p} = [p_1, p_2, \ldots, p_m]$.*

Definition 2 (Regular random intersection graph). *Let us consider a universe $M = \{1, 2, \ldots, m\}$ of elements and a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$. If we assign independently to each vertex v_j, $j = 1, 2, \ldots, n$, a subset S_{v_j} consisting of λ different elements of M, randomly and uniformly chosen, and draw an edge between two vertices v_{j_1}, v_{j_2} if and only if $S_{v_{j_1}} \cap S_{v_{j_2}} \neq \emptyset$, then the resulting graph is an instance of the regular random intersection graph $G_{n,m,\lambda}$.*

The latter model may abstract λ-SAT random formulae. We note the following:

Note 1: When $p_1 = p_2 = \cdots = p_m = p$ the general random intersection graph $G_{n,m,\boldsymbol{p}}$ reduces to the $G_{n,m,p}$ as in [10] and we call it the *uniform* random intersection graph.

Note 2: When in the uniform case $mp \geq \alpha \log n$ for some constant $\alpha > 1$ then the model $G_{n,m,p}$ and the model $G_{n,m,\lambda}$ for $\lambda \in (1 \pm \epsilon)mp$, $\epsilon \in (0, 1)$, are essentially *equivalent,* i.e. they assign *almost* the same probability to edge monotone graph events. This follows from degree concentration via Chernoff bounds. Thus, all our results proved here for $G_{n,m,p}$ translate to $G_{n,m,\lambda}$.

Importance and Motivation. First of all, we note that (as proved in [11]) any graph is a random intersection graph. Thus, the $G_{n,m,p}$ model is very general. Furthermore, for some ranges of the parameters m, p ($m = n^\alpha, \alpha > 6$) the spaces $G_{n,m,p}$ and $G_{n,p}$ are equivalent (as proved by Fill, Sheinerman and Singer-Cohen [8], showing that in this range the total variation distance between the graph random variables has limit 0).

Second, random intersection graphs (and in particular our new, non- uniform model) may model real-life applications more accurately (compared to the $G_{n,p}$ case). This is because in many cases the independence of edges is not well-justified. In fact, objects that are closer (like moving hosts in mobile networks or sensors in smart dust networks) are more probable to interact with each

other. Even epidemiological phenomena (like spread of disease) tend to be more accurately captured by this "proximity-sensitive" random intersection graphs model. Other applications may include oblivious resource sharing in a distributed setting, interactions of mobile agents traversing the web etc.

Other Related Work. The question of how close $G_{n,m,p}$ and $G_{n,p}$ are for various values of m, p has been studied by Fill, Sheinerman and Singer-Cohen in [8]. Also, geometric proximity between randomly placed objects is nicely captured by the model of random geometric graphs (see e.g. [4,7,13]) and important variations (like random scaled sector graphs, [6]).

Our Contribution

1. We first introduce *two new models,* as explained above: the $G_{n,m,\boldsymbol{p}}$ model and the $G_{n,m,\lambda}$ model. We feel that our models are important, in the sense that $G_{n,m,\boldsymbol{p}}$ is a very general model and $G_{n,m,\lambda}$ is very focused (so it is particularly precise in abstracting several phenomena).
2. We show interesting *relations between the models* we introduce, i.e. we prove that when $mp = \alpha \log n$ then $G_{n,m,\boldsymbol{p}}$ is almost equivalent to $G_{n,m,\lambda}$ (see Note 2 above).
3. Under these models we study the well known and fundamental problem of *finding a maximum independent set of vertices.* In particular, in the most general $G_{n,m,\boldsymbol{p}}$ model we estimate *exactly* the mean and the variance of the number of independent sets of size k. To get exact formulas for the variance, we introduce and use a *"vertex contraction technique"* to evaluate the covariance of random indicator variables of non-disjoint sets of vertices. This technique, we believe, has its own combinatorial interest and may be used in investigating other combinatorial problems as well.
4. Finally, we provide and analyse *three efficient algorithms* for finding large independent sets:
   - Algorithm I is the classic greedy algorithm (for example see [2]) for maximum independent set approximation.
   - Algorithm II is a variation of the above where a random new vertex is tried each time instead of that of current minimum degree.
   - Algorithm III is a totally new algorithm (that we propose) pertinent to the model $G_{n,m,\boldsymbol{p}}$.

   For clarity, all our algorithms are analysed for the uniform random intersection graphs model.

   Our algorithms are analysed for the interesting case where $mp \geq \alpha \log n$, (for some constant $\alpha > 1$), in which no isolated vertices exist in $G_{n,m,\boldsymbol{p}}$ and also the results translate to $G_{n,m,\lambda}$ (see Note 2).

   To our knowledge, this is the first time that algorithms for random intersection graphs are proposed and analysed. Our analyses show that in many interesting ranges of p, m, the sizes of the independent sets obtained by the algorithms are quite large.

2 The Size of Independent Sets – Exact Formulae

Theorem 1. *Let $X^{(k)}$ denote the number of independent sets of size k in a random intersection graph $G(n,m,\boldsymbol{p})$, where $\boldsymbol{p}=[p_1,p_2,\ldots,p_m]$. Then*

$$E\left[X^{(k)}\right]=\binom{n}{k}\prod_{i=1}^{m}\left((1-p_i)^k+kp_i(1-p_i)^{k-1}\right).$$

Proof. Remark that for a set of k vertices to be an independent set, each vertex in it must choose each label at most once (see [12]). □

Theorem 2. *Let $X^{(k)}$ denote the number of independent sets of size k in a random intersection graph $G(n,m,\boldsymbol{p})$, where $\boldsymbol{p}=[p_1,p_2,\ldots,p_m]$. Then*

$$Var\left(X^{(k)}\right)=\sum_{s=1}^{k}\binom{n}{2k-s}\binom{2k-s}{s}\left(\gamma(k,s)E\left[X^{(k)}\right]-E^2\left[X^{(k)}\right]\right)$$

where $E\left[X^{(k)}\right]$ is the mean number of independent sets of size k and

$$\gamma(k,s)=\prod_{i=1}^{m}\left((1-p_i)^{k-s}+(k-s)p_i(1-p_i)^{k-s-1}\left(1-\frac{sp_i}{1+(k-1)p_i}\right)\right).$$

Proof. Let V' be any set of k vertices and let

$$X_{V'}=\begin{cases}1 \text{ if } V' \text{ is an independent set}\\ 0 \text{ otherwise.}\end{cases}$$

Clearly, $X^{(k)}=\sum_{V',|V'|=k}X_{V'}$ and for V_1',V_2' any sets of k vertices,

$$\begin{aligned}Var\left(X^{(k)}\right)&=\sum_{V_1',V_2',|V_1'|=|V_2'|=k}Cov(X_{V_1'},X_{V_2'})\\&=\sum_{s=1}^{k}\sum_{\substack{V_1',V_2',|V_1'|=|V_2'|=k\\|V_1'\cap V_2'|=s}}P\{X_{V_1'}X_{V_2'}=1\}-E^2\left[X^{(k)}\right].\end{aligned}\tag{1}$$

Since

$$P\{X_{V_1'}X_{V_2'}=1\}=P\{X_{V_1'}=1|X_{V_2'}=1\}E\left[X^{(k)}\right]\tag{2}$$

the problem of computing the variance of $X^{(k)}$ is reduced to computing the conditional probability $P\{X_{V_1'}=1|X_{V_2'}=1\}$, i.e. the probability that V_1' is an independent set given that V_2' is an independent set, where V_1',V_2' are any two sets of k vertices that have s vertices in common. In order to compute $P\{X_{V_1'}=1|X_{V_2'}=1\}$, we will try to merge several vertices into one supervertex and study its probabilistic behaviour.

Towards this goal, let us fix an element i of $M = \{1, 2, \ldots, m\}$ and let us consider two (super)vertices v_1, v_2 of the $G(n, m, p)$ graph that choose element i independently with probability $p_i^{(1)}$ and $p_i^{(2)}$ respectively. Let also S_{v_1}, S_{v_2} denote the sets of elements of M assigned to v_1 and v_2 respectively. Then,

$$P\{i \in S_{v_1} | \nexists(v_1, v_2)\} = P\{i \in S_{v_1}, i \notin S_{v_2} | \nexists(v_1, v_2)\}$$
$$= \frac{P\{i \in S_{v_1}, i \notin S_{v_2}, \nexists(v_1, v_2)\}}{P\{\nexists(v_1, v_2)\}} = \frac{p_i^{(1)}(1 - p_i^{(2)})}{1 - p_i^{(1)} p_i^{(2)}} \quad (3)$$

where (v_1, v_2) is an edge between v_1 and v_2. From this we get:

- Conditional on the fact that (v_1, v_2) does not exist, the probabilistic behaviour of vertex v_1 is identical to thet of a single vertex that chooses element i of M independently with probability $\frac{p_i^{(1)}(1-p_i^{(2)})}{1-p_i^{(1)}p_i^{(2)}}$.
- Conditional on the fact that (v_1, v_2) does not exist, the probabilistic behaviour of v_1 and v_2 considered as a unit is identical to that of a single vertex that chooses element i of M independently with probability

$$P\{i \in S_{v_1} \cup S_{v_2} | \nexists(v_1, v_2)\} = P\{i \in S_{v_1} | \nexists(v_1, v_2)\} + P\{i \in S_{v_2} | \nexists(v_1, v_2)\}$$
$$= \frac{p_i^{(1)} + p_i^{(2)} - 2p_i^{(1)}p_i^{(2)}}{1 - p_i^{(1)} p_i^{(2)}} \quad (4)$$

where i is a fixed element of M. The first of the above equations follows from the observation that if there is no edge between v_1 and v_2 then the sets S_{v_1} and S_{v_2} are disjoint, meaning that element i cannot belong to both of them. The second equation follows from symmetry.

Let us now consider merging one by one the vertices of the $G(n, m, p)$ graph into one supervertex. Let w_j denote a supervertex of j simple vertices that form an independent set. It is obvious that the probabilistic behaviour of w_j is irrelevant to how partial mergings are made. Moreover, if w_{j_1}, w_{j_2} are two supervertices representing two disjoint sets of simple vertices, we say that an edge (w_{j_1}, w_{j_2}) exists iff any edge connecting a simple vertex in w_{j_1} and a simple vertex in w_{j_2} exists. Thus, the event $\{\nexists(w_{j_1}, w_{j_2})\}$ is equivalent to the event {the vertices in w_{j_1} together with those in w_{j_2} form an independent set}.

Using equation (4) one can show that $P\{i \in S_{w_2}\} = \frac{2p_i}{1+p_i}$, $P\{i \in S_{w_3}\} = \frac{3p_i}{1+2p_i}$ and by induction

$$P\{i \in S_{w_j}\} = \frac{jp_i}{1 + (j-1)p_i} \quad (5)$$

where i is a fixed element of M and S_{w_j} is the union of all the sets of elements of M assigned to each simple vertex in w_j. More formally,

$$S_{w_j} = \bigcup_{v \in w_j} S_v$$

where v is a simple vertex and S_v is the set of elements of M assigned to v. Because of the definition of w_j, the subsets S_v in the above union are disjoint.

Thus, let V_1' be any set of k (simple) vertices and let V_2' be an independent set of k vertices that has s vertices in common with V_1'. Since there is no edge between any vertices in V_2', we can treat the $k-s$ vertices of V_2' not belonging to V_1' and the s vertices belonging to both V_1' and V_2' as two seperate supervertices w_{k-s} and w_s respectively that do not communicate by an edge. Hence, by equations (3), (4) and (5), the probabilistic behaviour of w_s is identical to that of a single vertex w_s' that chooses the elements of M independently with probabilities $\{p_i^{(w_s')}, i = 1, \ldots, m\}$ respectively, where

$$p_i^{(w_s')} = \frac{p_i^{(w_s)}(1 - p_i^{(w_{k-s})})}{1 - p_i^{(w_s)} p_i^{(w_{k-s})}} = \frac{sp_i}{1 + (k-1)p_i}. \tag{6}$$

Let now V'' be a set of $k-s$ simple vertices and a vertex identical to w_s'. Then, for a fixed element i of M, each of the $k-s$ simple vertices chooses i independently with probability p_i, while the supervertex w_s' chooses i independently with probability $p_i^{(w_s')}$. Similarly to Theorem 1 we get

$$P\{X_{V_1'} = 1 | X_{V_2'} = 1\} = P\{V'' \text{ is an independent set}\} \stackrel{def}{=} \gamma(k, s).$$

Hence, by equations (1) and (2), we get the result. □

3 Finding Large Independent Sets in $G_{n,m,p}$

We start from the classic greedy approach, i.e. starting from the empty set we introduce (into the independent set under construction) each time the *minimum degree* vertex in the current graph and then delete it and its neighbours from the graph (Algorithm I).

The Expected Size of the Independent Set constructed. As can be seen in e.g. [2], if $r = |V'|$ eventually, and $\delta = \frac{|E|}{n}$, i.e. δ is the *density* of G),

$$r(2\delta + 1) \geq n. \tag{7}$$

This holds for any input graph G (for a proof see [12]). Taking expectations we get $E[r(2\delta + 1)] \geq n$, where the expectation is taken over all instances of the distribution $G_{n,m,p}$ (notice that both r, δ are random variables).

The property "$\exists$ independent set of size r" is monotone decreasing on the number of edges, while the property "the density of G is δ" is monotone increasing. Hence, by the FKG inequality (see [1]) we get $E[r\delta] \leq E[r]\,E[\delta]$ or equivalently $E[r(2\delta + 1)] = 2E[r\delta] + E[r] \leq 2E[r]\,E[\delta] + E[r] = E[r]\,(2E[\delta] + 1)$.

Using the fact that $E[r(2\delta + 1)] \geq n$, we conclude that

$$E[r] \geq \frac{n}{2E[\delta] + 1} = \frac{n}{2\frac{E(|E|)}{n} + 1}. \tag{8}$$

Easily, $E(|E|) = \binom{n}{2}(1-(1-p^2)^m)$. Hence, we get the following:

Lemma 1. *The expected cardinality of the independent set constructed by Algorithm I is at least*

$$\frac{n^2}{2\binom{n}{2}(1-(1-p^2)^m)+n} = \frac{n^2}{2E(|E|)+n}.$$

The next result is easily derived from Lemma 1.

Corollary 1 (Sparse $G_{n,m,p}$ theorem). *For p such that $E(|E|) = \Theta(n)$, the expected size of the independent set provided by Algorithm I is $\Theta(n)$.*

Remark: The above analysis carries out in an almost similar way to the general random intersection graphs model. For example, if $p = \frac{\alpha}{\sqrt{nm}}$, where $0 < \alpha < 1$, then $E[r] \geq \frac{n}{\alpha}$.

A Concentration Result for Sparce Graphs. We are interested in intersection graphs $G_{n,m,p}$ for p satisfying

$$\frac{\omega(n)}{n\sqrt{m}} \leq p \leq \sqrt{\frac{2\log n - \omega(n)}{m}}$$

for the smallest possible function $\omega(n) \to \infty$, as $n \to \infty$. This is the range for nontrivial graphs (see [8]).

We consider the case $p < \sqrt{\frac{1}{8nm}}$ which is in the range of nontrivial graphs. In the sequel we assume that $p(n) = \frac{c(n)}{m}$ where $c(n) \to \infty$, as $n \to \infty$. For example, since $c(n) = mp$, if we take p in the range of nontrivial graphs, then

$$\frac{\sqrt{m}}{n}\omega(n) \leq c(n) \leq \sqrt{2m\log n - \omega(n)m}. \tag{9}$$

A choice of $c(n)$ satisfying this is $c(n) = \alpha \log n$, where $\alpha > 1$, since, from (9), $\omega(n)$ must be less than $2\log n$.

Notice that our assumption $p < \sqrt{\frac{1}{8nm}}$ implies $c(n) = mp < \sqrt{\frac{m}{8n}}$ which is satisfied by $c(n) = \alpha \log n$, i.e. for $m \geq 8\alpha^2 n \log^2 n$.

Consider a vertex v and let S_v be the set of elements assigned to it. Using Chernoff bounds (see e.g. [5]) and Boole's inequality, for $mp = \alpha \log n$ and $\epsilon \in (0,1)$, we get

$$P\{\exists v : ||S_v| - mp| \geq \epsilon mp\} \leq \sum_{v \in V} P\{||S_v| - mp| \geq \epsilon mp\} = n^{-\frac{\alpha\epsilon^2}{2}+1}.$$

If we choose the parameter α so that $-\frac{\alpha\epsilon^2}{2} + 1 > 2$, then all vertices have each a number of chosen elements "around" mp with probability at least $1 - \frac{1}{n^2}$.

Let us condition $G_{n,m,p}$ on this event. Because of symmetry, the elements chosen by each vertex are otherwise uniform in $\{1, 2, \ldots, m\}$.

Consider a variation of Algorithm I (Algorithm II) where we select greedily each time a *random* vertex to insert from the random graph.

The difference between Algorithm I and Algorithm II is that in the latter we do not use the (useful) heuristic urging us to choose at each iteration the vertex with the current minimum degree. It is clear that if r_1, r_2 are the sizes of the independent sets provided by Algorithm I and Algorithm II respectively, then $P\{r_1 \geq x\} \geq P\{r_2 \geq x\}$ for all $x > 0$, i.e. the random variable r_1 stochastically dominates r_2.

We now concentrate on estimation of r_2 with high probability. Clearly, after i successful node insertions into V' the following are true:

- $|S(V')| \in (1 \pm \epsilon) imp = (1 \pm \epsilon) ic(n)$.
- The next tried node u is rejected with probability

$$P_{rej} = 1 - \left(1 - \frac{|S(V')|}{m}\right)^{|S_u|}$$

since each element $l \in S_u$ belongs also in $S(V')$ with probability $\frac{|S(V')|}{m}$, which in turn follows from independence and uniformity.

Combining these two observations we conclude that the probability that a vertex u is rejected after i successful insertions is

$$P_{rej} \leq \frac{|S(V')||S_u|}{m}$$

which is at most $\frac{(1+\epsilon)^2 ic^2(n)}{m}$, for any $\epsilon \in (0,1)$, provided that $\frac{ic^2(n)}{m} \to 0$, i.e. provided that $i = o\left(\frac{m}{c^2(n)}\right)$. (Note also that, by the Bernoulli inequality, we have

$$P_{acc} = 1 - P_{rej} = \left(1 - \frac{|S(V')|}{m}\right)^{|S_u|} \geq 1 - \frac{|S(V')||S_u|}{m}$$

and when $\frac{|S(V')||S_u|}{m} \to 0$, then $P_{acc} \to 1$.)

Since $i \leq n$ and $1 + \epsilon < 2$, for any $\epsilon \in (0,1)$, we have that $P_{rej} < \frac{4nc^2(n)}{m}$. Moreover, since $mp < \sqrt{\frac{m}{8n}}$ by assumption, we get $P_{rej} < \frac{1}{2}$. Thus, the number r_2 of nodes that are successfully inserted into V' is at least the number of successes of the Bernoulli $\mathcal{B}(n, \frac{1}{2})$. From Chernoff bounds then, for any $\beta > 0$ we have $r_2 \geq (1-\beta)\frac{n}{2}$ with probability at least $1 - \exp\{-\frac{\beta^2}{2}\frac{n}{2}\}$.

Since r_1 stochastically dominates r_2 we eventually have (set $\beta = \frac{1}{2}$), by combining events, the following

Theorem 3. *Consider a random intersection graph $G_{n,m,p}$ with $p < \sqrt{\frac{1}{8nm}}$ and $mp = \alpha \log n$, ($\alpha > 1$ a constant). Then Algorithm I constructs an independent set of size at least $\frac{n}{4}$ with probability at least $1 - \frac{1}{2n^2}$.*

3.1 Algorithm III

Algorithm III:

Input: A random intersection graph $G_{n,m,p}$.
Output: An independent set of verices A_m.

1. set $A_0 := V$; set $L := M$;
2. **for** $i = 1$ **to** m **do**
3. **begin**
4. select a random label $l_i \in L$; set $L := L - \{l_i\}$;
5. set $D_i := \{v \in V : l_i \in S_v\}$;
6. **if** $(|D_i| \geq 1)$ **then** select a random vertex $u \in D_i$ and set $D_i := D_i - \{u\}$;
7. set $A_i := A_{i-1} - D_i$;
8. **end**
9. **output** A_m;

Theorem 4 (Correctness). *Algorithm III correctly finds an independent set of vertices.*

Proof. See full paper [12]. □

Theorem 5 (Efficiency). *For the case $mp = \alpha \log n$ for some constant $\alpha > 1$ and $m \geq n$ with high probability we have for some constant $\beta > 0$:*

1. *If $np \to \infty$ then $|A_m| \geq (1 - \beta)\frac{n}{\log n}$.*
2. *If $np \to b$ where $b > 0$ is a constant then $|A_m| \geq (1 - \beta)n(1 - e^{-b})$.*
3. *If $np \to 0$ then $|A_m| \geq (1 - \beta)n$.*

Proof. Let us define the indicator random variables

$$X_v^{(i)} = \begin{cases} 1 \text{ if vertex } v \text{ of } A_{i-1} \text{ does not contain } l_i \\ 0 \text{ otherwise} \end{cases}$$

and

$$I_{D_i} = \begin{cases} 1 \text{ if } |D_i| \geq 1 \\ 0 \text{ otherwise.} \end{cases}$$

Clearly, $|A_i| = \sum_{v \in A_{i-1}} X_v^{(i)} + I_{D_i}$, for $i = 1, 2, \ldots, m$.

Since the elements of M are chosen independently, the variables $X_v^{(i)}$ are independent of the set A_{i-1}. Hence, by Wald's equation (for the expectation of the sum of a random number of independent variables, see [14]) and linearity of expectation,

$$E(|A_i|) = E(|A_{i-1}|)(1 - p) + P\{|D_i| \geq 1\} \quad , \text{for } i = 1, 2, \ldots, m.$$

Using the above equation we can prove inductively that

$$E(|A_m|) = n(1-p)^m + \sum_{i=1}^{m}(1-p)^{m-i}P\{|D_i| \geq 1\}. \tag{10}$$

(*Note:* By a similar proof one can verify that the term $n(1-p)^m$ is the mean number of isolated vertices in the graph. By choosing $mp \geq \alpha logn$ for some constant $\alpha > 1$ the mean number of isolated vertices tends to 0.)

Now let $L_i = \{v \in V : l_i \in S_v\}$, i.e. L_i is the set of vertices having l_i (before examining them for other elements of M). Then

$$P\{|D_i| \geq 1\} = 1 - P\{|D_i| = 0\} = 1 - (P\{v \notin D_i\})^n \tag{11}$$

where v is any specific vertex. The second equality follows from symmetry. But

$$\begin{aligned} P\{v \notin D_i\} &= P\{v \notin L_i \cap v \notin D_i\} + P\{v \in L_i \cap v \notin D_i\} \\ &= 1 - p + P\{v \in L_i \cap \{v \in L_1 \cup L_2 \cup \cdots \cup L_{i-1}\}\}. \end{aligned}$$

Since the choices of the elements of M are independent, the events $\{v \in L_i\}$ and $\{v \in L_1 \cup L_2 \cup \cdots \cup L_{i-1}\}$ are also independent. Hence

$$\begin{aligned} P\{v \notin D_i\} &= 1 - p + P\{v \in L_i\}P\{v \in L_1 \cup L_2 \cup \cdots \cup L_{i-1}\} \\ &= 1 - p + p\left(1 - (1-p)^{i-1}\right) = 1 - p(1-p)^{i-1}. \end{aligned}$$

By (11), $P\{|D_i| \geq 1\} = 1 - \left(1 - p(1-p)^{i-1}\right)^n$. By (10),

$$E(|A_m|) = n(1-p)^m + \frac{1}{p}\left(1 - (1-p)^m\right) - \sum_{i=1}^{m}(1-p)^{m-i}\left(1 - p(1-p)^{i-1}\right)^n.$$

In the interesting case where $mp \geq \alpha \log n$ for some constant $\alpha > 1$ and $m \geq n$ (implying that $p \to 0$) we get

$$\begin{aligned} E(|A_m|) &\sim n(1-p)^m + \frac{1}{p}\left(1 - (1-p)^m\right) - \sum_{i=1}^{m}(1-p)^{m-i}(1-p)^n \\ &\sim \frac{1}{p}\left(1 - (1-p)^n\right). \end{aligned}$$

We now distinguish three cases, covering all possible values of np.

1. If $np \to \infty$ then $E(|A_m|) \sim \frac{1}{p}$. The largest p to have $np \to \infty$, $mp \geq \alpha \log n$ and $m \geq n$ is $p = \frac{\log n}{n}$. So, we conclude that $E(|A_m|) = \Omega(\frac{n}{\log n})$.
2. If $np \to b$ where $b > 0$ is a constant then $E(|A_m|) \sim \frac{n}{b}(1 - e^{-b}) = \Theta(n)$.
3. If $np \to 0$ then $E(|A_m|) \sim \frac{1}{p}(1 - 1 + np) = \Theta(n)$.

The proof ends with the observation that since $E(|A_m|) \to \infty$ in all tree cases, then one can use Chernoff bounds to prove that $|A_m| \geq (1-\beta)E(|A_m|)$ for any constant $\beta > 0$ with very high probability.

□

4 Conclusions and Further Work

We proposed a very general, yet tractable, model for random intersection graphs. We believe that it can be useful in many technological applications. We also did the first step in analysing algorithms for such graphs, and for the problem of construction of large independent sets of vertices. The finding of efficient algorithms for other interesting graph objects (e.g. long paths, giant components, dominating sets etc) is a subject of our future work.

Acknowledgement. We wish to thank M. Karoński and K.B. Singer-Cohen for informing us about their work in random intersection graphs and providing useful material.

References

1. N. Alon and J.H. Spencer, *"The Probabilistic Method",* Second Edition, John Wiley & Sons, Inc, 2000.
2. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela and M. Protasi, *"Complexity and Approximation",* Springer-Verlag Berlin Heidelberg, 1999.
3. B. Bollobás, *"Random Graphs",* Second Edition, Cambridge University Press, 2001.
4. J. Díaz, M.D. Penrose, J.Petit and M. Serna, *"Approximating Layout Problems on Random Geometric Graphs",* Journal of Algorithms, 39:78-116, 2001.
5. J. Díaz, J.Petit and M. Serna, Chapter titled *"A Guide to Concentration Bounds",* in the "Handbook of Randomized Computing - Volumes I & II (Combinatorial Optimization 9)", pp.457-507, Kluwer Academic Publishers, Volume I, 2001.
6. J. Díaz, J.Petit and M. Serna, *"A Random Graph Model for Optical Networks of Sensors",* in the 1st International Workshop on Efficient and Experimental Algorithms, (WEA), 2003. Also in the IEEE Transactions on Mobile Computing Journal, 2(3):186-196, 2003.
7. J. Díaz, J.Petit and M. Serna, *"Random Geometric Problems on* $[0,1]^2$*",* in J. Rolim, M. Luby and M. Serna, editors, Randomization and Approximation Techniques in Computer Science, volume 1518 of Lecture Notes in Computer Science, pages 294-306, Springer Verlag, Berlin, 1998.
8. J.A. Fill, E.R. Sheinerman and K.B Singer-Cohen, *"Random Intersection Graphs when* $m = \omega(n)$*: An Equivalence Theorem Relating the Evolution of the* $G(n,m,p)$ *and* $G(n,p)$ *models",* `http://citeseer.nj.nec.com/fill98random.html`
9. E. Godehardt and J. Jaworski, *"Two models of Random Intersection Graphs for Classification",* Studies in Classification, Data Analysis and Knowledge Organisation, Opitz O., Schwaiger M, (Eds), Springer Verlag, Berlin, Heidelberg, New York (2002), 67-82.
10. M. Karoński, E.R. Scheinerman and K.B. Singer- Cohen, *"On Random Intersection Graphs: The Subgraph Problem",* Combinatorics, Probability and Computing journal (1999) 8, 131-159.
11. E. Marczewski, *"Sur deux propriétés des classes d' ensembles",* Fund. Math. 33, 303- 307 (1945).

12. S. Nikoletseas, C. Raptopoulos and P. Spirakis, *"The Existence and Efficient Construction of Large Independent Sets in General Random Intersection Graphs",* http://www.cti.gr/RD1/nikole/english/psfiles/paper.ps.
13. M. Penrose, *"Random Geometric Graphs",* Oxford Studies in Probability, 2003.
14. S.M. Ross, *"Stochastic Processes",* Second Edition, John Wiley &: Sons, Inc., 1996.
15. K.B. Singer-Cohen, *"Random Intersection Graphs",* PhD thesis, John Hopkins University, 1995.

Efficient Consistency Proofs for Generalized Queries on a Committed Database*

Rafail Ostrovsky[1], Charles Rackoff[2], and Adam Smith[3]

[1] UCLA Dept. of Computer Science, Los Angeles, CA, USA.
rafail@cs.ucla.edu
[2] University of Toronto, Toronto, Ontario, Canada.
rackoff@cs.toronto.edu
[3] MIT Computer Science and AI Lab, Cambridge, MA, USA.
asmith@csail.mit.edu

Abstract. A *consistent query protocol* (CQP) allows a database owner to publish a very short string c which *commits* her and everybody else to a particular database D, so that any copy of the database can later be used to answer queries and give short proofs that the answers are consistent with the commitment c. Here *commits* means that there is at most one database D that anybody can find (in polynomial time) which is consistent with c. (Unlike in some previous work, this strong guarantee holds even for owners who try to cheat while creating c.) Efficient CQPS for membership and one-dimensional range queries are known [4, 11,16]: given a query pair $a, b \in \mathbb{R}$, the server answers with all the keys in the database which lie in the interval $[a, b]$ and a proof that the answer is correct.

This paper explores CQPS for more general types of databases. We put forward a general technique for constructing CQPS for any type of query, assuming the existence of a data structure/algorithm with certain inherent robustness properties that we define (called a *data robust algorithm*). We illustrate our technique by constructing an efficient protocol for *orthogonal range queries,* where the database keys are points in $\mathbb{R}^d$ and a query asks for all keys in a rectangle $[a_1, b_1] \times \ldots \times [a_d, b_d]$. Our data-robust algorithm is within a $O(\log N)$ factor of the best known standard data structure (a range tree, due to Bentley [2]).

We modify our protocol so that it is also *private,* that is, the proofs leak no information about the database beyond the query answers. We show a generic modification to ensure privacy based on zero-knowledge proofs, and also give a new, more efficient protocol tailored to hash trees.

1 Introduction

Informally, a *consistent query* protocol (CQP) allows a database owner to publish a short string c which *commits* her to a particular database D, so that she can later answer queries and give short proofs that her answers are consistent with D. Here *commits* means that she cannot change her mind about D — there is at most one database she can find

* Preliminary work done during the summer of 2000 when all authors were visiting/working at Telcordia Technologies. Preliminary version appeared as MIT LCS Technical Report TR-887, Feb. 2003 [20]. Work of the first author at UCLA is partially supported by a gift from Teradata.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1041–1053, 2004.

(in polynomial time) which is consistent with c (e.g. c could be a secure hash of D). Similarly, she can only find valid proofs for query answers which are consistent with D. The challenge is to make both the commitment and the proofs of consistency as short and simple as possible.

One may also require *privacy* – that is, the proofs of consistency should not leak any information on the database beyond the query answers. Privacy is important, for example, in settings in which query answers are sold individually, or in which the database contains personal data. Adding this requirement to a CQP brings it much closer to the traditional cryptographic notion of a commitment scheme.

Below, we discuss relevant related work and then describe our results in detail.

Related Work. We discuss the related work in the context of cryptographic commitment protocols. These have been studied extensively, and part of our contribution is to tie them in to an algorithmic point of view. A commitment protocol allows Alice to put a value a in a virtual envelope and hand it to Bob. Bob learns nothing about the value *(hiding),* but Alice can later open the envelope, without being able to reveal a different value a' *(binding).*

Commitment Schemes for Large Datasets. The notion of commitment has been generalized considerably to allow revealing only partial information about the committed data, using very little communication. Merkle [17] proposed the following protocol for committing to a list of N values $a_1, ..., a_N$: Pick a collision-resistant hash-function[1] H (say from $2k$ bits to k bits), pair up inputs $(a_1, a_2), \ldots, (a_{N-1}, a_N)$ and apply H to each pair. Now, pair up the resulting hash values and repeat this process, constructing a binary tree of hash values, until you get to a single root of length k. If the root of the tree is published (or sent to Bob by Alice), the entire collection of values is now committed to, though not necessarily hidden—we discuss hiding further below. To reveal any particular value a_i, Alice can reveal a path from the root to a_i together with all the siblings along the path. This requires only k log N bits. This idea has many cryptographic applications, including efficient signature schemes [17,5], efficient zero-knowledge arguments [10,1] and computationally sound proofs [15].

Recently Buldas, Laud and Lipmaa [3], Kilian [11] and Micali and Rabin [16] independently generalized this idea to allow committing to a *set* of values. The server produces a short commitment to her set of (key, value) pairs which is made public. When a client makes a *membership query* (i.e. "do you have an entry with key x?"), the server returns the answer along with a short proof of consistency. (We call a scheme for this task a CQP for membership queries.) A very similar data structure (again, a Merkle tree) also allows one to also answer one-dimensional *range queries,* e.g. "What keys lie between x and y?" [4,11,16]. Merkle trees were subsequently modified to allow efficient updates by changing the structure to resemble a skip list [12]. Our work generalizes these ideas to more complex queries and data structures, and provides rigorous proofs of security.

[1] A hash function family $H_\kappa(\cdot)$ is *collision-resistant* if no poly-time algorithm given κ can find a pair of inputs that map to the same output for a randomly chosen key κ (see Section 2).

Protocols with a Trusted Committer. There is substantial work on *authenticated data structures* [18], which allow one to guarantee the consistency of many replicated copies of a database. That work tackles a different problem from ours, since it assumes that the commitment phase is always performed honestly. As with ordinary commitments, assuming a trusted committer allows for simpler, more efficient solutions than are known in our (general) setting; the generic construction in this paper can be viewed as a more robust version of the generic constructions of authenticated data structures [18,13,8]. For discussions of the dangers of assuming a trusted committer, see [3,12].

Privacy for Committed Databases. Micali, Rabin and Kilian [14] show how to prove consistency of answers to membership queries while also hiding information about unanswered queries. They require that consistency proofs leak nothing about the database except the query answer—not even the size of the database. (They call the primitive a *zero-knowledge set.*) They give an efficient protocol based on the DDH assumption, with proof length $O(k \log M)$ where M is an upper bound on the set size (k is the output length of the hash function). Our techniques achieve this result with $poly(k)$ communication under more general assumptions and for more general types of queries. Subsequent to our work, [9] achieved the results of [14] based on general assumptions.

Our Contributions. This paper considers CQPS for types of queries beyond simple membership and range queries. We give a general framework for designing such protocols based on query algorithms with a certain robustness property, and illustrate our paradigm for *orthogonal range queries,* constructing protocols with an $O(k \log N)$ overhead over the fastest known standard query alogrithms. We also show how to make the protocols *private* without too much loss of efficiency.

A general paradigm for CQPS. We introduce *data-robust algorithms* (DRAS). These are search algorithms (paired with data structures) which are robust against corruptions of the data by an unbounded, *malicious* adversary: for any input—essentially, an arbitrary string— the algorithm will answer all queries consistently with one (valid) database.

Assuming the existence of collision-resistant hash functions, any DRA which accesses memory via pointers can be transformed into a consistent query protocol whose (non-interactive) consistency proofs have length at most $O(kT)$, where k is the output size of the hash function and T is the running time of the DRA.

CQP *for Orthogonal Range Queries.* We present a consistent query protocol scheme that allows efficient orthogonal range queries in d dimensions. That is, the database consists of tuples $(\mathsf{key}_1, \ldots, \mathsf{key}_d, \mathsf{value})$, a query consists of d intervals $[a_1, b_1], \ldots, [a_d, b_d]$, and an answer is the set of all database elements whose keys lie inside the corresponding hypercube. The server not only proves that it has provided all the points in the database which match the query, but also that no others exist.

Our consistency proofs have size $O(k(m+1)\log^d N)$, where N is the database size, k is the security parameter, and m is the number of keys in the database satisfying the query (the computation required is $O((m+1)\log^d N)$ hash evaluations). For range queries on a single key, our construction reduces essentially to that of [4,16,11].

Our protocol is obtained by first constructing a DRA based on range trees, a classic data structure due to Bentley [2]. Existing algorithms (in particular, the authenticated data structures of [13]) do not suffice, as inconsistencies in the data structure can lead to inconsistent query answers. Instead, we show how local checks can be used to ensure that all queries are answered consistently with a single database. For d-dimensional queries, the query time is $O((m+1)\log^d N)$, where m is the number of hits for the query and N is the number of keys in the database. This is within log N of the best known (non-robust) data structure.

Achieving Privacy Efficiently. Consistent query protocols will, in general, leak information about the database beyond the answer to the query. It is possible to add privacy to any CQP using generic techniques: one can replace the proof of consistency π with a zero-knowledge proof of knowledge of π. Surprisingly, this leads to schemes with better asymptotic communication complexity, namely $O(poly(k))$. This generic transformation can hide the size of the database, as in [14].

However, the use of NP reductions and probabilistically checkable proofs in generic constructions means that the advantages only appear for extremely large datasets. We give a simpler zero-knowledge protocol tailored to Merkle trees, which does not hide the size of the database. The crux of that protocol is to avoid NP reductions when proving zero-knowledge statements about values of the hash function, and so the result is called an *explicit-hash Merkle tree.* As a sample application, we show how this protocol can be used to add privacy to one-dimensional range trees.

Organization. Section 2 formally defines CQPS. Section 3 explains data-robust algorithms, and the transformation from DRAS to CQPS. Section 4 gives our DRA for orthogonal range queries. Section 5 discusses techniques for making CQPS private. Due to lack of space, all proofs are deferred to the full version.

2 Definitions

A function $f(k)$ is *negligible* in a parameter k if $f(k) \in O(\frac{1}{k^c})$ for all integers $c > 0$. Assigning the (possibly randomized) output of algorithm A on input x to variable y is denoted by $y \leftarrow A(x)$. An important component is collision-resistant hash functions (CRHF). This is a family of length-reducing functions (say from $3k$ bits to k bits) such that given a randomly chosen function h from the family, it is computationally infeasible to find a collision, i.e. $x \neq y$ with $h(x) = h(y)$.

Consistent Query Protocols. A query structure is a triple $(\mathcal{D}, \mathcal{Q}, Q)$ where $\mathcal{D}$ is a set of *valid* databases, $\mathcal{Q}$ is a set of possible queries, and Q is a rule which associates an answer $a_{q,D} = Q(q, D)$ with every query/database pair $q \in \mathcal{Q}, D \in \mathcal{D}$.

In a CQP, there is a server who, given a database, produces a commitment which is made public. Clients then send queries to the server, who provides the query answer along with a proof of consistency of the commitment. There may also be a public random string to be provided by a trusted third party. Though we formulate our definitions in that context, our constructions mostly do not require the third party.

Syntactically, a query protocol consists several probabilistic poly-time (PPT) algorithms: (1) a server setup algorithm $\mathcal{S}_s$, which takes the database D, a security parameter 1^k and any public randomness σ, and outputs the commitment c and some internal state information *state;* (2) an answering algorithm $\mathcal{S}_a$ which takes *state* and a query q and returns an answer-proof pair (a, π); (3) a client verification algorithm which takes a triple (c, q, a, π) and outputs "accept" or "reject;" (4) an algorithm Σ for sampling the public random string.

Definition 1. *A query protocol is* consistent *if it is complete and sound:*

- *Completeness: For every valid database D and query q, if setup is performed correctly then with overwhelming probability, $\mathcal{S}_a$ outputs both the correct answer and a proof which is accepted by $\mathcal{C}$. Formally, for all $q \in \mathcal{Q}$ and for all $D \in \mathcal{D}$,*

$$\Pr[\sigma \leftarrow \Sigma(1^k); (c, \text{state}) \leftarrow \mathcal{S}_s(\sigma, D); (a, \pi) \leftarrow \mathcal{S}_a(q, \text{state}) : \mathcal{C}(\sigma, c, q, a, \pi) = \text{"accept" and } a = Q(q, D)] \geq 1 - negl(k)$$

- *(Computational) Soundness: For every non-uniform PPT adversary $\tilde{\mathcal{S}}$: run $\tilde{\mathcal{S}}$ to obtain a commitment c along with a list of triples (q_i, a_i, π_i). We say $\tilde{\mathcal{S}}$* acts consistently *if there exists $D \in \mathcal{D}$ such that $a_i = Q(q_i, D)$ for all i for which π_i is a valid proof. The protocol is* sound *if all PPT adversaries $\tilde{\mathcal{S}}$ act consistently. Formally:*

$$\Pr[\sigma \leftarrow \Sigma(1^k); (c, (q_1, a_1, \pi_1), \ldots, (q_t, a_t, \pi_t)) \leftarrow \tilde{\mathcal{S}}; b_i \leftarrow \mathcal{C}(\sigma, c, q_i, a_i, \pi_i) : \exists \tilde{D} \text{ such that } (a_i = Q(q_i, \tilde{D}) \text{ or } b_i = 0) \text{ for all } i] \geq 1 - negl(k)$$

Privacy. Informally, we require that an adversarial client interacting with an (honest) server learn no more information from the answer/proof pairs he receives than what he gets from the answers alone. specifically, a simulator who has access only to the query answers should be able to give believable-looking proofs of consistency. The definition comes from [11,16,14], though we use a cleaner formulation due to [9].

Definition 2 (Computational privacy). *A consistent query protocol for $(\mathcal{D}, \mathcal{Q}, Q)$ is private if there exists a PPT simulator* Sim, *such that for every non-uniform PPT adversary $\tilde{\mathcal{C}}$, the outputs of the following experiments are computationally indistinguishable:*

$\sigma \leftarrow \Sigma(1^k)$,	$\sigma', c', state_{\mathsf{Sim}} \leftarrow \mathsf{Sim}(1^k)$,
$(D, state_{\tilde{\mathcal{C}}}) \leftarrow \tilde{C}(\sigma)$,	$(D, state_{\tilde{\mathcal{C}}}) \leftarrow \tilde{C}(\sigma')$,
$(c, state) \leftarrow \mathcal{S}_s(\sigma, D)$,	
Output $z \leftarrow \tilde{\mathcal{C}}^{\mathcal{S}_a(\cdot, state)}(c, state_{\tilde{\mathcal{C}}})$	*Output* $z \leftarrow \tilde{\mathcal{C}}^{\mathsf{Sim}(\cdot, state_{\mathsf{Sim}}, Q(\cdot, D))}(c', state_{\tilde{\mathcal{C}}})$

Here $\tilde{\mathcal{C}}^{\mathcal{O}(\cdot)}$ denotes running $\tilde{\mathcal{C}}$ with oracle access to $\mathcal{O}$. The simulator Sim *has access to a query oracle $Q(\cdot, D)$, but* asks only queries which are asked to Sim by $\tilde{\mathcal{C}}$.

Hiding Set Size. In general, a private protocol should not leak the size of the database [14]. Nonetheless, for the sake of efficiency we will sometimes leak a *polynomial* upper bound T on the database size, and call the corresponding protocols *size-T-private* [11]. This can be reflected in the definition by giving the simulator an upper bound T on the size of D as an additional input. One recovers the original definition by letting T be exponential, e.g. $T = 2^k$.

Interactive proofs. The definitions extend to a model where consistency proofs are interactive (although the access of the simulator to the adversarial client is more tricky).

3 Data-Robust Algorithms and Consistent Query Protocols

In this section, we describe a general framework for obtaining secure consistent query protocols, based on designing efficient algorithms which are "data-robust". Assuming the availability of a collision-resistant hash function, we show that any such algorithm which accesses its input by "following" pointers can be transformed into a consistent query protocol whose (non-interactive) consistency proofs have complexity at most proportional to the complexity of the algorithm.

Data-robust algorithms. Suppose a programmer records a database on disk in some kind of static data structure which allows efficient queries. Such data structures are often augmented with redundant information, for example to allow searching on two different fields. If the data structure later becomes corrupted, then subsequent queries to the structure might be mutually inconsistent: for example, if entries are sorted on two fields, some entry might appear in one of the two structures but not the other. A data-robust algorithm prevents such inconsistencies.

Suppose we have a query structure $(\mathcal{D}, \mathcal{Q}, Q)$. A data-robust algorithm (DRA) for these consists of two polynomial-time[2] algorithms *(T, A):* First, a setup transformation $T : D \to \{0,1\}^*$ which takes a database D and makes it into a static data structure (i.e. a bit string) $S = T(D)$ which is maintained in memory. Second, a query algorithm A which takes a query $q \in \mathcal{Q}$ and an arbitrary "structure" $\tilde{S} \in \{0,1\}^*$ and returns an answer. The structure $\tilde{S}$ needn't be the output of T for any valid database D.

Definition 3. *The algorithms (T, A) form a* data-robust algorithm *for* $(\mathcal{D}, \mathcal{Q}, Q)$ *if:*

- **Termination** *A terminates in polynomial time on* all *input pairs* $(q, \tilde{S})$*, even when* $\tilde{S}$ *is not an output from T.*
- **Soundness** *There exists a function* $T^* : \{0,1\}^* \to \mathcal{D}$ *such that* for all *inputs* $\tilde{S}$*, the database* $D = T^*(\tilde{S})$ *satisfies* $A(q, \tilde{S}) = Q(q, D)$ *for all queries* q*.*
  (There is no need to give an algorithm for T; we only need it to be well-defined.)*
- **Completeness** *For all* $D \in \mathcal{D}$*, we have* $T^*(T(D)) = D$*.*
  (That is, on input q *and T(D), the algorithm A returns the correct answer* $Q(q, D)$*.)*

We only allow A *read* access to the data structure (although the algorithm may use separate space of its own). Moreover, A is *stateless:* it shouldn't have to remember any information between invocations.

The running time of A. There is a naive solution to the problem of designing a DRA: A could scan the corrupted structure $\tilde{S}$ in its entirety, decide which database D this corresponds to, and answer queries with respect to D. The problem, of course, is that this requires at least linear time *on every query* (recall that A is stateless). Hence the

[2] We assume for simplicity that the algorithms are deterministic; this is not strictly necessary.

task of designing robust algorithms is most interesting when there are natural *sub-linear* time algorithms; the goal is then to maintain efficiency while also achieving robustness. In our setting, efficiency means the running-time of the algorithm A on *correct* inputs, in either a RAM or pointer-based model. On incorrect inputs, an adversarially-chosen structure could, in general, make A waste time proportional to the size of the structure $\tilde{S}$; the termination condition above restricts the adversary from doing too much damage (such as setting up an infinite loop, etc).

Constructing consistent query protocols from DRAs. Given a DRA which works in a pointer-based memory model, we can obtain a cryptographically secure consistent query protocol of similar efficiency. Informally, a DRA is pointer-based if it operates by following pointer in a directed acyclic graph with a single source (see the full version for details). Most common search algorithms fit into this model.

Proposition 1. *(Informally) Let* (T, A) *be a* DRA/or *query structure* $(\mathcal{D}, \mathcal{Q}, Q)$ *which fits into the pointer-based framework described above. Suppose that on inputs* q *and* $T(D)$ *(correctly formed), the algorithm* A *examines* $b(q, D)$ *memory blocks and a total of* $s(q, D)$ *bits of memory, using* $t(q, D)$ *time steps. Assuming the availability of a public collision-resistant hash function, there exists a consistent query protocol for* $(\mathcal{D}, \mathcal{Q}, Q)$ *which has prooflength* $s(q, D) + kb(q, D)$ *on query* q*. The server's computation on each query is* $O(s(q, D) + t(q, D) + kb(q, D))$.

To get a consistent query protocol from a DRA, we essentially build a Merkle tree (or graph, in fact) which mimics the structure of the data, replacing pointers with hashes of the values they point to. The client runs the query algorithm starting from hash of the unique source in the graph (that hash value is the public commitment). When the query algorithm needs to follow a pointer, the server merely provides the corresponding pre-image of the hash value.

4 Orthogonal Range Queries

In the case of join queries, a database D is a set of key/value pairs (entries) where each key is a point in $\mathbb{R}^d$, and each query is a rectangle $[a_1, b_1] \times \cdots \times [a_d, b_d]$. Note that these are also often called *(orthogonal) range queries,* and we shall adopt this terminology here for consistency with the computational geometry literature. For concreteness, we consider the two-dimensional case; the construction naturally extends to higher dimensions. In two dimensions, each query q is a rectangle $[a_1, b_1] \times [a_2, b_2]$. The query answer $Q(q, D)$ is a list of all the entries in D whose key (xkey, ykey) lies in q.

4.1 A Data-Robust Algorithm for Range Queries

Various data structures for efficient orthogonal range queries exist (see [7] for a survey). The most efficient (non-robust) solutions have query time $O((m + 1) \log^{d-1} N)$ for d-dimensional queries. We recall the construction of *multi-dimensional range trees* (due to Bentley [2]), and show how they can be queried robustly. The query time of the robust algorithm is $O((m+1) \log^d N)$. It is an interesting open question to find a robust algorithm which does as well as the best non-robust algorithms.

Algorithm 1. $A_{\text{1DRT}}([a,b],\ n,\ )$
Input: a target range $[a,b]$, a node n in a (possibly misformed) 1-DRT.
Output: a set of (key, value) pairs.

1. **if** n is not properly formed (i.e. does not contain the correct number of fields) **then** return $\emptyset$
2. **if** n is a leaf: **if** $a_n = b_n = \text{key}_n$ and $\text{key}_n \in [a,b]$, **then** return $\{(\text{key}_n, \text{value}_n)\}$ **else** return $\emptyset$
3. **if** n is an internal node:
   - $l \leftarrow \text{left}_n$, $r \leftarrow \text{right}_n$
   - **if** $a_n = a_l \leq b_l < a_r \leq b_r = b_n$ **then** return $A_{\text{1DRT}}([a,b],\ l) \cup A_{\text{1DRT}}([a,b],\ r)$
   - **else** return $\emptyset$

One-dimensional range trees. Multidimensional range trees are built recursively from one-dimensional range trees (denoted 1-DRT), which were also used by [4,16,11]. In a 1-DRT, (key, value) pairs are stored in sorted order as the leaves of a (minimum-height) binary tree. An internal node n stores the minimum (denoted a_n) and maximum (denoted b_n) keys which appear in the subtree rooted at n. For a leaf l, we take $a_l = b_l$ to be the value of the key_l key stored at l. Additionally, leaves store the value value_l associated to key_l.

Setup. Given a database $D = \{(\text{key}_1, \text{value}_1), \ldots, (\text{key}_N, \text{value}_N)\}$, the setup transformation T_{1DRT} constructs a minimum-height tree based on the sorted keys. All the intervals $[a_n, b_n]$ can be computed using a single post-order traversal.

Robust queries. It is easy to see that a 1-DRT allows efficient range queries when it is correctly formed (given the root n of a tree and a target interval $[a,b]$, descend recursively to those children whose intervals overlap with $[a,b]$). However, in our setting we must also ensure that the queries return consistent answers even when the data structure is corrupted. The data structure we will use is exactly the one above. To ensure robustness we will modify the querying algorithm to check for inconsistencies.

Assume that we are given a *rooted* graph where all nodes n have an associated interval $[a_n, b_n]$, and all nodes have outdegree either 0 or 2. A *leaf* l is any node with outdegree 0. A leaf is additionally assumed to have to extra fields key_l and value_l. Consider the following definitions:

Definition 4. *A node n is* consistent *if its interval agrees with those of its children. That is, if the children are l and r respectively, then the node is consistent if $a_n = a_l \leq b_l < a_r \leq b_r = b_n$. Moreover, we should have $a_n = b_n$ for a node if and only if it is a leaf.*

A path from the root to a node is consistent *if n is consistent and all nodes on the path to the root are also consistent.*

Definition 5. *A leaf l in a* 1-DRT *is* valid *if there is a consistent path from the root to l.*

In order to query a (possibly misformed) 1-DRT in a robust manner, we will ensure that the query algorithm A returns *exactly* the set of valid leaves whose keys lie in the target range. Thus for any string $\tilde{S}$, the database $T^*(\tilde{S})$ consists of the data at all the valid leaves one finds when $\tilde{S}$ is considered as the binary encoding of a graph.

The following lemma proves that one-dimensional range trees, along with the algorithm A_{1DRT}, form a DRA for range queries.

Lemma 1. *The algorithm A_{1DRT} will return exactly the set of valid leaves whose keys are in the target range. In the worst case, the adversary can force the queries to take time $O(s)$ where s is the total size of the data structure. Conversely, given a collection of N entries there is a tree such that the running time of the algorithm is $O((m+1)\log N)$, where m is the number of points in the target range. This tree can be computed in time $O(N \log N)$ and takes $O(N)$ space to store.*

Two-dimensional range trees. Here, the database is a collection of triples (xkey, ykey, value), where the pairs (xkey, ykey) are all distinct (they need not differ in both components). The data structure, a two-dimensional range tree (denoted 2-DRT), is an augmented version of the one above. The skeleton is a 1-DRT (called the *primary* tree), which is constructed using the xkey's of the data as its key values. Each node in the primary tree has an attached 1-DRT called its *secondary* tree:

- Each leaf l of the primary tree (which corresponds to a single xkey value $a_l = b_l$) stores all entries with that xkey value. They are stored in the 1-DRT tree_l which is constructed using ykey's as its key values.
- Each internal node n (which corresponds to an interval $[a_n, b_n]$ of xkey's) stores a 1-DRT tree_n containing all entries with xkey's in $[a_n, b_n]$. Again, this "secondary" tree is organized by ykey's.

The setup algorithm T_{2DRT} creates a 2-DRT given a database by first sorting the data on the key xkey, creating a *primary* tree for those keys, and creating a secondary tree based on the ykey for each of nodes in the primary tree. In a 2-DRT, each point is stored d times, where d is its depth in the primary tree. Hence, the total storage can be made $O(N \log N)$ by choosing minimum-height trees.

Searching in a 2-DRT. The natural recursive algorithm for range queries in this structure takes time $O(\log^2 N)$ [7]: Given a target range $[a^{(x)}, b^{(x)}] \times [a^{(y)}, b^{(y)}]$ and an internal node n, there are three cases: if $[a^{(x)}, b^{(x)}] \cap [a_n, b_n] = \emptyset$, then there is nothing to do; if $[a^{(x)}, b^{(x)}] \supseteq [a_n, b_n]$, then perform a search on the second-level tree attached to n using the target range $[a^{(y)}, b^{(y)}]$; otherwise, recursively explore n's two children.

Based on the natural query algorithm, we can construct a DRA A_{2DRT} by adding the following checks:

- All queries made to the 1-D trees (both primary and secondary) are made robustly following Algorithm 1 (A_{1DRT}), i.e. checking consistency of each explored node.
- For every point which is retrieved in the query, make sure it is present and valid in all the secondary 1-D trees which are on the path to the root (in the primary tree).

Algorithm 2. $A_{\text{2DRT}}([a^{(x)}, b^{(x)}] \times [a^{(y)}, b^{(y)}], n)$
Input: a target range $[a^{(x)}, b^{(x)}] \times [a^{(y)}, b^{(y)}]$, a node n in a 2-DRT.
Output: a set of (xkey, ykey, value) triples.

1. **if** n is not properly formed (i.e. does not contain the correct number of fields), **then** return $\emptyset$.
2. Check for consistency (if check fails, return $\emptyset$):
   - **if** n is a leaf **then** check $a_n = b_n = \text{key}_n$
   - **if** n is an internal node, **then** check $a_n = a_{\text{left}_n} \leq b_{\text{left}_n} < a_{\text{right}_n} \leq b_{\text{right}_n} = b_n$
3. a) **if** $[a_n, b_n] \cap [a^{(x)}, b^{(x)}] = \emptyset$ **then** return $\emptyset$
   b) **if** $[a_n, b_n] \subseteq [a^{(x)}, b^{(x)}]$ **then**
      - $B \leftarrow A_{\text{1DRT}}([a^{(y)}, b^{(y)}], \text{tree}_n)$
      - Remove elements of B for which xkey $\notin [a_n, b_n]$
      - **if** n is an internal node:
        For each point p in B, check that p is 2-valid in either left_n or right_n.
        If the check fails, remove p from B.
      - Return B
   c) Otherwise
      - $B \leftarrow A_{\text{2DRT}}\left(([a^{(x)}, b^{(x)}] \cap [a_{\text{left}_n}, b_{\text{left}_n}]) \times [a^{(y)}, b^{(y)}], \text{left}_n\right)$
        $\cup\, A_{\text{2DRT}}\left(([a^{(x)}, b^{(x)}] \cap [a_{\text{right}_n}, b_{\text{right}_n}]) \times [a^{(y)}, b^{(y)}], \text{right}_n\right)$
      - Remove elements of B which are not valid leaves of tree_n.
      - Return B

Definition 6. *A point* $p = (\text{xkey}, \text{ykey}, \text{value})$ *in a (corrupted)* 2-DRT *is* 2-valid *if*

1. p *appears at a valid leaf in the secondary* 1-DRT tree_l *belonging to a* leaf l *of the primary tree with key value* $\text{xkey} = a_l = b_l$.
2. *For every (primary) node* n *on the path to* l *from the root of the primary tree,* n *is consistent and* p *is a valid leaf in the (one-dimensional) tree* tree_n.

For robust range queries, we obtain Algorithm 2 (A_{2DRT}). As before, the idea is to return only those points which are 2-valid. Thus, for an arbitrary string $\tilde{S}$, the induced database $T^*_{\text{2DRT}}(\tilde{S})$ is the collection of all 2-valid points in the graph represented by $\tilde{S}$. The following lemma shows that the algorithms $(T_{\text{2DRT}}, A_{\text{2DRT}})$ form a DRA for two-dimensional range queries with query complexity $O((m+1)\log^2 N)$ (where m is the number of points in the target range).

Lemma 2. *Algorithm 2 (*A_{2DRT}*) will return exactly the set of 2-valid points which are in the target range. On arbitrary inputs,* A_{2DRT} *terminates in worst-case time O(L), where L is the total size of the data structure.*

Conversely, given a collection of N entries there is a tree such that the running time of the algorithm A_{2DRT} *is* $O((m+1)\log^2 N)$*, where* m *is the number of points in the target range. This tree can be computed in time* $O(N \log^2 N)$ *and takes O(N log N) space to store.*

One can use similar ideas to make robust range queries on **d-dimensional** keys, where $d \geq 2$. The structure is built recursively, as in the 2-D case. Although the algorithm is polylogarithmic for any fixed dimension, the exponent increases:

Lemma 3. *There exists a* DRA *for* d *dimensional range queries such that queries run in time* $O((m+1)\log^d N)$, *and the data structure requires* $O(N \log^d N)$ *preprocessing and* $O(N \log^{d-1} N)$ *storage.*

Using the generic transformation of the previous section, we obtain:

Theorem 1 (Two dimensions). *Assuming the existence of collision-resistant hash functions, there is a consistent query protocol for two-dimensional range queries with commitment size* k *and non-interactive consistency proofs of length at most* $O(k(m+1)\log^2 N)$, *where* m *is the number of keys in the query range, and* k *is the security parameter (output size of the hash function).*

For higher dimensions, our construction yields proofs of length $O(k(m+1)\log^d N)$.

5 Privacy for Consistent Query Protocols

One can construct private CQPS (Definition 2) with good asymptotic complexity using generic techniques, as follows: Universal arguments [1] allow one to (interactively) give a zero-knowledge proof of knowledge of an NP statement of arbitrary polynomial length, using only a fixed, $poly(k)$ number of bits of communication. This allows one to handle arbitrary query structures (as long as answering queries takes at most polynomial time). It also hides the set size of the database as in [14], since the universal argument leaks only a super-polynomial bound on the length of the statement being proven.

The generic technique can be made slightly more efficient by starting from a (non-private), efficient CQP, and replacing each proof of consistency π with a zero-knowledge argument of knowledge of π. With a public random string, one can also use non-interactive zero-knowledge proofs. This approach will typically leak some bound on the size N of the database. One can avoid that leakage if the original proofs take time and communication *poly* (log N), as with membership and orthogonal range queries. Replacing N with the upper bound 2^k, we once again again get $poly(k)$ communication. (A different proof of the result for membership queries can be found in [9].)

Theorem 2. *(a) Assume that there exists a collision-resistant hashfamily. For any query structure with polynomial complexity, there exists a* private CQP *with a constant number of rounds of interaction and* $poly(k)$ *communication.*

(b) Given a public random string, any CQP *with proofs of length* $\ell(N)$ *can be made size-N-private with no additional interaction at a* $poly(k\ \ell(N))$ *multiplicative cost in communication, assuming non-interactive zero-knowledge proof systems exist.*

Although the asymptotics are good, the use of generic NP reductions and probabilistically checkable proofs in [1] means that the advantages only appear for extremely large datasets. We therefore construct simpler protocols tailored to Merkle trees.

Explicit-Hash Merkle trees. The Merkle tree commitment scheme leaks information about the committed values, since a collision-resistant function cannot hide all information about its input. At first glance, this seems easy to resolve: one can replace the values a_i at the leaves of the tree with hiding commitments $C(a_i)$. However, there is often additional structure to the values $a_1, ..., a_N$. In CQPS for range queries, they are stored in sorted order. Revealing the path to a particular value then reveals its rank in the data set. The problem gets even more complex when we want to reveal a subset of the values, as we have to hide not only whether paths go left or right at each branching in the tree, but whether or not different paths overlap.

When one attempts to solve the problem using generic zero-knowledge proofs, the main bottleneck lies in proving that $y = H(x)$, given commitments $C(x)$ and $C(y)$—the circuit complexity of the statement is too high. The challenge, then, is to provide zero-knowledge proofs that a set $a'_1, ..., a'_t$ is a subset of the committed values, without going through oblivious evaluation of such complicated circuits. We present a modification of Merkle trees where one reveals all hash-function input-output pairs explicitly, yet retains privacy. We call our construction an *Explicit-Hash Merkle Tree.*

Lemma 4. *Assuming the existence of collision-resistant hash families and homomorphic perfectly-hiding commitment schemes,* explicit-hash Merkle trees *allow proving (in zero-knowledge) the consistency of* t *paths (of length* $d = \log N$*) using* $O(d \cdot t^2 \cdot k^2)$ *bits of communication, where* k *is the security parameter. The protocol uses 5 rounds of interaction. It can be reduced to a single message in the random oracle model.*

To illustrate, we apply this idea to the for one-dimensional range queries. The main drawback of the resulting protcol is that the server needs to maintains state between invocations; we denote by t the number of previous queries.

Theorem 3. *There exists an efficient,* size-*N*-private *consistent query protocol for 1-D range queries. For the* t*-th query to the server, we obtain proofs of size* $O((t + m) \cdot s \cdot k^2 \cdot \log N)$*, where* s *is the maximum length of the keys used for the data, and* m *is the total number of points returned on range queries made so far. The protocol uses 5 rounds of interaction and requires no common random string. The protocol can be made non-interactive in the random oracle model.*

Acknowledgements. We thank Leo Reyzin and Silvio Micali for helpful discussions.

References

1. B. Barak and O. Goldreich. Universal Arguments. In *Proc. Complexity (CCC) 2002.*
2. J. L. Bentley. Multidimensional divide-and-conquer. *Comm. ACM,* 23:214–229,1980.
3. A Buldas, P. Laud and H. Lipmaa. Eliminating Counterevidence with Applications to Accountable Certificate Management. *J. Computer Security,* 2002. (Originally in *CCS 2000.*)
4. A. Buldas, M. Roos, J. Willemson. Undeniable Replies to Database Queries. In *DBIS 2002.*
5. I. B. Damgård, T. P. Pedersen, and B. Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In *CRYPTO '93,* pp. 22–26.
6. A. De Santis and G. Persiano Zero-Knowledge Proofs of Knowledge Without Interaction (Extended Abstract). In *Proc. of FOCS 1992,* pp. 427–436.

7. J. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press, 1997.
8. M. T. Goodrich, R. Tamassia, N. Triandopoulos and R. Cohen. Authenticated Data Structures for Graph and Geometric Searching. In *Proc. RSA Conference, Cryptographers'Track,* 2003.
9. A. Healy, A. Lysyanskaya, T. Malkin, L. Reyzin. Zero-Knowledge Sets from General Assumptions. Manuscript, March 2004.
10. J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *24th STOC,* 1992.
11. J. Kilian. Efficiently committing to databases. Technical report, NEC Research, 1998.
12. P. Maniatis and M. Baker. Authenticated Append-only Skip Lists. ArXiv e-print cs.CR/0302010, February, 2003.
13. C. Martel, G. Nuckolls, M. Gertz, P. Devanbu, A. Kwong, S. Stubblebine. A General Model for Authentic Data Publication. Manuscript, 2003.
14. S. Micali, M. Rabin and J. Kilian. Zero-Knowledge Sets. In *Proc. FOCS 2003.*
15. S. Micali. Computationally Sound Proofs. *SIAM J. Computing,* 30(4):1253–1298, 2000.
16. S. Micali and M. Rabin. Accessing personal data while preserving privacy. Talk announcement (1997), and personal communication with M. Rabin (1999).
17. R. Merkle A digital signature based on a conventional encryption function. In *CRYPTO '87,* pp. 369–378,1988.
18. M. Naor and K. Nissim. Certificate Revocation and Certificate Update. In *7th USENIX Security Symposium,* 1998.
19. M. Naor, M. Yung. Universal One-Way Hash Functions and their Cryptographic Applications. In *21st STOC,* 1989.
20. R. Ostrovsky, C. Rackoff, A. Smith. Efficient Consistency Proofs on a Committed Database MIT LCS Technical Report TR-887. Feb 2003. See http://www.lcs.mit.edu/publications

A $2\frac{1}{8}$-Approximation Algorithm for Rectangle Tiling

Katarzyna Paluch*

Institute of Computer Science
University of Wrocław
abraka@ii.uni.wroc.pl

Abstract. We study the following problem. Given an $n \times n$ array A of nonnegative numbers and a natural number p, partition it into at most p rectangular tiles, so that the maximal weight of a tile is minimized. A tile is any rectangular subarray of A. The weight of a tile is the sum of the elements that fall within it. In the partition the tiles must not overlap and are to cover the whole array. We give a $2\frac{1}{8}-$*approximation* algorithm, which is tight with regard to the only known and used lower bound. Although the proof of the ratio of the approximation is somewhat involved, the algorithm itself is quite simple and easy to implement. Its running time is linear in the size of the array, but can be also made to be near-linear in the number of non-zero elements of the array.

1 Introduction

We study the following problem:

The RTILE problem. Given an $n \times n$ array A of nonnegative numbers and a natural number p, partition it into at most p rectangular tiles, so that the maximal weight of a tile is minimized. A tile is any rectangular subarray of A. The weight of a tile is the sum of the elements that fall within it. In the partition the tiles must not overlap and are to cover the whole array.

Previous work. Khanna, Muthukrishnan and Paterson showed in [5] that the problem is *NP*-hard to approximate to within a factor of $\frac{5}{4}$. They also gave the first approximation algorithm with ratio $2\frac{1}{2}$. Next, the factor of approximation was improved to $2\frac{1}{3}$ independently by Sharp [11] and by Lorys, Paluch [7]. The best approximation result up till now is by Berman, DasGupta, Muthukrishnan and Ramaswami [1], who delivered a $2\frac{1}{5}$-approximation algorithm. In the meantime the $2\frac{1}{4}$-approximation was proved by Lorys, Paluch in [8]. The problem has applications in load balancing, in parallel computing environments, data compression, and query optimization in databases ([5], [11], [1]).

New results. Our main result is a $2\frac{1}{8}$ approximation for the RTILE problem. The proof of the $2\frac{1}{8}$ approximation is somewhat involved. Some of the ideas evolved from those used in [8], in particular we use an extended but similar classification of arrays into types. In [8] however, it was enough to consider subarrays which had short types (of length at most 8). To obtain this result, however, we have to prove some properties for

* Partially supported by KBN grant 8 T11C 044 19.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1054–1065, 2004.

subarrays of arbitrarily long type. To do this we adopt a more orderly approach and create more refined apparatus. In case of large troublesome arrays we sort of "dig out" their regularities and notice that then proving some of their properties reduces to solving certain classes of (arbitrarily large) linear programs which describe their locally recursive structure. The approximation is also tight with regard to the used lower bound, which is the only known and used lower bound so far. In particular, we show that for every $\epsilon > 0$ there exists such an array A and the number of rectangles p, that any partition of A into p rectangles will contain a tile of weight equal to $2\frac{1}{8} - \epsilon$ times the value of the lower bound for this instance of the problem. The algorithm itself is quite simple and easy to implement. Its running time is linear in the size of the array, but can be also made to be near-linear in the number of non-zero elements of the array.

2 Preliminaries

Obviously the value of the maximum weight of a tile in the partition cannot be lower than the average weight of a tile or the maximal element in the array. Thus, if $w(A)$ denotes the weight of A, then

$$W = \max\{\frac{w(A)}{p}, \max\{a_{ij} : 1 \leq i, j \leq n\}\}$$

is the simple lower bound on the maximum weight in an optimal solution. We show how to find the tiling, in which the weight of each tile does not exceed $2\frac{1}{8}W$.

Since the weights of rectangles are considered in their relation to the value of the lower bound W, it will be convenient to rescale the array A by dividing its elements by W. After rescaling the lower bound is naturally equal to exactly 1. Now, if we partition the array A into some number of disjoint subarrays $A_1, A_2, ..., A_m$ and partition each of them separately using at most $\lfloor w(A_i) \rfloor$ tiles for a subarray A_i of weight $w(A_i)$, then we will not exceed the allowed number of tiles p. Moreover, since $p \geq \lceil \frac{w(A)}{W} \rceil$, we can use $\lceil w(A) \rceil$ tiles. Thus we are allowed to use $\lceil w(A_i) \rceil$ tiles in one case.

Further on we will say that a subarray A_i has been *well-partitioned,* if it has been partitioned into at most $\lfloor w(A_i) \rfloor$ tiles. Clearly the subarray of weight less than 1 cannot be well-partitioned. A subarray A_i partitioned into $\lceil w(A_i) \rceil$ tiles will be called *nearly-well- partitioned* and partitioned into $\lfloor w(A_i) \rfloor - 1$ tiles will be called *extra-partitioned.* For abbreviation we will also use the notion of ***f-partitioning,*** which will mean that in the partition the maximum weight of a tile does not exceed f. f will be further on referred to as *a factor.* f will always have value at least 2.

The first stage of the approach to the partition begins by looking at the columns of the array and dividing them into two classes: those having weight less than 1 (<-columns) and those having weight at least 1 (>-columns). A group of adjacent columns having weight less than 1, whose total weight is greater than 1 can be treated like a single >-column, because any statement concerning a >-column holds also for such a group (it suffices to look at the columns of a group as the elements of a >-column). Similarly a group of adjacent <-columns, whose total weight is less than 1 can be treated like a single <-column. Thus we can assume, that in the array A there are no adjacent <-columns (every group of adjacent <-columns will be merged into one column). We

will also assume that in the array every >-column is succeeded by a <-column and the first column of the array is a <-column. It will sometimes be achieved by putting artificial columns of weight 0. From [7] it is known, that any >-column can be well 2-partitioned and that any two-column subarray consisting of a <- and a >-column *(a block)* can be **well-$2\frac{1}{3}$-partitioned,** which was enough to yield **$2\frac{1}{3}$-approximation.** When we looked closer at blocks, i.e. two-column subarrays consisting of a <- and a >-column, it turned out that only one specified kind of them (a "prototype" of *a complex*) cannot be well-2-partitioned, while the rest can (be well- 2-partitioned). In order to improve the approximation we must deal with such blocks. We might try to extend them to the next block and check, whether now it could be well-partitioned with a better factor. If then the factor proved unsatisfactory we might try to extend the subarray further and so on. The process naturally has to end when the extension is no longer possible, that is when we have reached the end of the array and there are no more blocks, i.e. we are left with one <-column or nothing. In that case we are allowed to nearly-well-partition the remaining subarray, that is a subarray that successively defied **well-f-partitioning** together with the last <-column, if there is such left. Nearly-well-partitioning usually allows a better factor than well-partitioning. Aiming at **f- approximation** the algorithm can be described as follows:

```
while possible
    subarray S := the first block from the left
    if S can be well-f-partitioned, partition it
    else
        while S cannot be well-f-partitioned and extension possible
            extend S to the next block
        if S can be well-f-partitioned , partition it
        else nearly-well-f−partition S together with the remaining part
nearly-well-f-partition the remaining (which must be a <-column)
```

3 Classification into Types

Definition 1. *For every natural m, a column of type m, also referred to as an m-column, denotes a >-column having weight from the interval $[m, m+1)$. A block of type $\_\, m$ denotes a block consisting of a <-column and a m-column that can be f-partitioned into m tiles. A block of type $\sqcup m$ denotes a block consisting of two columns as above that cannot be f-partitioned into m tiles.*

Based on this, we characterize the type of the whole array by describing the type of every block contained in it. To be precise, we do it in the following way. Recall that we have assumed that the whole array consists of alternating <- and >-columns and begins with a <-column, we take every <-column, beginning from the leftmost, and put it into one block with the >-column on the right, describing its type. If the array's last column is a <-column, we denote it by _. Examples of types of the array: $\sqcup 2 \sqcup 2 \sqcup 3$ or $\sqcup 2 \_\, 3 \_$. Notice, that the type of the array does not contain information about the types of the blocks, in which a <-column is on the right of a >-column. For example, if the type of the array is $\sqcup 2 \sqcup 3$, we do not know whether the second and a third column are a block of type $2\sqcup$ or $2\_$.

The subarrays we will consider, will consist of columns of the array. The type of the subarray is given in the same way, except for two things. The subarray may begin with a >-column, then we first give the type of this column. If the subarray ends with a <-column, then we denote it by $\sqcup$, if it is the part of a block of type $\sqcup m$ in the whole array.

$\diamond$ denotes $\sqcup$ or _.

For blocks of type $\sqcup m$ we have the following lemma:

Lemma 1. *If a subarray consisting of a <-column and an m-column cannot be $(2+x)$-partitioned into m tiles, then its weight is greater than, correspondingly: $m+1+(m-1)x$, if $m \geq 3$; $2\frac{1}{2}+\frac{3}{2}x$, if $m=2$ and $2+x$, if $m=1$.*

Proof. Here we give the proof only for $m = 2$.

A block cannot be $(2+x)$-partitioned into 2 parts iff

1. the vertical partition into 2 parts is impossible, which implies that the column of type 2 has weight greater than $2+x$
2. the horizontal partition is impossible, which means, that there exists such a row b, that both the weight of b together with the part of the subarray above it is greater than $2+x$ and so the weight of b together with the part below it.

The minimal weight of the block that cannot be $(2+x)$-partitioned into 2 tiles can be estimated using linear programming. Consider Fig. 1 and the following linear program:

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{i=1}^{6} x_i \\ \text{subject to} \quad & x_2 + x_5 + x_6 < 1 \\ & x_1 \leq 1 \\ & x_1 + x_3 + x_4 \geq 2 + x \\ & x_1 + x_2 + x_3 + x_5 \geq 2 + x \\ & x_1 + x_2 + x_4 + x_6 \geq 2 + x \end{aligned}$$

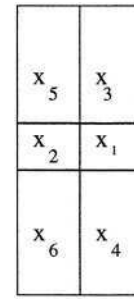


Fig.1.

We get that the weight of such a block is minimal when $x_1 = 1$, $x_2 = x_3 = x_4 = \frac{1}{2} + \frac{x}{2}$ and $x_5 = x_6 = 0$, and is equal to $2\frac{1}{2} + \frac{3}{2}x$. □

Let us observe that every block of type $\diamond m$ can always be 2-partitioned into $m+1$ tiles, since every m-column can be well-2-partitioned ([7]), i.e. 2-partitioned into m tiles. Therefore the above lemma implies that all blocks, except for blocks of type $\sqcup 2$, can be well-2-partitioned.

4 Definitions

We assign every type T a certain natural number $N(T)$ in the way described below.

Definition 2. *$N(T)$ is defined as follows. For types of columns:*

$$N(\_) = -1$$
$$N(\sqcup) = 0$$
$$N(m) = m - 1$$

For more complex types:

$$N(T_1T_2) = N(T_1) + N(T_2) + 1.$$

For allmost all types $N(T)$ is defined in such a way that it is the smallest number n, for which we are guaranteed from the definition of types of blocks, that every subarray of type T can be **f-partitioned** into $n + 1$ tiles. For example $N(\_\, m) = m - 1$ and $N(\sqcup m) = m$ and indeed we know that a block of type $\_\, m$ can be **f-partitioned** into $N(\_\, m) + 1 = m$ tiles and a block of type $\sqcup m$ can be 2-partitioned (hence also f-partitioned) into $m + 1$ tiles. Also $N(\sqcup 2 \_\, 4) = 7$ and we can **f-partition** every subarray of this type by 2-partitioning the first block into 3 tiles and **f-partitioning** the second one into 4 tiles. The above does not hold for types ended with _. (For example $N(2\_) = 1$, however we are not guaranteed that a subarray of this type can be **f-partitioned** into 2 tiles.)

$N(T)$ is also to denote that every subarray of type T, encountered in the course of carring out the algorithm, will always have weight at least $N(T)$. For some subarrays, it is immediate from Lemma 1 and the definition of types, like every subarray of type $\sqcup 2 \_\, 2$ has weight greater than $2\frac{1}{2} + \frac{3}{2}x + 2$ (the second 2 comes from the fact that a 2-column has weight at least 2) or every subarray of type $\sqcup 2 \sqcup 2$ has weight greater than $5 + 3x = N(\sqcup 2 \sqcup 2) + 3x$. However, $N(\sqcup 2 \sqcup 2 \sqcup 2 \sqcup 2) = 11$ and from Lemma 1, it only implies that every subarray of this type has weight greater than $10 + 3x$.

The notions of well- and nearly-well-partitioning are related with the actual weight of the subarray. Sometimes, however, it would be convenient to have analogous notions defined with regard to $N(T)$.

Definition 3. *To well-tile a subarray of type T means to partition it into $N(T)$ tiles. Similarly, to nearly-well-tile a subarray of type T means to partition it into $N(T) + 1$ tiles.*

We will often take advantage of the following facts:

Fact 1 *If we have a subarray S of type $T = T_1T_2$, then in order to well-tile S we can well-tile its subarray of type T_1 and nearly-well-tile its subarray of type T_2 or the other way round. Similarly, in order to nearly-well-tile S we can nearly-well-tile both its subarray of type T_1 and its subarray of type T_2.*

Proof. $N(T) = N(T_1) + N(T_2) + 1$ □

Fact 2 *Every subarray of type* $\diamond k_1 \diamond k_2 \ldots \diamond k_n$ *or of type* $\diamond k_1 \diamond k_2 \ldots \diamond k_n \sqcup$ *can always be nearly-well-2-tiled.*

We will also quite often use expressions of the form *"A* can be **well-f-tiled** or has weight at least w". To this end we have

Definition 4. *For a subarray A of type T we will say that*

$$well(A) \geq_f w$$

if A can be ***well-f-tiled*** *or has weight at least* $N(T) + w$.
Similarly, we will say that

$$nearly(A) \geq_f w$$

if A can be ***nearly-well-f-tiled*** *or has weight at least* $N(T) + w$.

Let us notice that if we have a subarray A of type $\diamond k_1 \diamond k_2 \ldots \diamond k_n$ and we are able to state that $well(A) \geq_f 1$, then we are in a good situation in the sense that assuming A has indeed weight at least $N(T)$, it means that A can be **well-f-partitioned.** It is so, because either we are able to **well-f-tile** the subarray (i.e **f-partition** it into $N(T)$ tiles) or it has weight at least $N(T) + 1$, which means in turn that for well-partitioning we are allowed to use $N(T) + 1$ tiles and thus able to well-2-partition it (block-wise).

Fact 3 *If a subarray A of type T has weight at least N(T), can be* ***nearly-f-tiled*** *and* $well(A) \geq_f 1$, *then A can be* ***well-f-partitioned.***

Often the knowledge about the subsubarrays can be combined to tell something about the subarray.

Lemma 2. *Suppose we have a subarray A of type* $T = T_1T_2$ *that consists of two subarrays B and C correspondingly of type* T_1 *and* T_2. *Suppose also that both B and C can be* ***nearly-f-tiled.*** *Then it holds*

$$well(B) \geq_f w_1 \text{ and } well(C) \geq_f w_2 \Rightarrow well(A) \geq_f w_1 + w_2 - 1.$$

Proof. It follows from Fact 1. □

5 Subarrays of Type $\sqcup 2 \diamond k$

Suppose we have a subarray A of type $T = \sqcup 2 \sqcup k$ and $k \geq 3$. From Lemma 1 we know that its weight is at least $2\frac{1}{2} + \frac{3}{2}x + k + 1 + (k-1)x$, which is greater than $N(T) = 2 + k + 1$. Let B denote the beginning complex $\sqcup 2$ and C the subarray of type $\sqcup k$. By the same lemma, we also have that $well(B) \geq_{2+x} \frac{1}{2} + \frac{3}{2}x$ and $well(C) \geq_{2+x} 1 + (k-1)x$. Thus by Fact 2 and Lemma 2 we get, that $well(A) \geq_{2+x} \frac{1}{2} + \frac{3}{2}x + (k-1)x$. Next we calculate that $well(A) \geq_{2+x} 1$ for $x \geq \frac{1}{2k+1}$, which by Fact 3 means that for $k \geq 3$,

we can always **well-2$\frac{1}{2k+1}$-partition** A, thus for $k \geq 4$, it means that we are able to **well-2$\frac{1}{9}$-partition** it. Since we aim at a **2$\frac{1}{8}$-approximation,** from this time on by writing $well(A) \geq w$ we will mean $well(A) \geq_{2\frac{1}{8}} w$ and the same for *nearly* (A). Similarly by well-, nearly-well-tiling we will mean **well-2$\frac{1}{8}$-, nearly-well-2$\frac{1}{8}$-tiling.** A complex will mean a **2$\frac{1}{8}$-complex.** We will further prove, that for every subarray A that the algorithm could not **well-2$\frac{1}{8}$-partition,** it holds $well(A) \geq \frac{5}{8}$. Thus if A is followed by B such that $well(B) \geq 1\frac{3}{8}$, then $well(AB) \geq 1$ and as all other conditions of Fact 3 and Lemma 1 will be satisfied, it will mean that AB can be **well-2$\frac{1}{8}$-partitioned.**

6 Two Neighbouring Complexes and Degenerate Subarrays

Let us now look at the subarray C of type $T = \sqcup 2 \sqcup 2$. By Lemma 1 we know that its weight is at least $5\frac{3}{8} = N(T) + \frac{3}{8}$. By the same lemma and by Lemma 2, we also know that $well(C) \geq \frac{3}{8}$. We notice that the situation seems to be worse now than in the case of only one complex, because we do not state that $well(C) \geq \frac{5}{8}$ and hence cannot say that if C is followed by some block D such that $well(D) \geq 1\frac{3}{8}$, then CD can always be **well-2$\frac{1}{8}$-partitioned.** We have to examine the structure of the subarray of type $\sqcup 2 \sqcup 2$ somewhat closer. Before doing this however, we introduce degenerate subarrays. We later prove that degenerate subarrays can always be **well-2$\frac{1}{8}$-partitioned** and also that whenever some subarray we could not **well-2$\frac{1}{8}$-partition** is followed by a degenerate subarray, then the whole can be **well-2$\frac{1}{8}$-partitioned.** It will turn out that in the non-degenerate and difficult to partition subarrays the boundaries of $\sqcup 2$, $2\sqcup$ or $\sqcup 1 \sqcup$ and >-columns (i.e.1- and 2-columns) form "nonoverlapping crosses". More precisely one type of a cross arises in the part of type $\sqcup 2 \sqcup$ if the boundaries of the complexes $\sqcup 2$ and $2\sqcup$ are in the same row, it is formed by these boundaries and the 2-column. The second type of a cross is the one in the part of type $\sqcup 1 \sqcup$, which is formed by the boundary of this part and the 1-column. The crosses non-overlap, if the boundaries of overlapping regions (except for $\sqcup 2$ and $2\sqcup$ in the part of type $\sqcup 2 \sqcup$) fall into different rows, that is, for example, if in the part of type $2 \sqcup 1 \sqcup$ the boundaries of $2\sqcup$ and $\sqcup 1 \sqcup$ are in different rows or if in the part of type $2 \sqcup 2$ the boundaries of $2\sqcup$ and $\sqcup 2$ are also in different rows. Degenerate subarrays are mostly those whose structure contradicts in some way the idea of "non-overlapping crosses".

Definition 5. *The following kinds of subarrays will be called degenerate (compare Fig 2):*

- $\sqcup 1 \sqcup 2$ *with the boundaries of subarrays* $\sqcup 1 \sqcup$ *and* $\sqcup 2$ *in the same row,*
- $\sqcup 2 \sqcup k$ *with the boundaries of the first complex and the "right" complex* $2\sqcup$ *in different rows,*
- $\sqcup 2 \sqcup k$ *with the part above (or under) the boundaries of the complexes* $\sqcup 2$ *and* $2\sqcup$ *greater than* $2\frac{1}{8}$,
- $\sqcup 1 \sqcup 1 \sqcup 2$ *with the boundaries of two subarrays of type* $\sqcup 1 \sqcup$ *in the same row.*

Also, those of type $\sqcup T$ *and T, where T is*

- $2 \sqcup 2$ *with the boundaries of subarrays* $\sqcup 2$, $2\sqcup$ *in the same row,*

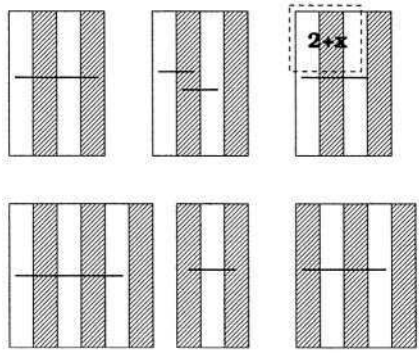


Fig. 2.

- $2 \sqcup 1 \sqcup k$ *with the boundaries of* $2\sqcup$ *and* $\sqcup 1 \sqcup$ *in the same raw.*

We define also some degenerate subarrays of type $\sqcup 2 \sqcup 3$ *and* $\sqcup 2 \sqcup 3 \sqcup 2$.

Lemma 3. *For every degenerate subarray D of type T, it holds*

$$well(D) \geq 1\frac{3}{8}.$$

Since we will further prove, that for every subarray S (of type $\diamond k_1 \diamond k_2 \ldots \diamond k_n$) $well(S) \geq \frac{5}{8}$, by the above lemma, we get that the appearance of a degenerate subarray D, causes that we are able to well-$2\frac{1}{8}$-partition S D.

Assume now we have a subarray S of type $\sqcup 2 \sqcup 2$, that is not degenerate. We will prove

Fact 4 *For every non-degenarate subarray S of type* $\sqcup 2 \sqcup 2$, *it holds*

$$well(S) \geq \frac{21}{32}.$$

Proof. Let us try to give the conditions, under which it cannot be well-tiled. First we can notice that the part $2\sqcup$ should be a complex - a so called "right" complex, because if it were not, then it could be $2\frac{1}{8}$-partitioned into 2 tiles and the whole could be $2\frac{1}{8}$-partitioned into 5 tiles, that is the whole could be well-tiled. Another condition is that the sum of the three elements contained in the boundaries of $\sqcup 2$ and $2\sqcup$ in the part $\sqcup 2 \sqcup$ should be greater than $2\frac{1}{8}$. If it were not, then we could well-tile the subarray, as we know that the part above and under this tile has weight not greater than $2\frac{1}{8}$ (because the subarray is not degenerate), and the rest (the 2-column) can easily be 2-partitioned into 2 tiles.

Let us now estimate the minimal weight of such a subarray. Since it is not degenerate its structure is as shown in Figure 3: the boundaries of the first complex $\sqcup 2$, the "right" one $2\sqcup$ and the second one $\sqcup 2$ are represented by variables correspondingly s_1, x_2 and s_1, x_4 and s_2, x_6. Variables x_1, x_3, x_5 and x_7 denote the sums of the values of the elements accordingly above or below the appropriate boundary.

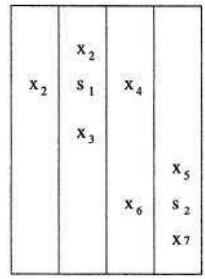


Fig. 3.

We do not need more variables in <-columns and can assume that the whole weight of <-columns is concentrated in x_2, x_4, x_6, i.e. in the boundaries. We calculate:

$$
\begin{aligned}
&\text{minimize } s_1 + s_2 + \textstyle\sum_{i=1}^{7} x_i \\
&\text{subject to} \\
&\qquad s_1 \le 1 \\
&\qquad s_1 + x_1 + x_2 \ge 2\tfrac{1}{8} \\
&\qquad s_1 + x_2 + x_3 \ge 2\tfrac{1}{8} \\
&\qquad s_1 + x_1 + x_4 \ge 2\tfrac{1}{8} \\
&\qquad s_1 + x_3 + x_4 + x_6 \ge 2\tfrac{1}{8} \\
&\qquad s_1 + x_1 + x_3 \ge 2\tfrac{1}{8} \\
&\qquad s_1 + x_2 + x_4 \ge 2\tfrac{1}{8} \\
&\qquad s_2 \le 1 \\
&\qquad s_2 + x_4 + x_5 + x_6 \ge 2\tfrac{1}{8} \\
&\qquad s_2 + x_6 + x_7 \ge 2\tfrac{1}{8} \\
&\qquad s_2 + x_5 + x_7 \ge 2\tfrac{1}{8},
\end{aligned}
$$

which turns out to be $5\frac{21}{32}$. □

7 The Algorithm in Greater Detail

In the description of the algorithm in Section 2, we start from the leftmost block and try to **well-$2\frac{1}{8}$-partition** it. From Lemma 1, we know, that unless it is a complex, its weight is at least $N(T)$ +1 and if it is a complex, its weight is greater than $N(T) + \frac{11}{16} = 2\frac{11}{16}$. Therefore in new terminology, if we want to **well-$2\frac{1}{8}$-partition** it, we in fact want to well- or nearly-tile it, depending on whether its weight is less than $N(T)$ + 1 or at least $N(T)$ + 1. Further on we will prove that every subarray encountered in the course of carrying out the algorithm, will have weight at least $N(T)$. Thus we will always be interested in either well- or nearly-tiling it. From Fact 2 we know that nearly-tiling is easy. As for well-tiling, we have that it is enough to well-tile some subarray and nearly-tile the preceding and succeeding part. Therefore in order to well-tile some subarray, we will try well-tiling its *small* subarrays i.e. those of type $\diamond k$, $k\diamond$, $\diamond 1\diamond$, $\diamond 2\diamond$ and $\diamond 3\diamond$.

$S := \emptyset$
while extension of S to the next block possible
    extend S to the next block
    if S of type T has weight at least $N(T) + 1$, nearly-well-tile S and $S := \emptyset$
    else if any small subarray can be well-tiled
    or if S is ended with a degenerate subarray,
        well-tile S and $S := \emptyset$
nearly-well-$2\frac{1}{8}$-partition S together with the last $<$-column.

Theorem 1. *The above algorithm is a $2\frac{1}{8}$ approximation algorithm for the RTILE problem.*

To prove that the above is a **$2\frac{1}{8}$-approximation** algorithm for rectangle tiling, we should show, that it is correct and also executable, that is whenever the instruction to well-tile, nearly-well-tile or **nearly-well-$2\frac{1}{8}$-partition** appears, we are capable of carring it out.

To prove, that it is correct it is enough to show, that every time we well-tile S (of type T) in the algorithm, S indeed has weight at least $N(T)$.

In the following, we show, that if the subarray of type T that does not qualify for well-tiling according to the algorithm and therefore has to be extended, has weight greater than $N(T) + \frac{1}{2} + \frac{1}{8}$.

We will say that a subarray is *difficult* if no small subarray of it can be well-tiled and if it does not contain a degenerate subarray.

The following lemma concerns subarrays arbitrarily large and is in some sense the most essential and difficult part of $2\frac{1}{8}$ approximation.

Lemma 4. *Every difficult subarray S of type $T = \sqcup 2(B \sqcup 2)^n (n \geq 0)$, where B denotes either an empty type or types $\sqcup 1$, $\sqcup 1 \sqcup 1$ or $\sqcup 3$, has weight greater than $N(T) + \frac{5}{8}$.*

Every difficult subarray S of type $T = 2A \sqcup 2(B \sqcup 2)^n (n \geq 0)$, where B is as above and A denotes either an empty type or a subarray of type $\sqcup 1$, has weight greater than $N(T) + 1$.

We will say that a subarray is *proper,* if it is of type $\diamond k_1 \diamond k_2 \ldots \diamond k_n$ or $\diamond k_1 \diamond k_2 \ldots \diamond k_n$- and its every subarray of type $T_i = \diamond k_1 \ldots \diamond k_i$ consisting of the beginning $2i$ columns has weight less than $N(T_i) + 1$.

The corollary of Lemma 4 is

Corollary 1. *(a) For every proper subarray S*

$$well(S) \geq \frac{5}{8}.$$

(b) For every proper subarray S of type $\diamond k_1 \diamond k_2 \ldots \diamond k_n$-

$$nearly(S) \geq 1.$$

From the above corollary we get, that every subarray that the algorithm tries to well-tile, has weight at least $N(T)$. First, it is true for every block. Next, suppose we have a

subarray S of type T, that we could not well-tile, then by Corollary 1(a), its weight is greater than $N(T) + \frac{5}{8}$. If we extend it to the next block B of type T', then the weight of SB is greater than $N(T) + \frac{5}{8} + N(T') + \frac{11}{16} > N(T) + N(T') + 1 = N(TT')$.

If we are able to well-tile any small subarray, then by Facts 1,2 and Corollary 1(b), we are able to well-tile the whole subarray S we are dealing with at the moment.

If the subarray S is followed by a degenerate subarray D, then by Lemma 3, Corollary 1(a) and Lemma 2, we have that *well*$(SD) > 1$.

Another implication of Corollary 1(b) is that we are always in a position to nearly-well-$2\frac{1}{8}$-partition the subarray at the end of running the algorithm, as either it can be nearly-tiled i.e. nearly-well-$2\frac{1}{8}$-partitioned as it has weight at least $N(T)$ and we use $N(T) + 1$ tiles or it has weight greater than $N(T) + 1$, but then in order to nearly-well-f-partition it, we can use $N(T) + 2$ tiles.

Theorem 2. *The approximation with a better factor using the lower bound used in this paper is impossible.*

Proof. Let us consider the subarrays of the type $\sqcup 2(\sqcup 1 \sqcup 2)^n\_$ having weight $N(T) + 1$, which for n will be equal to $5n + 3$. Let them have the following structure drawn in Figure 4:

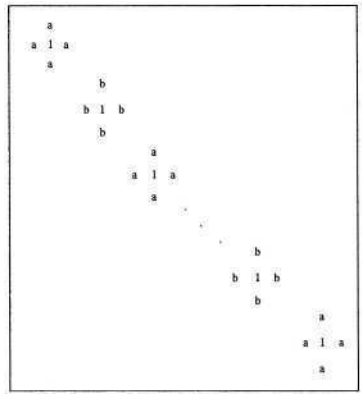


Fig. 4.

There are as many ones encircled by a as the columns of type 2 and as many ones encircled by b as columns of type 1. Additionally $b = \frac{a}{3}$. Let $a = \frac{9n+6}{16n+12}$. It can be noticed, that if the subarray is to be partitioned into $5n + 3$ tiles, then one tile will have to contain at least one 1 and two a or at least one 1, one a and three b. The weight of such a tile amounts to $2 + \frac{2n}{16n+12}$. □

References

1. P. Berman, B. DasGupta, S. Muthukrishnan, S. Ramaswami, *Efficient Approximation Algorithms for Tiling and Packing Problems with Rectangles,* J. Algorithms 41(2): 443-470 (2001).
2. M. Charikar, C. Chekuri, R. Motwani, Unpublished.
3. M. Grigni, F. Manne, *On the complexity of generalized block distribution,* Proc. 3rd intern. workshop on parallel algorithms for irregularly structured problems (IRREGULAR'96), Springer, 1996, LNCS 1117, 319-326.
4. Y. Han, B. Narahari, H.-A. Choi, *Mapping a chain task to chained processors,* Information Processing Letters 44,141-148,1992.
5. S. Khanna, S. Muthukrishnan, M. Paterson, *On approximating rectangle tiling and packing,* Proc. 19th SODA (1998), 384–393.
6. S. Khanna, S. Muthukrishnan, S. Skiena, *Efficient array partitioning,* Proc. 24th ICALP, 616-626,1997.
7. K. Loryś, K. Paluch, *Rectangle Tiling,* Proc. 3rd APPROX, Springer, 2000, LNCS 1923, 206-213.
8. K. Loryś, K. Paluch, *A new approximation algorithm for RTILE problem,* Theor. Comput. Sci. 2-3(303): 517-537 (2003).
9. G. Martin, S. Muthukrishnan, R. Packwood, I. Rhee,*Fast algorithms for variable size block matching motion estimation with minimal error.*
10. S. Muthukrishnan, V. Poosala, T. Suel, *On rectangular partitioning in two dimensions: algorithms, complexity, and applications,* Proc. 7th ICDT, 1999, 236-256.
11. J. Sharp, *Tiling Multi-dimensional Arrays,* Proc. 12th FCT, Springer, 1999, LNCS 1684, 500-511.
12. A. Smith, S. Suri, *Rectangular Tiling in Multidimensional Arrays,* J.Algorithms 37(2): 451-467 (2000).

Extensional Theories and Rewriting

Grigore Roşu

Department of Computer Science,
University of Illinois at Urbana-Champaign.
grosu@cs.uiuc.edu

Abstract. This paper is an attempt to develop a unifying algebraic framework for *extensional theories* capturing formally the informal concept of extensionality, as well as a generic automated proving technique, called *extensional rewriting,* that can be instantiated and then used to prove equational properties in various particular extensional theories.

1 Introduction

Extensionality is a powerful principle for proving equalities, based on the observation that in many theories two terms or expressions are equal whenever they prove to be equal in any context of interest. For example, under the rule of extensionality, two expressions f and g in λ-calculus are considered equal whenever they can be shown equal in contexts of the form $\star(x)$, that is, whenever $f(x) = g(x)$. Since f and g can be high-order, the extensionality rule may be applied several times before the two can be shown equal. Similarly, in set theory, under the axiom of extensionality, two sets are said to be equal whenever they have the same elements, that is, whenever they behave the same under membership contexts of the form $x \in \star$. Further, in behavioral, or observational, theories where equality is "indistinguishability under experiments" for some appropriate formalization of experiments, it can be shown that $t = t'$ whenever $\omega(t) = \omega(t')$ for any experiment generator ω, so these theories are also extensional.

This paper is an attempt to develop a unifying algebraic framework for *extensional theories,* which are theories that have extensional behavior. Using the technical concepts of *tagged term* and *context generator,* the informal notion of "extensional behavior" is properly formalized both syntactically and semantically, and a sound inference rule, called *extensionality,* is provided. Extensionality essentially states that $t = t'$ is derivable whenever $\omega(t) = \omega(t')$ is derivable for any context generator, or *extensional,* operator ω. *Extensional rewriting* further automates the process of proving equalities of terms in extensional theories. It intuitively works as follows: first reduces the two terms t and t' to their normal forms $nf(t)$ and $nf(t')$, respectively; if $nf(t) = nf(t')$ then declares the two terms equal else applies the procedure recursively on each pair $\omega(t), \omega(t')$. The method is correct only if the term equality relation is extensional.

The work presented in this paper has been mainly motivated by an effort to organize λ-calculus and combinatorial logic as behavioral theories, so that the equivalence of λ-expressions becomes a special case of behavioral equivalence.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1066–1079, 2004.

Our systematic failures to do so, despite the intuitive behavioral and extensional flavor of both, suggested us to take a general, foundational look at extensionality and develop a new framework that smoothly integrates both formalisms. To the author's knowledge, the major results presented in this paper are novel. Proofs were removed from this paper due to space limitations, but can be found in [15].

2 Extensional Theories

Here we define extensional equational theories together with the extensionality inference rule on these. Important infrastructure concepts, such as terms tagged by variables and contexts, are defined first.

2.1 Terms Tagged by Variables and Contexts

Given an algebraic many-sorted signature Σ over sorts S, a term over some variables is usually by default considered to also be a term over more variables, where the extra-variables simply do not occur in the term. Since extensionality requires pairs of terms on top of which operations potentially adding new variables can be placed, it turns out that extensional rewriting can be more elegantly and abstractly formalized if terms are tagged by the set of variables over which they are *intended* to be considered. For example, terms t and t' in an equation $(\forall X)\ t = t'$ are both intended to be defined over the set of variables X, even if not all these variables occur in each of t and t'.

Definition 1. *Given a set $\mathcal{X}$ of* ***variables****, the S-indexed set $\mathcal{T}_\Sigma(\mathcal{X})$ of* ***tagged terms over*** *$\mathcal{X}$ is defined recursively as follows: $x.\{x\}$ is in $\mathcal{T}_{\Sigma,s}(\mathcal{X})$ for each $x \in \mathcal{X}_s$; $\sigma(t_1.X_1, ..., t_n.X_n).X_1 \cup \cdots \cup X_n$ is in $\mathcal{T}_{\Sigma,s}(\mathcal{X})$ if $\sigma : s_1 ... s_n \to s$ and $t_i.X_i \in \mathcal{T}_{\Sigma,s_i}(\mathcal{X})$ for all $1 \leq i \leq n$; $t.X'$ is in $\mathcal{T}_\Sigma(\mathcal{X})$ whenever $t.X$ is in $\mathcal{T}_\Sigma(\mathcal{X})$, $X \subseteq X' \subseteq \mathcal{X}$, and X' is finite. Given a tagged term $t.X$, then X is called its* ***tag*** *and t is called its* ***term.***

Thus, many occurrences of the same term in the standard sense are allowed, indexed by finite sets of variables including the variables occurring in the term.

Proposition 1. *$\mathcal{T}_\Sigma(\mathcal{X})$ is a Σ-algebra. Moreover, if M is another Σ-algebra and $\theta : \mathcal{X} \to M$, then $\theta^* : \mathcal{T}_\Sigma(\mathcal{X}) \to M$ is a morphism of Σ-algebras, where θ^* is defined recursively: $\theta^*(x.\{x\}) = \theta(x)$; $\theta^*(\sigma(t_1.X_1, ..., t_n.X_n).X_1 \cup \cdots \cup X_n) = M_\sigma(\theta^*(t_1.X_1), ..., \theta^*(t_n.X_n))$; and $\theta^*(t.X') = \theta^*(t.X)$. However, θ^* is <u>not</u> the only morphism $\mu : \mathcal{T}_\Sigma(\mathcal{X}) \to M$ with $\mu(x.\{x\}) = \theta(x)$, and not even the only one with $\mu(x.X) = \theta(x)$ for all X with $x \in X$.*

From now on, $\theta^* : \mathcal{T}_\Sigma(\mathcal{X}) \to M$ is the designated extension of a map $\theta : X \to M$; to simplify writing, we write just θ instead of θ^*. Also, for any S-sorted signature Σ we assume an S-indexed set $\mathcal{X} = \{\mathcal{X}_s\}_s$ of variables, where each $\mathcal{X}_s$ is infinite. The notions of equation and rewriting rule need to be redefined as follows:

Definition 2. *A* ***Σ-equation of sort s*** *is a pair of terms in $\mathcal{T}_{\Sigma,s}(\mathcal{X})$ having the same tag, written $t.X = t'.X$ or $(\forall X)\ t = t'$. A* ***Σ-rewrite rule of sort s*** *is also a pair of terms having the same tag, but written $t.X \to t'.X$ or $(\forall X)\ t \to t'$.*

Note that $t.X \in \mathcal{T}_\Sigma(X)$ whenever $t.X \in \mathcal{T}_\Sigma(\mathcal{X})$; to keep the notation simple, we may write "$t \in \mathcal{T}_\Sigma(X)$" instead of "$t.X \in \mathcal{T}_\Sigma(\mathcal{X})$", and t instead of $t.X$ whenever X is understood. Terms t and t' are assumed defined over the same variables whenever the pair (t, t') occurs in any binary relation. We omit X if empty, so a ground equation $(\forall\emptyset)\ t = t'$ is written $t = t'$. The inference rules of equational logics (reflexivity, symetry, transitivity, congruence, substitution) naturally extend to our tagged notation, using the designated extension maps $\theta^\star$ when applying substitution inference steps.

Definition 3. *A* ***context generator signature*** $\Omega = \{\Omega_{s,s'}\}_{s,s' \in S}$ ***for*** Σ *is a family with* $\Omega_{s,s'}$ *a set of triples* (τ, i, φ), *called* ***context generators,*** *such that*

- τ *is a derived operator of sort* s', *i.e., an untagged term over special variables* $\_1 : s_1, \ldots, \_n : s_n$ *(*τ*'s arguments), written compactly* $\tau : s_1..s_n \to s'$;
- *the* i*-th argument sort of* τ *is* s; *and*
- φ *is a function* $\mathcal{P}_f(\mathcal{X}) \to \mathcal{T}_{\Sigma,s'}(\mathcal{X} \cup \{\star : s\})$, *where for all* $X \in \mathcal{P}_f(\mathcal{X})$, $\varphi(X)$ *is a tagged term* $\tau(x_1, .., x_{i-1}, \star : s, x_{i+1}, .., x_n).\{x_1, .., x_{i-1}, \star : s, x_{i+1}, .., x_n\}$, *with* $x_1, .., x_{i-1}, x_{i+1}, .., x_n$ *all different variables in* $\mathcal{X} - X$.

A context generator Ω can be viewed as a signature, regarding (τ, i, φ) in $\Omega_{s,s'}$ as operators $\omega : s \to s'$. Then $\mathcal{T}_\Sigma(\mathcal{X})$ becomes an Ω-***algebra*** by letting $\omega(t.X)$ be $\tau(x_1, .., x_{i-1}, t.X, x_{i+1}, .., x_n).(X \cup \{x_1, .., x_{i-1}, x_{i+1}, .., x_n\})$ for any $t.X$ of sort s, where τ and $x_1, .., x_{i-1}, \star : s, x_{i+1}, .., x_n$ are those in $\varphi(X)$. We may write $\omega(t)$ when X is understood from context. From now we regard context generator signatures Ω as S-sorted signatures.

Definition 4. *Given* Ω *as above, we define the* $S \times S$*-indexed family* $\mathcal{C}[\Omega]$ *of* Ω***-contexts*** *by extending* Ω *with* ***composition of contexts.*** *If* $\omega_1 = (\tau_1, i_1, \varphi_1) : s_1 \to s_2$ *and* $\omega_2 = (\tau_2, i_2, \varphi_2) : s_2 \to s_3$ *are contexts then we define a new context* $\omega_1; \omega_2 = (\tau, i_1, \varphi) : s_1 \to s_3$ *as follows:*

- τ *is the* Σ***-derived*** *operator where* $\tau(y_1, .., y_{i_2-1}, x_1, .., x_{n_1}, y_{i_2+1}, .., y_{n_2})$ *is the (untagged) term* $\tau_2(y_1, .., y_{i_2-1}, \tau_1(x_1, .., x_{n_1}), y_{i_2+1}, .., y_{n_2})$;
- $\varphi : \mathcal{P}(\mathcal{X}) \to \mathcal{T}_{\Sigma,s_3}(\mathcal{X} \cup \{\star : s_1\})$ *is defined as* $\varphi(Z) = \tau(y_1, .., y_{i_2-1}, x_1, .., x_{i_1-1}, \star : s_1, x_{i_1+1}, .., x_{n_1}, y_{i_2+1}, .., y_{n_2}).(X \cup Y)$, *with* $X = \{x_1, .., x_{i_1-1}, x_{i_1+1}, .., x_{n_1}\}$, $Y = \{y_1, .., y_{i_2-1}, y_{i_2+1}, .., y_{n_2}\}$, *with* $\varphi_1(Z) = \tau_1(x_1, .., x_{i_1-1}, \star : s_1, x_{i_1+1}, .., x_{n_1}).(X \cup \{\star : s_1\})$ *and with* $\varphi_2(Z \cup X)\} = \tau_2(y_1, .., y_{i_2-1}, \star : s_2, y_{i_2+1}, .., y_{n_2}).(Y \cup \{\star : s_2\})$.

$\omega = (\tau, i, \varphi) : s \to s'$ *in* $\mathcal{C}[\Omega]_{s,s'}$ *generated inductively as above are* ***contexts of sort*** s' ***for sort*** s; τ *is its* ***shape,*** i *its* ***slot,*** *and* φ *its* ***variable generator.***

Composition of contexts is associative. The constructions above seem unavoidable in order to generate arbitrary but fixed distinct variables in contexts.

2.2 Extensional Equational Theories

We next define the extensional theories and the extensionality inference rule.

Definition 5. *An* ***extensional (equational) theory*** *is a triple* (Σ, E, Ω), *where* (Σ, E) *is an S-sorted equational theory and* Ω *is a* Σ*-context generator signature whose operations are called* ***extensional****.*

Extensional operators take a term and "extend" it by placing it in some generic but fixed context. One can define a smallest or a largest extensional equivalence relation in many concrete extensional theories. To keep the formalization generic, we assume that any model comes together with its extensional equivalence.

Definition 6. *Given an S-signature* Σ *and a context generator* Ω *for* Σ, *an* ***Ω-extensional Σ-model (or algebra)*** *is a pair* $(M, \equiv_M)$, *where* M *is a* Σ*-algebra and* $\equiv_M$ *is a* Σ*-congruence on* M *which is* ***Ω-extensional****, in the sense that for any sort* s *and any* $a, a' \in M_s$, *it is the case that* $a \equiv_{M,s} a'$ *whenever*[1] $M_\tau(a_1, .., a_{i-1}, a, a_{i+1}, .., a_n) \equiv_{M,s'} M_\tau(a_1, .., a_{i-1}, a', a_{i+1}, .., a_n)$ *for all appropriate* $\omega = (\tau, i, \varphi) : s \to s'$ *in* Ω *and elements* $a_1, .., a_{i-1}, a_{i+1}, .., a_n$ *in* M. *An* Ω*-extensional* Σ*-model* $\mathcal{M} = (M, \equiv_M)$ ***satisfies*** *an equation* $(\forall X)\ t = t'$, *written* $\mathcal{M} \models^\Omega_\Sigma (\forall X)\ t = t'$, *if and only if* $\theta(t) \equiv_M \theta(t')$ *for any map* $\theta : X \to M$. *As usual, satisfaction is extended to* $\mathcal{M} \models^\Omega_\Sigma E$ *and* $E \models^\Omega_\Sigma (\forall X)\ t = t'$. *Moreover, if* (Σ, E, Ω) *is an extensional theory, then we may write* $\mathcal{M} \models (\Sigma, E, \Omega)$, *saying that* $\mathcal{M}$ *is a* ***model*** *of* (Σ, E, Ω), *and* $(\Sigma, E, \Omega) \models (\forall X)\ t = t'$.

Equational reasoning can now be extended with a very powerful new inference rule. Let us consider the "derivation" binary relation between sets of equations and equations, written $E \Vdash^\Omega_\Sigma (\forall X)\ t = t'$, defined by extending the inference rules of equational logic with the *extensionality* inference rule

$$\frac{E \Vdash^\Omega_\Sigma (\forall X_\omega)\ \omega(t.X) = \omega(t'.X) \text{ for all appropriate } \omega \in \Omega}{E \Vdash^\Omega_\Sigma (\forall X)\ t = t'},$$

where $X_\omega = X \cup \{x_1, .., x_{i-1}, x_{i+1}, .., x_n\}$ for each $\omega = (\tau, i, \varphi)$ with $\varphi(X) = \tau(x_1, .., x_{i-1}, \star, x_{i+1}, .., x_n).\{x_1, .., x_{i-1}, x_{i+1}, .., x_n\}$. It is this inference rule that motivated the definition of tagged terms, because the two terms $\omega(t.X)$ and $\omega(t'.X)$ must contain *exactly* the same variables $x_1, .., x_{i-1}, x_{i+1}, .., x_n$ which must *not* occur in X. If (Σ, E, Ω) is an extensional theory then we may write $(\Sigma, E, \Omega) \Vdash (\forall X)\ t = t'$ instead of $E \Vdash^\Omega_\Sigma (\forall X)\ t = t'$. As usual, this derivability relation generates a Σ-congruence on terms in $\mathcal{T}_\Sigma(\mathcal{X})$, say $\equiv^E$.

Theorem 1. ***(Soundness)*** *For any extensional theory* (Σ, E, Ω), $(\mathcal{T}_\Sigma(\mathcal{X}), \equiv^E)$ *is a model of* (Σ, E, Ω) *and* $E \Vdash^\Omega_\Sigma (\forall X)\ t = t'$ *implies* $E \models^\Omega_\Sigma (\forall X)\ t = t'$.

We conjecture that extensional satisfaction is incomplete. Our claim is based on a related incompleteness result for behavioral logics [2], saying that behavioral satisfaction is a Π^2_0-complete problem. As shown in Section 3.3, behavioral theories are special cases of extensional theories but over a *restricted* class of models. Therefore, the incompleteness result in [2] does not apply directly here.

[1] M_τ is the usual interpretation of a term in an algebra.

3 Examples of Extensional Theories

Here we show how set theory, λ-calculus, combinatorial logic and behavioral, or observational, theories can be regarded as extensional theories.

3.1 Sets

Let us define sets of natural numbers with union, intersection and membership, as an extensional theory on top of existing (standard) equational theories of natural numbers and booleans. Σ includes the signatures of natural numbers and booleans, and adds a sort *Set* together with operations $\emptyset :\to Set$, $\{\_\} : Nat \to Set$, $\_\cup\_ : Set \times Set \to Set$, $\_\cap\_ : Set \times Set \to Set$, and $\_\in\_ : Nat \times Set \to Bool$. E adds to the equations of numbers and booleans the following Σ-equations:

$(\forall n : Nat)\ n \in \emptyset = false,$
$(\forall n : Nat, m : Nat)\ n \in \{m\} = equal?(n, m),$
$(\forall n : Nat, s : Set, s' : Set)\ n \in (s \cup s') = (n \in s) \vee (n \in s'),$
$(\forall n : Nat, s : Set, s' : Set)\ n \in (s \cap s') = (n \in s) \wedge (n \in s').$

Even though one can construct sets using $\{\_\}$, $\cup$, and $\cap$, we do not want to commit to any particular inductive definition, because that would involve technical and likely unsatisfactory definitions of set operators. Additionally, we would like to allow as many models as possible, including ones of infinite sets.

Without extensionality, one can prove few properties of sets. Let Ω contain the membership. Formally, it contains only one operation corresponding to the triple $(\_\in\_, 2, \varphi)$, where $\varphi(X) = n \in \star$ for any set of natural, boolean and/or set variables X, n is a variable of sort *Nat* that does not occur in X, and $\star$ has sort *Set*. One can now easily prove properties of sets; e.g., in order to prove the distributivity of $\cap$ and $\cup$, by the extensionality rule one needs to derive

$$(\forall s_1, s_2, s_3 : Set, n : Nat)\ n \in s_1 \cap (s_2 \cup s_3) = n \in (s_1 \cap s_2) \cup (s_1 \cap s_3),$$

which follows by equational reasoning using the distributivity in booleans.

3.2 λ-Calculus and Combinatorial Logic

We assume the reader familiar with basic notions of λ-calculus, abbreviated λ, and combinatorial logic, abbreviated **CL.** All the λ-calculus concepts needed in this paper can be found in [1,10]. Combinatorial logic is the equational theory over a signature containing a sort *Exp* and two constants $\mathsf{S}, \mathsf{K} :\to Exp$, called *combinators,* and a binary operation $\_\ \_ : Exp \times Exp \to Exp$, called *application* and written as left associative, together with the following equations:

$$(\forall x, y, z)\ \mathsf{S}\,x\,y\,z = x\,z\,(y\,z), \qquad (\forall x, y)\ \mathsf{K}\,x\,y = x.$$

A derived operator is routinely also considered part of **CL,** namely the identity I defined as $\mathsf{S}\,K\,K$, which has the property that $\mathbf{CL} \vdash (\forall x)\ \mathsf{I}\,x = x$. λ is

also defined equationally, but its definition is more technical and we do not show it here. **CL** and $\boldsymbol{\lambda}$ can be defined in terms of each other. **CL** can be defined in $\boldsymbol{\lambda}$ by letting S and K be the $\boldsymbol{\lambda}$-expressions $\lambda x.\lambda y.\lambda z.\, x\, z\, (y\, z)$ and $\lambda x.\lambda y.\, x$, respectively, and $\boldsymbol{\lambda}$ can be defined in **CL** via the following *bracket abstraction:*

$$\begin{array}{lll} \lambda x.\, \rho \Rightarrow [x]\rho, & [x]x \Rightarrow \mathsf{SKK}, & [x]y \Rightarrow \mathsf{K}\,y, \\ [x](\rho\,\rho') \Rightarrow \mathsf{S}\,([x]\rho)\,([x]\rho'), & [x]\mathsf{K} \Rightarrow \mathsf{KK}, & [x]\mathsf{S} \Rightarrow \mathsf{KS}, \end{array}$$

for any distinct variables x and y and any expressions ρ, ρ'. Because of such translations, in what follows we make no distinction between expressions in $\boldsymbol{\lambda}$ or in **CL,** assuming that they are automatically converted to corresponding expressions within the theory in discussion. It is known that, without additional requirements, $\boldsymbol{\lambda}$ and **CL** are *not* equivalent. Indeed, $\boldsymbol{\lambda} \vdash \mathsf{SKK} = \mathsf{SKS}$, both reducing to $\lambda z.\, z$, but there is no way to show $\mathbf{CL} \vdash \mathsf{SKK} = \mathsf{SKS}$. However, in the presence of the rule of *extensionality,* which in these theories is defined as

$$\mathsf{ext}: \quad \frac{Px = Qx,\ x \text{ is } \textit{not} \text{ free in } PQ}{P = Q},$$

the two theories are known to become equivalent, that is, $\mathbf{CL} + \mathsf{ext}$ and $\boldsymbol{\lambda} + \mathsf{ext}$ can derive the same equations (modulo the corresponding translations). One can, e.g., show $\mathbf{CL} + \mathsf{ext} \vdash \mathsf{SKK} = \mathsf{SKS}$ by showing $\mathbf{CL} \vdash (\forall x)\ \mathsf{SKK}x = \mathsf{SKS}x$, which is immediate, and then by applying ext. Under extensionality, the two theories are also equivalent to $\boldsymbol{\lambda\eta}$, which is $\boldsymbol{\lambda}$ extended with the $\boldsymbol{\eta}$ equation $(\forall x, y)\ \lambda x.\, y\, x = y$. Many interesting properties hold only via extensionality.

We can naturally extend both $\boldsymbol{\lambda}$ and **CL** to appropriate extensional equational theories in the sense of Definition 5, by defining in both cases Ω to contain the application operator. Formally, Ω contains only one operation corresponding to the triple $(\_\ \_, 1, \varphi)$, where $\varphi(X) = \star\, x$ for any set of variables X and x is a variable that does not occur in X. Note that in the case of $\boldsymbol{\lambda}$-calculus, our extensionality rule is a bit more restrictive that ext, because x is taken such that it does not occur at all in PQ. Nevertheless, the following result holds:

Proposition 2. *For any equation* e *(in either* $\boldsymbol{\lambda}$ *or* **CL***), the following are equivalent:* $\boldsymbol{\lambda} + \mathsf{ext} \vdash e$, $\mathbf{CL} + \mathsf{ext} \vdash e$, $\boldsymbol{\lambda} \Vdash e$, $\mathbf{CL} \Vdash e$.

Therefore, one can use the inference rules for extensional equational theories to derive equivalences of $\boldsymbol{\lambda}$ or **CL** expressions. For instance, one can derive

$$\mathbf{CL} \Vdash \mathsf{S(K(S(KS)))(S(KS)(S(KS)))} = \mathsf{S(S(KS)(S(KK)(S(KS)(S(K(S(KS)))S)))))(KS)}$$

using the extensionality inference rule 4 times and the standard equational rules 32 times. *Extensional rewriting* will automate the application of these rules.

3.3 Behavioral Theories

There are several variants of behavioral (or observational, or hidden) theories, all extending algebraic specification with support for *behavioral (or observational)*

equivalence, which can be thought of as "indistinguishability under experiments". The different variants and formalisms tend to have different names, such as *observational logic* [9], *hidden algebra* [7], *coherent hidden algebra* [5], *swinging types* [13], *coalgebraic specification* [4], etc. The first work introducing the concept of behavioral equivalence seems to be [14]. There is a close relationship between behavioral equivalence in these formalisms and bisimulation in coalgebra [11]. Instead of choosing an existing formalism, we instead define a variant which is general enough to capture all the interesting properties of each of the other variants. The above can all be proved extensional w.r.t. their observers, so the results in this paper apply to all of them, but the formalism that we define next is tuned to make its relationship to extensional theories straightforward.

Definition 7. *A* ***hidden signature*** *is a signature over sorts $S = V \cup H$, split into* ***visible*** *(V) and* ***hidden*** *(H). An* ***experiment generator signature*** *for a hidden signature Σ is a context generator signature Ω for Σ containing only operations $\omega : h \to s$ with $h \in H$ and all the identity operations $(\_ : v, 1, \emptyset) : v \to v$ with $v \in V$. An* ***Ω-experiment*** *is an Ω-context of visible result. We let $\mathcal{E}[\Omega]$ denote the $(H \times V)$-indexed family $\{\mathcal{E}[\Omega]_{h,v}\}_{h \in H, v \in V}$, where the contexts in each $\mathcal{E}[\Omega]_{h,v} := \mathcal{C}[\Omega]_{h,v}$ are called* ***experiments of sort*** *v* ***for*** *h.*

We next define a special binary relation on hidden Σ-algebras, the behavioral equivalence, declaring two elements equivalent iff they cannot be distinguished by any experiment that can be generated with the experiment generators:

Definition 8. *Given a hidden signature Σ and an experiment generator signature for it Ω, an* ***Ω-behavioral Σ-model (or algebra)*** *is a pair $(M, \equiv_M)$, where M is a Σ-algebra and $\equiv_M$ is a Σ-congruence which is the* ***Ω-behavioral (or observational) equivalence on*** *M, that is, it is the identity on visible sorts and for any hidden sort $h \in H$, $a \equiv_{M,h} a'$ if and only if $\theta_a(\varphi(\emptyset)) = \theta_{a'}(\varphi(\emptyset))$ for any $v \in V$, any Ω-experiment $\omega = (\tau, i, \varphi) : h \to v$ with the property that $\varphi(\emptyset) = \tau(x_1, .., x_{i-1}, \star : h, x_{i+1}, .., x_n).\{x_1, .., x_{i-1}, \star : h, x_{i+1}, .., x_n\}$, and any maps $\theta_a, \theta_{a'} : \{x_1, .., x_{i-1}, \star : h, x_{i+1}, .., x_n\}$ with $\theta_a(x) = \theta_{a'}(x)$ for each x in $\{x_1, .., x_{i-1}, x_{i+1}, .., x_n\}$ and with $\theta_a(\star : h) = a$ and $\theta_{a'}(\star : h) = a'$.*

Even though most behavioral formalisms assume behavioral equivalence to be a Σ-congruence as a core requirement, it should be regarded here just as a simplifying assumption; there are indeed situations where the observational equivalence is *not* a Σ-congruence [8]. The results on extensional rewriting also apply in those situations, but one needs to consider *behavioral rewriting* [5] as the basic rewriting relation and then to define extensional rewriting using it.

Proposition 3. *Ω-behavioral Σ-models are Ω-extensional.*

We define *behavioral satisfaction* like in Definition 6. Proposition 3 says that extensional equational reasoning is sound; however, behavioral satisfaction does *not* admit complete deduction [2].

Definition 9. *A* ***behavioral theory*** *is a triple (Σ, E, Ω), where Σ is a hidden signature, E is a set of Σ-equations, and Ω is an experiment generator signature.*

As an example, let us consider a behavioral theory of infinite streams of bits. The hidden signature contains one visible sort, *Bit,* one hidden sort *Stream,* and, besides usual (visible) operations on bits, such as the bit complement ($\neg$), the following operators: $head : Stream \to Bit$ to return the head bit of a stream, $odd, even : Stream \to Stream$ to return the streams of elements on odd and on even positions, respectively, $zip : Stream \times Stream \to Stream$ to merge two streams, $dup, flip : Stream \to Stream$ to duplicate and flip each bit of a stream, respectively, and constants $zero, one, blink :\to Stream$ for streams of 0's, 1's, and repetitions of 01. The set E of equations may contain:

$$
\begin{array}{ll}
(\forall s, s')\ head(zip(s, s')) = head(s) & head(blink) = 0 \\
(\forall s, s')\ odd(zip(s, s')) = s & odd(blink) = zero \\
(\forall s, s')\ even(zip(s, s')) = s' & even(blink) = one \\
(\forall s)\ head(dup(s)) = head(s) & (\forall s)\ head(flip(s)) = \neg head(s) \\
(\forall s)\ odd(dup(s)) = s & (\forall s)\ odd(flip(s)) = flip(odd(s)) \\
(\forall s)\ even(dup(s)) = s & (\forall s)\ even(flip(s)) = flip(even(s)) \\
head(zero) = 0 & head(one) = 1 \\
odd(zero) = zero & odd(one) = one \\
even(zero) = zero & even(one) = one
\end{array}
$$

The behavior of stream operations is defined by *head, odd,* and *even.* This definitional style is often called *coinductive,* because of its dual flavor to inductive definitions: *head, odd,* and *even* play a role dual to constructors in inductive definitions. They also act as generators, but for experiments: two infinite streams are equivalent iff they cannot be distinguished by experiments performed with *head, odd,* and *even.* These three operations form a set of generators for experiments that can "reach" a certain bit in a stream using an exponentially shorter experiment than the standard head/tail pair.

Therefore, we define Ω to contain these three operations. Rigorously, Ω contains the triples $\omega_{head} = (head(\_), 1, \varphi_{head})$, $\omega_{odd} = (odd(\_), 1, \varphi_{odd})$, and $\omega_{even} = (even(\_), 1, \varphi_{even})$, where $\varphi_\sigma(X) = \sigma(\star : Stream).\{\star : Stream\}$ for any $\sigma \in \{head, odd, even\}$ and any set X of variables. With these, the extensionality inference rule for the behavioral theory (Σ, E, Ω) of streams becomes:

$$\frac{E \Vdash^{\Omega}_{\Sigma} (\forall X)\ \sigma(t) = \sigma(t') \text{ for each } \sigma \in \{head, odd, even\}}{E \Vdash^{\Omega}_{\Sigma} (\forall X)\ t = t'}.$$

It is now a simple exercise to derive all the following equations, by potentially repeated appeals to extensionality:

$$
\begin{array}{lll}
zip(zero, zero) = zero & zip(zero, one) = blink & (\forall s)\ zip(s, s) = dup(s) \\
(\forall s)\ zip(odd(s), even(s)) = s & (\forall s, s')\ flip(zip(s, s')) = zip(flip(s), flip(s')) &
\end{array}
$$

All the above can be proved *automatically* by extensional rewriting.

4 Abstract Extensional Rewriting

Many important rewriting results are special cases of results about *abstract rewriting systems (ARS)* (see, e.g., [12]), which are pairs $(A, \to)$, with A a set and

$\rightarrow$ a binary relation on A. In this section we generalize ARSs, in what A has a structure of algebra, to *Ω-abstract rewriting systems (Ω-ARS)*, and give several abstract rewriting techniques and results.

Definition 10. *Let Ω be a many-sorted signature containing only unary operations. An **Ω-abstract rewriting system**, abbreviated Ω-ARS, is a pair $(A, \rightarrow)$, where A is an Ω-algebra and $\rightarrow$ is a relation on A compatible with the operations.*

If Ω is empty then Ω-ARSs are just ARSs. In this section, we consider an arbitrary but fixed Ω-ARS, $(A, \rightarrow)$. We let A also denote the carrier of the algebra A, and let $A \times A$ denote the usual product (both as an algebra and as a set). Let $\stackrel{*}{\rightarrow}$ denote the reflexive and transitive closure of $\rightarrow$. Intuitively, extensional rewriting in Ω-ARSs attempts to show two elements equivalent first by rewriting them, and then, if this fails, by iteratively applying the operations in Ω on each of the pairs and repeating the process on each of the newly formed pairs. To define this formally, we next introduce some useful notions and conventions, taking the liberty to write $(a, b) \stackrel{*}{\rightarrow} (a', b')$ whenever $a \stackrel{*}{\rightarrow} a'$ and $b \stackrel{*}{\rightarrow} b'$.

Definition 11. *Finite lists of pairs in $A \times A$ are called **goals in** A and we refer to them using letters G, G', G_1, etc., or lists of pairs $[(a_1, b_1), ..., (a_n, b_n)]$, or even implicit matching $[X, (a_i, b_i), Y]$, where X and Y are sequences of pairs $(a_1, b_1), ..., (a_{i-1}, b_{i-1})$ and $(a_{i+1}, b_{i+1}), ..., (a_n, b_n)$, respectively. Let $\mathcal{G}_A$ denote the set of goals in A. An operation $\omega : s \rightarrow s'$ in Ω is **appropriate for** $a \in A_{s''}$ iff $s = s''$. Given a goal $G = [X, (a_i, b_i), Y]$ s.t. there is some appropriate $\omega \in \Omega$ for a_i and b_i, let $G_{\Omega(i)}$ be $[X, (A_{\omega_1}(a_i), A_{\omega_1}(b_i)), ..., (A_{\omega_k}(a_i), A_{\omega_k}(b_i)), Y]$, where $\omega_1, ..., \omega_k$ with $k \geq 1$ are all the appropriate operations in Ω for a_i and b_i.*

$G_{\Omega(i)}$ is thus undefined if there is no $\omega \in \Omega$ that is appropriate for a_i (and b_i). We use $G_{\Omega(i_1, i_2, ..., i_n)}$ as a shorthand for $(...((G_{\Omega(i_1)})_{\Omega(i_2)})...)_{\Omega(i_n)}$, often written $G_{\Omega(\alpha)}$, where α is the sequence $(i_1, ..., i_n)$. Definedness of $G_{\Omega(i)}$ extends naturally to sequences α. Given a sequence (i, j) and a goal $G = [X, (a_i, b_i), Y]$, j may refer to a pair in $(A_{\omega_1}(a_i), A_{\omega_1}(b_i)), ..., (A_{\omega_k}(a_i), A_{\omega_k}(b_i))$, so in general $G_{\Omega(\alpha, \beta)}$ may be defined even if $G_{\Omega(\beta)}$ is not. We sometimes call a sequence α **valid for** G iff $G_{\Omega(\alpha)}$ is defined. Notice that validity of α does not depend on the concrete elements in the pairs of a goal, but only on their sort and their order.

Definition 12. *Let $\rightrightarrows_r, \rightrightarrows_e$ be relations on $\mathcal{G}_A$, called **rewriting** and **extensionality**, respectively, defined as $G \rightrightarrows_r G'$ iff $G = [(a_1, b_1), ..., (a_n, b_n)]$, $G' = [(a'_1, b'_1), ..., (a'_n, b'_n)]$, and $(a_i, b_i) \stackrel{*}{\rightarrow} (a'_i, b'_i)$ for all $1 \leq i \leq n$, and $G \rightrightarrows_e G'$ iff $G' = G_{\Omega(i)}$ for some $1 \leq i \leq n$. Let $\rightrightarrows$ be the relation $(\rightrightarrows_r \cup \rightrightarrows_e)^*$. If $G \rightrightarrows G'$ then we say that G **extensionally rewrites to** G'.*

Let $\surd$ denote the diagonal relation on $\mathcal{G}_A$, i.e., $\surd = \{(G,G) \mid G \in \mathcal{G}_A\}$. Given two relations $R \subseteq A \times B$ and $S \subseteq B \times C$, their composition is the relation $R;S \subseteq A \times C$ defined as $a\ (R;S)\ c$ iff there is some $b \in B$ such that $a\ R\ b$ and $b\ S\ c$. Since functions are special relations, one can have compositions between functions and relations as well. R^{-1} is the **inverse** of a relation R, that is, $a\ R^{-1}\ b$ iff $b\ R\ a$. Thus, a relation R is *confluent* iff $(R^{-1})^*;R^* \subseteq R^*;(R^{-1})^*$. We take the liberty to use the "mirror" notation for inverses whenever it makes sense; for example, given a relation $\Rightarrow$, then $\Leftarrow$ is the same as $(\Rightarrow)^{-1}$.

Proposition 4. *The following hold: (1)* $\rightrightarrows_r;\rightrightarrows_r = \rightrightarrows_r$; *(2)* $G \rightrightarrows_r G'$ *yields* $G_{\Omega(i)} \rightrightarrows_r G'_{\Omega(i)}$ *for all* $1 \leq i \leq |G|$; *(3)* $\rightrightarrows_r;\rightrightarrows_e \subseteq \rightrightarrows_e;\rightrightarrows_r$; *(4)* $\rightrightarrows_r;\overset{*}{\rightrightarrows}_e \subseteq \overset{*}{\rightrightarrows}_e;\rightrightarrows_r$; *(5)* $\rightrightarrows = \overset{*}{\rightrightarrows}_e;\rightrightarrows_r$; *(6)* $\leftrightarrows_e;\rightrightarrows_r \subseteq \rightrightarrows_r;\leftrightarrows_e$; *(7)* $\overset{*}{\leftrightarrows}_e;\rightrightarrows_r \subseteq \rightrightarrows_r;\overset{*}{\leftrightarrows}_e$; *(8)* $\leftrightarrows_e;\rightrightarrows_e \subseteq \surd \cup (\rightrightarrows_e;\leftrightarrows_e)$; *(9)* $\overset{*}{\leftrightarrows}_e;\rightrightarrows_e \subseteq (\surd \cup \rightrightarrows_e);\overset{*}{\leftrightarrows}_e$; *(10)* $\rightrightarrows_e$ *is confluent, that is, given a goal G and sequences* α, β *of valid indexes* ($G_{\Omega(\alpha)}$ *and* $G_{\Omega(\beta)}$ *exist), there are some sequences of indexes* α', β' *such that* $G_{\Omega(\alpha,\beta')} = G_{\Omega(\beta,\alpha')}$; *(11)* $\leftrightarrows;\rightrightarrows_e \subseteq (\surd \cup \rightrightarrows_e);\leftrightarrows$; *(12)* $\leftrightarrows;\overset{*}{\rightrightarrows}_e \subseteq \overset{*}{\rightrightarrows}_e;\leftrightarrows$; *(13) If* $\rightarrow$ *is confluent then* $\rightrightarrows_r$ *is strongly confluent; (14) If* $\rightarrow$ *is confluent then* $\leftrightarrows;\rightrightarrows_r \subseteq \rightrightarrows_r;\leftrightarrows$; *(15) If* $\rightarrow$ *is confluent then* $\rightrightarrows$ *is strongly confluent.*

A goal is **trivial** iff it contains only pairs of equal elements. Let $\downdownarrows_n$ be the relation defined as $a \downdownarrows_n b$ iff $[(a,b)]\ ((\surd \cup \rightrightarrows_e)^n;\rightrightarrows_r)\ G_\perp$ for some trivial goal $G_\perp$, and let $\downdownarrows$ be $\bigcup_{n \geq 0} \downdownarrows_n$. Therefore, $a \downdownarrows_n b$ iff the goal $[(a,b)]$ can be extensionally reduced to a trivial goal by at most n extensional steps. Obviously, $\downdownarrows_n \subseteq \downdownarrows_{n+1}$ for all $n \geq 0$, and $a \downdownarrows b$ iff $[(a,b)] \rightrightarrows G_\perp$ for some trivial goal $G_\perp$. A relation R on A is **extensional** iff for any $a, b \in A$ for which there are some appropriate $\omega \in \Omega$, $a\ R\ b$ whenever $A_\omega(a)\ R\ A_\omega(b)$ for all appropriate $\omega \in \Omega$.

Lemma 1. *The following hold: (1)* $\downdownarrows$ *is reflexive; (2)* $\downdownarrows$ *is symmetric; (3) If* $\rightarrow$ *is confluent then* $\downdownarrows$ *is transitive; (4)* $\downdownarrows$ *is extensional; (5) If* $a \downdownarrows_n b$ *for some* $n \geq 1$ *then there is some* $m < n$ *with* $A_\omega(a) \downdownarrows_m A_\omega(b)$.

Theorem 2. $\downdownarrows$ *is included in any extensional congruence including* $\rightarrow$. *If* $\rightarrow$ *is confluent then* $\downdownarrows$ *is the smallest extensional congruence that includes* $\rightarrow$.

Theorem 2 gives a mechanizable technique for proving equalities in extensional theories. The idea is to first define a rewrite relation $\rightarrow$ included in the extensional provability relation in the respective theory, typically by orienting several or all the equational axioms of that theory, and then to apply the relation $\downdownarrows$ on the two given terms in some systematic way covering all possible derivations. If a trivial goal is encountered then the two terms are equivalent. Moreover, if the extensional provability is the smallest extensional congruence over terms including $\rightarrow$, typically realized by defining $\rightarrow$ via ordering *all* the equational axioms of the theory, then this technique becomes a semi-decision procedure for extensional provability whenever $\rightarrow$ is confluent. The technique above is clearly impractical. We next propose a generic algorithm for proving equalities in extensional theories that tends to work well in many situations.

ALGORITHM $\mathcal{A}(a, a')$

1. initialize G with $[(a, a')]$
2. rewrite all terms in all pairs in G to normal forms (relation $\rightrightarrows_r$)
3. remove from G all pairs containing equal terms
4. **if** $(G = \emptyset)$ **then return** *YES*
5. **if** *stop*?(G) **then return** G
6. let (b, b') be *pick*$(G) \in G$
7. let G be $(G - \{(b, b')\}) \cup \{(\omega(b), \omega(b')) \mid \omega \in \Omega$ appropriate for $b, b'\}$
8. **goto** 2.

stop?(G) and *pick(G)* are uninterpreted here and need to be defined for each particular framework; they are intended to stop the algorithm when certain conditions are fulfilled and to pick an appropriate pair to expand, respectively. The current set of proof tasks is returned if the algorithm is stopped; one can use different techniques to prove them. E.g., *stop?(G)* can hold when G reaches a certain maximal size, or when it contains some "bad patterns", etc. *pick(G)* may simply pick randomly a pair, or, more intelligently, it may pick a pair that may likely lead to termination of $\mathcal{A}$. A good definition of these functions can make a crucial difference in practical situations. The idea underlying the algorithm above is straightforward: iteratively reduce all the terms to their normal forms, remove the trivial pairs, then apply one extensionality, and then repeat. The next sections will show this procedure at work in different extensional theories.

Proposition 5. *If* $\mathcal{A}(a, a')$ *returns YES then* $a \downdownarrows a'$, *that is,* $\mathcal{A}$ *is sound.*

The next section shows conditions under which $\mathcal{A}$ is both sound and complete for extensional provability in some specific theories.

5 Experimenting with Extensional Rewriting

The algorithm above can be implemented on top of any term rewriting engine. We have manually applied it using the Maude [3] system and derived equalities using the extensional theories in Section 3. In Maude all equations are treated as rewrite rules, so we do not need to explicitly define $\rightarrow$. Our experiments with λ-calculus and behavioral theories are reported in [15]. In the case of sets, which can be organized as a special case of behavioral theory, the application of the algorithm is straightforward because the extensionality rules can be applied only once, so we do not discuss this case here.

5.1 λ-Calculus and Combinatorial Logic

An algebraic setting is presented next where extensional rewriting is a sound and complete, but obviously undecidable, procedure for equivalence of expressions in extensional λ-calculus and combinatorial logic, respectively. It is also shown how the algorithm $\mathcal{A}$ above works as a semi-decision procedure to prove equivalences.

When the equations defining **CL** as in Section 3.2 are regarded as rewrite rules, the corresponding rewriting relation is called *weak reduction,* denoted $\rightarrow_w$, and known to be confluent [1]. Similarly, $\rightarrow_\beta$ denotes the β-reduction relation in $\boldsymbol{\lambda}$. Let $\mathcal{T}_{\mathbf{CL}}$ and $\mathcal{T}_{\boldsymbol{\lambda}}$ be the two Ω-algebras of expressions in **CL** and $\boldsymbol{\lambda}$, respectively, defined like after Definition 3. Then clearly $(\mathcal{T}_{\mathbf{CL}}, \rightarrow_w)$ and $(\mathcal{T}_{\boldsymbol{\lambda}}, \rightarrow_\beta)$ are Ω-ARSs.

Theorem 3. *If* $\rightrightarrows^{\mathbf{CL}}$, $\downdownarrows^{\mathbf{CL}}$ *and* $\rightrightarrows^{\boldsymbol{\lambda}}$, $\downdownarrows^{\boldsymbol{\lambda}}$ *are the extensional rewriting relations associated to* $\rightarrow_w$ *and* $\rightarrow_\beta$ *like in Section 4, then* $\rightrightarrows^{\mathbf{CL}}$ *and* $\rightrightarrows^{\boldsymbol{\lambda}}$ *are strongly confluent, and* $\downdownarrows^{\mathbf{CL}} = \equiv_{\mathbf{CL+ext}}$ *and* $\downdownarrows^{\boldsymbol{\lambda}} = \equiv_{\alpha\beta\eta}$. *Therefore,* $\rightrightarrows^{\mathbf{CL}}$ *and* $\rightrightarrows^{\boldsymbol{\lambda}}$ *are sound and complete for extensional* **CL** *and* $\boldsymbol{\lambda}$*-calculus, respectively.*

Since Ω has only one operation, goals have one pair. Specifically, $[(s,t)] \rightrightarrows [(s',t')]$ iff there are variables $x_1, x_2, ..., x_n$ which do not occur in s and/or t, such that $[(s,t)] \rightrightarrows_e^* [(sx_1x_2...x_n, tx_1x_2...x_n)]$ and $sx_1x_2...x_n \rightarrow^* s'$ and $tx_1x_2...x_n \rightarrow^* t'$, where $\rightrightarrows_e$ and $\rightarrow$ can be either $\rightrightarrows_e^{\mathbf{CL}}$ and $\rightarrow_w$ or $\rightrightarrows_e^{\boldsymbol{\lambda}}$ and $\rightarrow_\beta$.

It is known that **CL** + ext is actually equivalent to an equational theory which finitely extends **CL**, in the sense that there is a finite set of equations $A_{\beta\eta}$, such that **CL** + ext is equivalent to **CL** + $A_{\beta\eta}$. Specifically, there are four such equations [1] and $\mathcal{A}$ can prove them all automatically, so:

$\mathbf{CL} \Vdash \mathsf{S(S(KS)(S(KK)(S(KS)K)))(KK)} = \mathsf{S(KK)}$,
$\mathbf{CL} \Vdash \mathsf{S(KS)(S(KK))} = \mathsf{S(KK)(S(S(KS)(S(KK)(SKK)))(K(SKK)))}$,
$\mathbf{CL} \Vdash \mathsf{S(K(S(KS)))(S(KS)(S(KS)))} = \mathsf{S(S(KS)(S(KK)(S(KS)(S(K(S(KS)))S)))))(KS)}$,
$\mathbf{CL} \Vdash \mathsf{S(S(KS)K)(K(SKK))} = \mathsf{SKK}$.

$\mathcal{A}$ needs to apply 3, 3, 4, and 2 times the rule of extensionality (step 7), respectively, before it terminates with the answer *YES* for each of the four tasks. That means that, e.g., the expressions in the third equation are *not* in the relations $\downdownarrows^{\mathbf{CL},0}$, $\downdownarrows^{\mathbf{CL},1}$, $\downdownarrows^{\mathbf{CL},2}$ and $\downdownarrows^{\mathbf{CL},3}$, but that they are in the relation $\downdownarrows^{\mathbf{CL},4}$.

Other interesting applications of extensional rewriting can be found in defining and proving properties about numerals in $\boldsymbol{\lambda}$. Let us define the $\boldsymbol{\lambda}$-expressions:

$$0 = \lambda x.\lambda y.y, \qquad + = \lambda x.\lambda y.\lambda z.\lambda u.\, x\, z\, (y\, z\, u),$$
$$s = \lambda x.\lambda y.\lambda z.\, y\, (x\, y\, z), \qquad * = \lambda x.\lambda y.\lambda z.\, x\, (y\, z).$$

Using the bracket abstraction algorithm in Section 3.2, these can be translated in **CL** expressions, and then one can use the algorithm $\mathcal{A}$ for **CL** to prove that these numerals satisfy the Peano axioms, that is, that $\mathbf{CL} \Vdash (\forall n) + 0\, n = n$, that $\mathbf{CL} \Vdash (\forall n, m) + (s\, n)\, m = s\, (+\, n\, m)$, and so on. These require 2 or 3 extensionality steps to be proved. Note that commutativity of + (and $*$) does not hold, because this is not an initial model of Peano numerals, so $\mathcal{A}$ will loop forever on the equation $(\forall n, m)\ +n\,m = +m\,n$ unless stopped at its step 5. Note, however, that all properties involving concrete numerals should hold, and that some of them need extensionality to be proved. E.g., it takes $\mathcal{A}$ about 80,000 rewrite steps and 2 applications of extensionality to prove $\mathbf{CL} \Vdash *\,(+\,3\,6)\,(+\,(*\,9\,8)\,5)) = (*\,7\,(*\,9\,(+\,3\,8)))$, where $3 = s\,(s\,(s\,0))$, etc.

5.2 Behavioral Theories

All the equations at the end of Section 3.3 can be derived using the algorithm $\mathcal{A}$ associated to the behavioral theory of streams. However, note that the equation *flip(zero) = one cannot* be derived. This is because the extensionality step generates, among other two, the proof task *odd (flip (zero)) = odd(one),* which reduces to *flip(zero) = one,* and so on, infinitely. The problem here is that *flip* also occurs in the right-hand terms of its odd/even definition. An interesting question is whether removing *flip* leads to termination of $\mathcal{A}$. Another interesting question is whether in that case $\mathcal{A}$ can prove any property of streams that can be derived extensionally. We next give general criteria under which both these questions have positive answers. In what follows let (Σ, E, Ω) be a behavioral theory where Ω corresponds to a subset $\Delta \subseteq \Sigma$, let $\equiv_{Eq,\Delta}$ be the Ω-(or rather Δ-)extensional derivability relation on Σ-terms induced by E, and let $\Rrightarrow$ be the rewriting relation generated by E regarded as rewrite rules. Also, assume that $\mathcal{A}$ stops when G has no term of hidden sort rooted in $\Sigma - \Delta$, and otherwise picks a pair containing such a term and applies it the extensionality step. Then

Theorem 4. *If $\Rrightarrow$ terminates and each rule $(\forall X)\ l \to r$ is such that r is a Δ-term and l is not a term $\delta(W)$ with $\delta \in \Delta$, then $\mathcal{A}$ terminates. If in addition*

- *$\Rrightarrow$ is confluent,*
- *if $\delta(W, \delta'(t_1, ..., t_n))$ is a left hand side of a rewriting rule with $\delta, \delta' \in \Delta$, then $t_1, ..., t_n$ are all variables,*
- *for each $\delta, \delta' \in \Delta$ (not necessarily distinct) there is a $\delta_1 \in \Delta$ s.t. neither $\delta_1(W_1, \delta(W))$ nor $\delta_1(W_1, \delta'(W'))$ appears as left hand side in a rewrite rule,*

then $t \equiv_{Eq,\Delta} t'$ iff $\mathcal{A}(t, t')$ returns YES.

$\delta(W)$ is the term having δ at top and only variables in W as arguments. Note that this result can be used to show that if we remove *flip* from the behavioral theory of streams then $\mathcal{A}$ terminates and can prove any derivable property.

6 Conclusion

The notion of extensional theory has been presented, together with an automated technique combining term rewriting and extensionality to show equalities of terms in extensional theories, called extensional rewriting, Two important case studies have been discussed, **λ-calculus** and behavioral satisfaction, together with examples showing the generality and the strength of the presented formalism.

Areas of further research include an implementation of and more experimentation with the extensional rewriting algorithm, investigating new applications, understanding in depth the relationship between extensional rewriting and circular coinductive rewriting [6], and especially characterizing those extensional theories in which circular reasoning is sound.

References

1. H. P. Barendregt. *The Lambda Calculus.* Number 103 in Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, revised edition, 1991.
2. S. Buss and G. Roşu. Incompleteness of behavioral logics. In *Proceeding of CMCS'00,* volume 33 of *ENTCS,* pages 61–79. Elsevier, 2000.
3. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. SRI International, January 1999, `http://maude.csl.sri.com`.
4. A. Corradini. A complete calculus for equational deduction in coalgebraic specification. In *Proceedings WADT'97,* volume 1376 of *LNCS.* Springer, 1997.
5. R. Diaconescu and K. Futatsugi. *CafeOBJ Report.* World Scientific, 1998. AMAST Series in Computing, volume 6.
6. J. Goguen, K. Lin, and G. Roşu. Circular coinductive rewriting. In *Proceedings of Automated Software Engineering 2000,* pages 123–131. IEEE, 2000.
7. J. Goguen and G. Malcolm. A hidden agenda. *J. of TCS,* 245(1):55–101, 2000.
8. J. Goguen and G. Roşu. Hiding more of hidden algebra. In *Proceeding of FM'99,* volume 1709 of *LNCS,* pages 1704–1719. Springer, 1999.
9. R. Hennicker and M. Bidoit. Observational logic. In *Proceedings of AMAST'98,* volume 1548 of *LNCS,* pages 263–277. Springer, 1999.
10. J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinatory Logic and Lambda Calculus.* Cambridge University Press, 1986.
11. Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Computer Science,* 62:222–259, 1997.
12. J.W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science,* Volumes 1 and 2, Clarendon, 1992.
13. P. Padawitz. Swinging data types: Syntax, semantics, and theory. In *Proceedings, WADT'95,* volume 1130 of *LNCS,* pages 409–435. Springer, 1996.
14. H. Reichel. Behavioural equivalence - a unifying concept for initial and final specifications. *Proc., 3rd Hungarian CS Conference.* Akademiai Kiado, 1981. Budapest.
15. G. Roşu. Extensional Theories and Rewriting - Extended version. University of Illinois, Technical Rep. `http://fsl.cs.uiuc.edu/~grosu/download/icalp04.pdf`.

Hardness of String Similarity Search and Other Indexing Problems

S. Cenk Sahinalp[1] and Andrey Utis[2]

[1] School of Computing Science, Simon Fraser University, BC, Canada
cenk@cs.sfu.ca
[2] Department of Computer Science, University of Maryland, College Park, USA
utis@cs.umd.edu

Abstract. Similarity search is a fundamental problem in computer science. Given a set of points $A = \{A_1, \ldots, A_p\}$ from a universe U and a distance measure D, it is possible to pose similarity search queries on a point Q in the form of nearest neighbors (find the string that has the smallest edit distance to a query string) or in the form of furthest neighbors (find the string that has the longest common subsequence with a query string).

Exact similarity search appears to be a very hard problem for most application domains; available solutions require either a preprocessing time/space exponential with p or query time exponential with $|Q|$. For such problems approximate solutions have recently attracted considerable attention. Approximate nearest (furthest) neighbor search aims to find a point in A whose distance to query point Q is within a small multiplicative factor of that between Q and its nearest (furthest) neighbor.

In this paper, we study hardness of several important similarity search problems for strings as well as other combinatorial objects, for which exact solutions have proven to be very difficult to achieve. We show here that even the approximate versions of these problems are quite hard; more specifically they are as hard as exact similarity search in Hamming space. Thus available cell probe lower bounds for exact similarity search in Hamming space apply for approximate similarity search in string spaces (under Levenshtein edit distance and longest common subsequence) as well as other spaces.

As a consequence of our reductions we also make observations about pairwise approximate distance computations. One such observation gives a simple linear time 2-approximation algorithm for permutation edit distance.

1 Introduction

Similarity search is a fundamental problem in computer science with applications in molecular biology, databases, pattern recognition, data compression, etc. Similarity search queries can be posed in the form of "nearest neighbors" (NN) queries (e.g. find the string that has the smallest Levenshtein edit distance to a query string) or "furthest neighbors" (FN) queries (e.g. find the string that has the longest common subsequence with a query string). In this paper, we will say Nearest Neighbors if we want to minimize a certain distance measure D, and we will say Furthest Neighbors if we want to

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1080–1098, 2004.

maximize a certain distance measure D; this is consistent notation but may be counter-intuitive in some cases such as longest common subsequence. Similarity search problems in many domains suffer from the curse of dimensionality; i.e. given set of points $A = \{A_1, A_2, \ldots A_p\}$ and a query Q, all known algorithms for exact NN (ENN) or exact FN (EFN) for these domains either have a preprocessing time exponential with p or have a search time exponential with $|Q| = n$. (n could be the number of dimensions in a vector, the number of characters in a string, etc.)

Perhaps the most fundamental and best studied domain in similarity search is the *Hamming space*. Recently established cell probe lower bounds indicate that any ENN (equivalently EFN) data structure for Hamming distance on binary vectors of size n requires either $2^{\Omega(\log p \log n)}$ time/space preprocessing or $\Omega(p^{1-\epsilon})$ time querying, provided that the number of probes used are sublogarithmic with n [2,1,14]. These results are not necessarily prohibitive but they provide substantial evidence of hardness for this basic measure for which no polynomial time search algorithm has been developed even after years of extensive research.

Due to apparent hardness of exact similarity search, recent algorithm development effort has focused on approximate similarity search. Given a query point Q, the approximate NN (ANN) problem, for example, aims to return a point A_j whose distance to Q is within a small multiplicative factor from that between Q and its exact nearest neighbor. The approximate FN (AFN) problem can be defined similarly.

Recent results on Hamming space and a few other normed spaces have polynomial time *randomized* solutions to ANN problem, as will be described below. Nevertheless ANN (or AFN) problems on most metric spaces of interest and all non-metric spaces have resisted the attempts for developing an efficient solution. The situation is especially dire for string similarity search problem which is fundamental to computational biology: After many years of extensive research, the most popular methods (e.g. BLAST) for string similarity search are still based on exhaustive search.

In this paper we provide further evidence on the hardness of ANN (and AFN) for string spaces. In particular, we show that ANN and AFN problems for string spaces (as well as other combinatorial spaces) is as hard as exact NN for Hamming space. (Observe that ENN and EFN for binary vectors under Hamming distance are equivalent problems: simply complement the query sequence to reduce one into the other.)

Below we first summarize known positive and negative results on similarity search problems. Then we describe our contributions in more detail.

Positive results on approximate similarity search. Much of the recent effort in ANN problem has been concentrated on normed spaces such as the Hamming space, l_k and l_∞.

Let p be the number of points in the data set and n be the number of dimensions. In [16] randomized algorithms for $(1+\epsilon)$-**ANN** problem are described for three domains: Hamming distance, l_1 and l_2. All of them have a preprocessing time polynomialin p and n and exponential in $1/\epsilon^2$; the search time is polynomial in n and $1/\epsilon$ and polylogarithmic in p. Similar results for Hamming distance and the l_1 norm were given in [13] where it is also shown that, for a large class of distance measures, an efficient algorithm for ϵ-**PLEB** (Point Location in Equal Balls) implies the existence of an efficient ANN algorithm.

Other known positive results for normed spaces can be summarized as follows. For l_∞ norm a $(4 \log \log 4n+1)$-**ANN** algorithm with preprocessing time $O(n\, p\, polylog\, p)$, and query time $O(n\ polylog\ p)$, is obtained in [10] via a reduction into ϵ-**PLEB** problem. An efficient solution to the ϵ-**PLEB** is also described there. Later, for l_2 norm, an $O(n\, polylog\, p)$-**time ANN** algorithm was obtained through a reduction to l_∞ via random projections [12]. For other metric spaces, an efficient solution to a variant of Hausdorff metric (only sets of finite size are considered) is given in [8] via a reduction to l_∞. For Frechet metric, which measures the distance between sequences of points, an efficient algorithm was provided in [11] by reduction to a product of smaller metric spaces.

Hardness results on approximate similarity search. There are few general hardness results and lower bounds for ANN problems, and most of them deal with norms and metrics: A lemma due to Bourgain [3] states that, for all n, there exist n-point metric spaces that cannot be reduced to l_2 with distortion smaller than $c\frac{\log n}{\log \log n}$. It is also known [17] that an expander cannot be reduced to l_2 with distortion smaller than $c \log n$. These are not necessarily negative results, since logarithmic distortion with n may still be acceptable for many applications.

As for the ANN problem under Hamming distance, (for which efficient randomized solutions are described above) the interesting question is how efficient can such solutions be made: In [4] it was shown that any deterministic constant factor ANN data structure for Hamming distance requires a query time of $\Omega(\log \log n / \log \log \log n)$ in the cell probe model. More recently it was shown in [5] that, any randomized algorithm that uses polynomial storage and word size $d^{O(1)}$ requires a worst case query time of $\Omega(\frac{\log \log d}{\log \log \log d})$. Here, the approximation factor is $2^{\log^{1-\epsilon} d}$, for any fixed $\epsilon > 0$. An even stronger lower bound is given in [18] which shows that any deterministic algorithm that uses a polynomial number of cells, with maximum cell size of $(d \log n)^{O(1)}$, must make $\Omega(\frac{d}{c^2 \log n})$ probes, for any approximation factor $c \leq \frac{\sqrt{d}}{20}$. These lower bounds are all based on the Richness Technique developed in [19,20].

Almost nothing is known about relative hardness of approximate similarity search for strings or other combinatorial objects. The only result of interest on strings states that Levenshtein edit distance can not be embedded into l_2 within a distortion of 3/2 or less [12].

Our Contributions. This paper provides the first study on the hardness of similarity search for strings and other combinatorial objects. For strings we consider two string similarity search problems which are fundamental to combinatorial pattern matching and computational biology: given a set of strings and an on-line query string, (1) find the string that has the smallest Levenshtein edit distance to the query (NN), (2) find the string that has the longest common subsequence with the query (FN). These two problems are equivalent for exact queries and for the original form of Levenshtein edit distance, ℓ, between two strings x and y, defined as the minimum number of character insertions and deletions to convert one string into the other: $LCS(x, y) = (|x| + |y| - \ell)/2)$ [9]. However they may lead to very different answers when a small factor of approximation is tolerated.

Our first result shows that the existence of a deterministic AFN (or ANN) algorithm for LCS within *one* constant factor approximation implies the existence of a deterministic ANN (equivalently AFN) algorithm for Hamming Distance within *all* constant factors of approximation. Because after years of study, no efficient deterministic AFN or ANN algorithm has been found for Hamming Distance within *any* constant factor, we conclude that AFN or ANN under LCS is (likely to be) hard to approximate within a constant factor. We also show that the existence of an AFN algorithm for LCS within *one* constant factor would imply AFN algorithms for both LCS and Set Intersection (SI) within *all* constant factors. Again, no constant factor AFN algorithm is known for these important problems, strengthening our evidence. Similar results can be established for sublogarithmic factors of approximation as well.

We then show that for any $0 < c < 1$, $(1 + 1/n^c)$-**ANN** problem under Levenshtein edit distance (as well as $(1 + 1/n^c)$-**AFN** problem under LCS) is at least as hard as ENN (equivalently EFN) under Hamming distance - whose hardness has been described by the cell probe lower bounds above.

We also provide relative hardness results for other combinatorial objects such as sets (under set intersection), permutations (under permutation edit distance) and labeled graphs (under a new difference vertex cover (DVC) measure we introduce here). Many of our results hold even if, in addition to multiplicative error, one also allows a polylogarithmic size additive error of approximation.

We obtain most of our results via "oblivious reductions" between problems of interest which are establihed in section 2. Based on these reductions, we establish a complexity hierarchy which is summarized in figure 1.

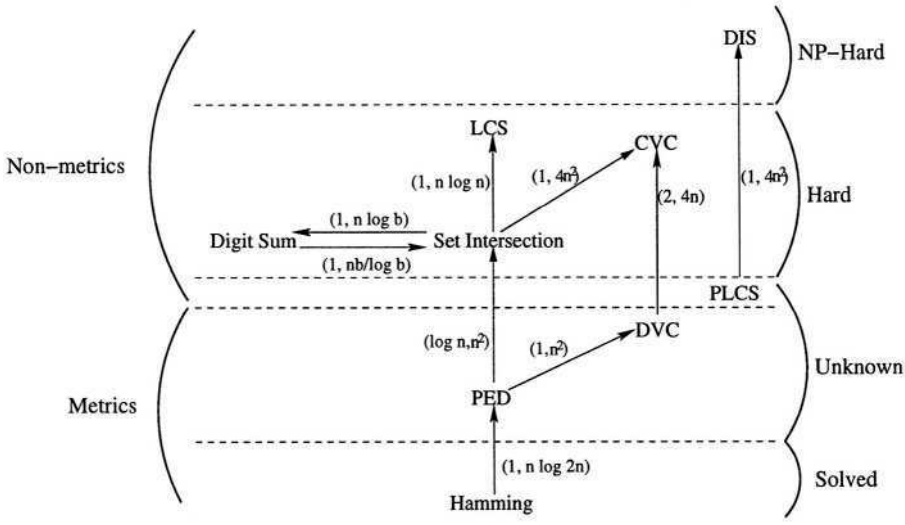


Fig. 1. Complexity hierarchy of ANN problems. Legend. PED:permutation edit distance, LCS: longest common subsequence, PLCS: permutation LCS, DIS: difference independent set, DVC: difference vertex cover, CVC: common vertex cover. See Section 3 for definitions and the proofs.

One final contribution is that, based on the insight provided by our reductions, we obtain a deterministic linear time 2-approximation algorithm for the permutation edit distance problem [7], improving on the running time of the best known exact algorithm by a factor of $\log n$ (see Section 3).

2 Reducibility Among Distance Measuring Problems

Our results on the reducibility of ANN and AFN problems are based on the relationships among problems of measuring distances between data points in the context of communication complexity. In the remainder of the paper we focus on ANN problems; generalizations to AFN problems are quite simple.

In this section we focus on the reducibility of distance measuring problems which can all be described within the following framework (Although this framework has been described in earlier work -with slight modifications- we still include it in the paper for completeness.) We have two parties, Alice and Bob, with respective data points a and b from a given universe U_1, and given a distance D_1 between data points in U_1; they would like to compute $D_1(a,b)$ by exchanging fewest number of bits possible. We use the following framework for reducing such a problem P_1 to another problem P_2 in which the goal again is to have Alice and Bob communicate the fewest number of bits to compute the distance $D_2(a',b')$ where a', b' are data points from another universe U_2 on which a distance D_2 is defined. The reductions allow approximations; i.e. P_1 or P_2 may ask Alice and Bob to compute the distances D_1 or D_2 within small factors of approximation. The reductions work through two mappings f_A and f_B used by Alice and Bob respectively, where $a' = f_A(a)$ and $b' = f_B(b)$. Our framework can be formalized as follows:

Definition 1. *Let D_1 and D_2 be distance measures on two given universes U_1 and U_2, respectively. Given two non-decreasing functions $k, s : \Re \to \Re$, we define a $[k(), s()]$-**oblivious** reduction from D_1 to D_2 as a pair of mappings $f_A, f_B : U_1 \to U_2$, with the following three properties:*
1) Let $|a|$ and $|b|$ be the number of bits to represent a and b respectively. Then for all $a, b \in U_1$ such that $\max\{|a|,|b|\} = n$, $k(n)\, D_1(a,b) \geq D_2(f_A(a), f_B(b)) \geq D_1(a,b)$. Thus $k()$ quantifies the distortion in approximating D_1 by D_2.
2) The number of bits needed to represent the mappings $f_A(a)$ and $f_B(b)$ are upper bounded by $s(|a|)$ and $s(|b|)$ respectively; i.e. $|f_A(a)| \leq s(|a|)$ and $|f_B(b)| \leq s(|b|)$.
3) The mappings $f_A(a)$ and $f_B(b)$ are computable in time $O(poly(|a|))$ and $O(poly(|b|))$ respectively.

The notion of an oblivious reduction leads us to the following relationships on the communication complexity of distance estimation in U_1 and U_2.

Lemma 1. *Suppose D_1 is $[k(), s()]$-**obliviously** reducible to D_2. Among all data points $a', b' \in U_2$ such that $\max\{|a'|,|b'|\} = m$, let the maximum number of bits Alice and Bob need to exchange to exactly compute $D_2(a',b')$ via a probabilistic protocol be $CC_2(m)$. Then the number of bits Alice and Bob need to exchange to compute a $k(n)$ factor approximation to $D_1(a,b)$ such that $n = \max\{|a|,|b|\} \leq s^{-1}(m)$ is at most $CC_2(s(n))$.*

Proof. Here is a protocol for Alice and Bob to exchange at most $CC_2(s(n))$ bits to compute a $k(n)$ factor approximation $D_1(a,b)$. Let $a' = f_A(a)$ and $b' = f_B(b)$; clearly $|a'| \leq s(n)$, and $|b'| \leq s(n)$. Alice and Bob can compute $D_2(f_A(a), f_B(b))$, by exchanging $CC_2(m)$ bits where $m = \max\{f_A(a), f_B(b)\}$. However D_1 is $[k(), s()]$-**obliviously** reducible to D_2; thus $m \leq s(n)$, and $D_2(f_A(a), f_B(b))$ is a $k(n)$ factor approximation to $D_1(a,b)$, implying the correctness of the lemma. ∎

Corollary 1. *Suppose D_1 is $[k(), s()]$-**obliviously** reducible to D_2, and let $c : \Re \to \Re$ be a non-decreasing function. Among all data points $a', b' \in U_2$ such that $\max\{|a'|, |b'|\} = m$, let the maximum number of bits Alice and Bob need to exchange to compute a $c(m)$ factor approximation to $D_2(a', b')$ via a probabilistic protocol be $CC_2(m)$. Then the number of bits Alice and Bob need to exchange to compute a $c(s(n)) \cdot k(n)$ factor approximation to $D_1(a,b)$ such that $n = \max\{|a|, |b|\} \leq s^{-1}(m)$ is at most $CC_2(s(n))$.*

Similarly, among all data points $a, b \in U_1$ such that $\max\{|a|, |b|\} = n$, let the maximum number of bits Alice and Bob need to exchange to compute a $c(n)$ factor approximation to $D_1(a,b)$ via a probabilistic protocol be $CC_1(n)$. Then, for any probabilistic protocol, there exists a pair $a', b' \in U_2$ such that $\max\{|a'|, |b'|\} \leq s(n)$, for which the number of bits Alice and Bob need to exchange to compute a $\frac{c(n)}{k(n)}$ factor approximation to $D_2(a', b')$ is at least $CC_1(n)$; i.e., the communication complexity of computing a $\frac{c(n)}{k(n)}$ approximation to $D_2(a', b')$ is at least $CC_1(n)$.

2.1 Reducibility Among ANN (and AFN) Problems

Oblivious reductions for distance measurement problems in the communication complexity context impose limitations on the efficiency of nearest and furthest neighbor search data structures. Given data points $a_1, a_2, \ldots, a_p \in U$, and a distance measure $D(.,.)$, a nearest neighbor search data structure should return for any on-line query point q, its "nearest" data point, i.e. it should return a_i for which $D(a_i, q) \leq D(a_j, q)$ for all a_j. An approximate nearest neighbor (ANN) search data structure relaxes this constraint by allowing some small factor of approximation in the answers returned for a query q. The relationship between the ANN-search and communication complexity problems (which have been established in earlier works starting with [20]) provide interesting lower bounds on the efficiency of ANN-search in certain domains of interest. A more formal framework for analyzing ANN-search problems is as follows.

Definition 2. *Let $a_1, \ldots, a_p, b \in U$, and let $n = max\{|a_1|, \ldots, |a_p|, |b|\}$. Let D be a distance measure on U. Then a $k(n)$-**Approximate** Nearest Neighbor (denoted by $k()$-**ANN(n)**) is an element a_i such that $D(a_i, b) \leq k \cdot D(a_j, b)$, for all $1 \leq j \leq p$. A $k(n)$-**ANN(n)** algorithm is called "acceptable", if it succeeds with some probability at least $\frac{2}{3}$, with pre-processing time $O(poly(n,p))$ and query time $O(poly(n, \log p))$.*

Lemma 2. *Suppose D_1 is $[k(), s()]$-**oblivious** reducible to D_2, with $s = O(poly(n))$. Then there exists an acceptable 1-**ANN(s(n))** algorithm for D_2 only if there exists an acceptable $k(n)$-**ANN(n)** algorithm for D_1.*

Proof. Suppose there exists an acceptable 1-ANN algorithm for D_2. This means that given $a_1, \ldots, a_p \in U_1$, we should be able to compute the values of and pre-process

$f_A(a_1), ..., f_A(a_p) \in U_2$ in $O(poly(s(n), p)) = O(poly(n, p))$ time so that it becomes possible to compute for any $b \in U_1$, a1-ANN for $f_B(b) \in U_2$ in $O(poly(s(n), \log p)) = O(poly(n, \log p))$ time.

Let $f_A(\hat{N}_b)$ be the 1-ANN of $f_B(b)$ in U_2 (which would be returned by the above algorithm for D_2) and let N_b be the nearest neighbor of b in U_1. Then:

$$D_1(\hat{N}_b, b) \leq D_2(f_A(\hat{N}_b), f_B(b)) \text{ as } D_2 \text{ is lower bounded by } D_1 \quad (1)$$

$$\leq D_2(f_A(N_b), f_B(b)) \text{ as } f_A(\hat{N}_b) \text{ is the nearest neighbor of } f_B(b) \quad (2)$$

$$\leq k(n) \cdot D_1(N_b, b) \quad \text{as } D_2 \text{ is upper bounded by } k(n) \cdot D_1. \quad (3)$$

Thus, the proof follows. ∎

Corollary 2. *Suppose D_1 is $[k(), s()]$-**oblivious** reducible to D_2, with $s = O(poly(n))$, and let $c()$ be a non-decreasing function. Then there exists an acceptable $\frac{c(s^{-1}(m))}{k(s^{-1}(m))}$-ANN(m) algorithm for D_2 only if there exists an acceptable $c(n)$-ANN(n) algorithm for D_1.*

The main purpose of the lemma above is to show the non-existence of efficient ANN algorithms for certain problems, thus it is more useful in its contrapositive form:

Corollary 3. *Suppose D_1 is $[k(), s()]$-**oblivious** reducible to D_2, with $s = O(poly(n))$, and let $c()$ be a non-decreasing function. Then, if there does not exist an acceptable $c(n)$-ANN(n) algorithm for D_1, there can not exist an acceptable $\frac{c(s^{-1}(m))}{k(s^{-1}(m))}$-ANN(m) algorithm for D_2.*

3 Relative Hardness of ANN/AFN Under LCS and Other Measures

In this section we describe an oblivious reduction from Hamming space to string space under LCS. Some of our intermediate steps and further results establish the complexity hierarchy among ANN problems summarized in Figure 1.

Theorem 1. *Hamming distance on $\{0,1\}^n$ is $(\log n, O(n^2 \log n))$-**oblivious** reducible to LCS.*

Proof. We establish the results via several intermediate oblivious reductions involving permutation edit distance and set intersection.

Definition 3. *Given two permutations a, b of an alphabet σ, their permutation edit distance, $PED(a, b)$, is the minimum number of character move operations needed to transform a into b (or vice versa).*

Lemma 3. *Hamming distance on $\{0,1\}^n$ is $(1, n \log 2n)$ - **oblivious** reducible to permutation edit distance.*

Proof. Let $\sigma = \{q_i, r_i : 1 \leq i \leq n\}$. Given $x = x_1 x_2 \ldots x_n$, let $f_A(x_i) = f_B(x_i) = q_i, r_i$ if $x_i = 0$, and r_i, q_i otherwise; $f_A(x) = f_B(x)$ is the concatenation of $f_A(x_1), \ldots, f_A(x_n)$. Then $k(n) = 1$ as $PED(f_A(x), f_B(y)) = h(x, y)$, because for each $x_i \neq y_i$ exactly one swap (say, from q_i, r_i to r_i, q_i) is needed between $f_A(x)$ and $f_B(y)$. The lemma follows as $s(n) = |f_A(x)| = |f_B(y)| = n \log 2n$. ∎

Definition 4. *Given two vectors $a, b \in \{0,1\}^n$, their set intersection is $SI(a,b) = a \cdot b$, i.e. their dot product.*

Lemma 4. *Permutation edit distance is* $(\log n, n^2)$ - ***oblivious*** *reducible to set intersection.*

Proof. Given a, b permutations of σ, we construct two matrices $f_A(a), f_B(b)$ as follows: let the rows and the columns of both matrices be labeled by the identity permutation of σ. Then $f_A(a)[x, y] = 1$ whenever the character x occurs before y in a, and 0 otherwise. Meanwhile, $f_B(b)[x, y] = 1$ whenever x occurs after y in b, and the distance between x and y is a power of 2. Clearly, $s(n) = n^2$. For proof that $k(n) = \log n$ see [7]. ∎

Definition 5. *Digit Sum, for parameters $b, k \in \mathcal{N}$, between any pair of data points $a, b \in U = \{0, 1, \ldots, b-1\}^d$, is defined to be, $DS[b,k](a,b) = |\{i : a_i + b_i = k\}|$*

Remark 1. We argue that *DS*[b,k] should be considered only for b > 1, and $b - 1 \leq k \leq 2b - 2$. Indeed, if $b \leq 1$, the problem is trivial as every digit is 0. If k < b – 1, then the digits between k + 1 and b – 1 will never add up to k, thus we could as well consider *DS*[k+1, k]. If k > 2b – 2, then no two digits will add up to k and the problem is trivial.

Definition 6. *Given a set $a' \in \{u_1, \ldots, u_d\}$, we define its binary vector representation a such that $a_i = 1 \iff u_i \in a'$.*

Example 1. $DS[2,1](a,b)$ is equal to Hamming distance between two binary vectors a, b. Similarly, $DS[2,2](a,b)$ is equal to intersection of two sets $a', b' \subset \{u_1, \ldots, u_d\}$ whose vector representations are a, b respectively.

Lemma 5. *Set intersection is* $(1, n \log b)$-***oblivious*** *reducible to DS[b,k], except when $b = 2$ and $k = 1$.*

Proof. Let $t = \lfloor \frac{k}{2} \rfloor$. Given two sets $a', b' \subset \{u_1, \ldots, u_d\}$, let a, b be their vector representations. Also let $f_A(a_i) = a_i \cdot t$, and $f_B(b_i) = b_i \cdot (k - t)$. It's easy to see that as long as $t > 0$, $f_A(a_i) + f_B(b_i) = k$ if and only if $a_i = b_i = 1$; thus $k = 1$ For any $x \in \{u_1, \ldots, u_d\}$, observe that $|f_A(x)| = |f_B(x)| \leq |x| \cdot \log b$, since each digit can be stored by log b bits; thus $s = n \log b$. ∎

Lemma 6. *$DS[b,k]$ is $(1, n \cdot \frac{b}{\log b})$-**oblivious** reducible to Set Intersection.*

Proof. Given two vectors $a, b \in \{0, \ldots, \mathbf{b}-1\}^d$, let $U = \{u_{i,j} : 1 \leq i \leq d, 0 \leq j \leq \mathbf{b}-1\}$. Let $f_A(a) = \{u_{i,j} : a_i = j\}$, and $f_B(b) = \{u_{i,j} : b_i = \mathbf{k} - j\}$. Then $u_{i,j} \in f_A(a) \cap f_B(b)$ if and only if $a_i = j$ and $b_i = \mathbf{k} - j$, i.e. $a_i + b_i = \mathbf{k}$, thus this reduction gives $k = 1$. Since $n = |a| = d \cdot \log \mathbf{b}$, $s(n) = |f_A(a)| = d \cdot \mathbf{b} = n \cdot \frac{\mathbf{b}}{\log \mathbf{b}}$. ∎

Remark 2. The preceding two lemmas show that, in terms of communication complexity and complexity of finding approximate nearest neighbors, Set Intersection and Digit Sum are equally hard.

Lemma 7. *DS[b,k] is* $(1, \frac{n}{\log b} \log \frac{n}{\log b})$-***oblivious*** *reducible to Longest Common Subsequence.*

Proof. Let $\sigma = \{c_{i,j} : 1 \leq i \leq d, 0 \leq j \leq \mathbf{b}-1\}$. For any vector $x \in \{0, \ldots, \mathbf{b}-1\}^d$, we define $f_A(x_i) = c_{i,x_i}$, and $f_B(x_i) = c_{i,\mathbf{k}-x_i}$, and $f_A(x)$ and $f_B(x)$ as the concatenation of $f_A(x_1), \ldots, f_A(x_d)$, and $f_B(x_1), \ldots, f_B(x_d)$, respectively. Then given vectors a, b, the size of the LCS of $f_A(a)$ and $f_B(b)$ is exactly the number of coordinates in which $a_i + b_i = \mathbf{k}$; thus $k = 1$. Also, for any vector x, notice that $|f_A(x)| = |f_B(x)| \leq |x| \cdot \log d$, since there is an increase in the size of the alphabet by a factor of d; thus $s = d \log d$ and because $n = d \cdot \log \mathbf{b}$, we can write $s = \frac{n}{\log \mathbf{b}} \log \frac{n}{\log \mathbf{b}}$. ∎

Corollary 4. *Set Intersection is* $(1, n \log n)$-***oblivious*** *reducible to Longest Common Subsequence.*

This completes the proof of the theorem 1. ∎

3.1 The Complexity Hierarchy of ANN Problems

In this section we complete the complexity hierarchy of Figure 1. While deriving this hierarchy we also provide answers to two interesting problems:
(1) The communication complexity of LCS is $\Omega(\frac{n}{\log n})$.
(2) There is a linear time deterministic 2-approximation algorithm for computing permutation edit distance.

We start with the observation that Lemma 7 implies a lower bound on the (probabilistic) communication of complexity of LCS.

Lemma 8. *The (probabilistic) communication complexity of LCS is* $\Omega(\frac{n}{\log n})$.

Proof. The (probabilistic) communication complexity of set intersection is $\Theta(n)$ [15]. Proof follows from Lemmas 7 and 1. ∎

We now provide two similarity measures for labeled graphs and establish the hardness of ANN problems under both measures.

Definition 7. *Let* G_a, G_b *be graphs on* n *vertices labeled by numbers* 1 *through* n. *Then the Difference Graph of* G_a *and* G_b, *denoted* $GD(G_a, G_b)$ *is a graph on* n *vertices, such that an edge* e *is in GD if and only if it occurs in exactly one of* G_a *or* G_b.

Definition 8. *Let G_a, G_b be graphs on n vertices. Then the Difference Vertex Cover of G_a and G_b, denoted $DVC(G_a, G_b)$, is the size of the minimum vertex cover of $GD(G_a, G_b)$ (a vertex cover is a set of vertices such that each edge in the graph is touched by some vertex in this set).*

Lemma 9. *Difference Vertex Cover is a metric.*

Proof. Let G_1, G_2, G_3 be graphs on n vertices labeled by numbers 1 through n. Note that $DVC(G_1, G_2) = DVC(G_2, G_1)$ because of the symmetry in the definition. Let $G_{1,2} = GD(G_1, G_2)$, and $G_{2,3} = GD(G_2, G_3)$. Then notice that $G_{1,3} = GD(G_1, G_3) = GD(G_{1,2}, G_{2,3})$. Let $S_{1,2}$, $S_{2,3}$, and $S_{1,3}$ be minimum vertex covers of $G_{1,2}, G_{2,3}$, and $G_{1,3}$, respectively. Clearly, $S_{1,2} \bigcup S_{2,3}$ is a vertex cover of $G_{1,3}$, since, by definition, every edge of $G_{1,3}$ is either in $G_{1,2}$ or in $G_{2,3}$. Thus, $|S_{1,3}| \leq |S_{1,2}| + |S_{2,3}|$, and the triangular inequality holds. In addition, note that $DVC(G_1, G_1) = 0$, since $GD(G_1, G_1)$ has no edges (however, this is not required of a metric). ∎

Lemma 10. *Permutation edit distance is $(1, n^2)$-**oblivious** reducible to DVC.*

Proof. Given permutation x of $\{1, \ldots, n\}$, we construct $f_A(x) = f_B(x)$ as graph G_x as follows: label the vertices of G_x by $\{1, \ldots, n\}$. We construct G_x by connecting vertices $i < j$ with an edge if and only if character i occurs before character j in x. Given a, b as permutations of $\{1, \ldots, n\}$, it's clear that the difference graph *GD* of $f_A(a)$ and $f_B(b)$ has an edge (i, j) if and only if characters i and j occur in different order in permutations a and b. A vertex cover of *GD* represents a set of vertices which corresponds to characters, such that the removal of these characters from both a and b will make a and b identical. Thus, the minimum vertex cover is the minimum number of characters we need to move in order to make a identical to b, which, by definition, is equal to $PED(a, b)$. ∎

Interestingly the above oblivious reduction provides a linear time 2-approximation algorithm for PED.

Corollary 5. *There exists a linear time 2-approximation algorithm for PED.*

Proof. Let a, b be permutations of $\{1, ..., n\}$. The 2-approximation algorithm to compute the PED between a and b is based on the reduction to vertex cover problem as described above. It uses the observation that given any pair of vertices on a graph G which are connected by an edge, at least one of the vertices must be a part of the vertex cover as follows.

Scan the characters of a and b from left to right. While doing this , try to mark characters in a that must be moved to transform it to b. If any character is marked once it can be assumed to be in its correct location; there is no need for marking it twice. The marking is performed by considering at each step a particular edge of the "difference graph", namely the edge between the leftmost "unprocessed" characters of a and b as follows.

Set $i = j = 1$
While $i \leq n$ and $j \leq n$:

1. if $a_i = b_j$, mark a_i and increment i and j by 1
2. else if a_i is already marked, increment i by 1
3. else if b_j is already marked, increment j by 1
4. else increase the size of PED by 2; also mark both a_i and b_j, and increment both i and j by 1.

The algorithm clearly runs in $O(n)$ time. ∎

Definition 9. *Let G_a, G_b be graphs on n labeled vertices. Then the Largest Common Subgraph of G_a and G_b is a graph $GC(G_a, G_b)$ on the same n vertices such that an edge e is in $GC(G_a, G_b)$ if and only if it occurs in both G_a and G_b.*

Definition 10. *Let G_a, G_b be graphs on n labeled vertices. Then the Common Vertex Cover of G_a and G_b, denoted $CVC(G_a, G_b)$, is the size of the minimum vertex cover of the largest common subgraph of G_a and G_b.*

Lemma 11. *DVC is $(2, 4n)$-**oblivious** reducible to CVC.*

Proof. Let G_a, G_b be vertex labeled graphs with vertices $\{1, \ldots, n\}$. Let G_a^C and G_b^C be the complements of G_a and G_b respectively. Let G'_a be the graph derived from G_a by renaming each vertex i as $n + i$ and complementing the resulting graph. Let G'_b be the graph derived from G_b by renaming each vertex i as $n + i$. Then we set $f_A(G_a) = G_a \bigcup G'_a$, and $f_B(G_b) = G_b^C \bigcup G'_b$. Notice that every edge of $GD(G_a, G_b)$ is either in $GC(G_a, G_b^C)$ or $GC(G_a^C, G_b)$. Any vertex cover of $GD(G_a, G_b)$ will give vertex covers for both "halves" of $GC(f_A(G_a), f_B(G_b))$, providing a full vertex cover on $GC(f_A(G_a), f_B(G_b))$. Conversely, any vertex cover of $GC(f_A(G_a), f_B(G_b))$ cannot be smaller than $DVC(G_a, G_b)$, since it induces a vertex cover of $GD(G_a, G_b)$. ∎

Lemma 12. *Set intersection is $(1, 4n^2)$-**oblivious** reducible to CVC.*

Proof. Let a, b be n dimensional binary vectors. Let $f_A(a) = f_B(a)$ be a graph on $2n$ vertices with the following property: edge (i, j) is in the graph if and only if $j = i + n$ and $a_i = 1$. Then $LCG(f_A(a), f_B(b))$ will have an edge (i, j) if and only if $j = i + n$ and $a_i = b_i = 1$. Thus, the minimum vertex cover of this graph equals the intersection of the sets represented as binary vectors a and b. ∎

Corollary 6. *CVC is not a metric.*

Proof. Just notice that set intersection is reducible to CVC with no distortion. ∎

We further establish the hardness of LCS via the following lemma.

Theorem 2. *If there is an acceptable **k-ANN** algorithm for LCS for some $k = O(1)$, then there is an acceptable **k-ANN** algorithm for LCS for every $k = O(1)$.*

Proof.

Definition 11. *Given two permutations a, b of an alphabet σ, the permutation longest common subsequence $PLCS(a,b)$ is the maximum size of an ordered subset c such that c is a subsequence of both a and b.*

Definition 12. *Let G_a, G_b be graphs on n vertices. Then the Difference Independent Set of G_a and G_b, denoted $DIS(G_a, G_b)$, is the size of the maximum independent set of $GD(G_a, G_b)$.*

Lemma 13. *If there is an acceptable k-ANN algorithm for PLCS for some $k = O(1)$, then there is an acceptable k-ANN algorithm for PLCS for every $k = O(1)$.*

Proof. Let $\pi(\Sigma)$ denote the set of permutations of Σ. We give a function $f : \pi(\Sigma) -> \pi(\Sigma^2)$ such that $PLCS(f(X), f(Y)) = PLCS(X,Y)^2$, and f is poly-time computable. If this can be done, then clearly we can apply the k-ANN algorithm to $f(Q)$ and $f(A_1)...f(A_p)$, thus obtaining a $\sqrt{k}$-ANN algorithm for $Q, A_1, ..., A_p$.

We define f on a permutation $X = c_1...c_n$ of Σ to be $(c_1,c_1)(c_1,c_2)...(c_1,c_n)(c_2,c_1)...(c_2,c_n)...(c_n,c_1)...(c_n,c_n)$. In other words, it is a concatenation of all the "products" $c_1 \cdot X ... c_n \cdot X$, where $c_i \cdot X$ is $(c_i,c_1)...(c_i,c_n)$. Clearly, $f(X)$ is a permutation of Σ^2. Suppose X, Y have a common subsequence $Z = c_{i_1}...c_{i_k}$. Then $f(Z)$ is a common subsequence of $f(X), f(Y)$. Suppose $f(X), f(Y)$ have a common subsequence $(c_{i_1}, c_{j_1})...(c_{i_{k^2}}, c_{j_{k^2}})$. Then both $c_{i_1}...c_{i_{k^2}}$ and $c_{j_1}...c_{j_{k^2}}$, and at least one of the sequences must contain k distinct elements. So X, Y have a common subsequence of length k. ∎

This completes the proof of the theorem. ∎

Lemma 14. *PLCS is $(1, n^2)$-oblivious reducible to DIS.*

Proof. Given permutation x of $\{1, \ldots, n\}$, we construct $f_A(x) = f_B(x)$ as graph G_x as follows: label the vertices of G_x by $\{1, \ldots, n\}$. We construct G_x by connecting vertices $i < j$ with an edge if and only if character i occurs before character j in x. Given a, b as permutations of $\{1, \ldots, n\}$, it's clear that the difference graph GD of $f_A(a)$ and $f_B(b)$ has an edge (i,j) if and only if characters i and j occur in different order in permutations a and b. An independent set of GD represents a set of vertices which correspond to characters, such that every pair of characters within the set occurs in the same order in a and b. Thus, each independent set gives a common subsequence. On the other hand, each common subsequence of a and b has a corresponding independent set in GD, since no two characters within the subsequence can have an edge between them. Thus, the size of the maximum independent set of GD is the same as the size of the longest common subsequence of a and b. ∎

Remark 3. It is trivial to see that an acceptable $O(1)$-ANN algorithm for DIS exists only if NP=P, since the maximum independent set problem has a constant approximation algorithm only if NP=P. Because of this, DIS is not an interesting problem. However, we include it in this paper to give a better perspective of the whole complexity hierarchy of ANN problems.

4 Hardness of k-ANN Under Set Intersection, LCS, and Other Problems

In this section, we show that solving the k-ANN problem under set intersection, for any $k > 1$, is as hard as solving the 1-NN problem (i.e. the exact nearest neighbor) under set intersection. Thus, this problem is as hard as exact nearest neighbors under Hamming distance. This fact implies that an acceptable k-ANN algorithm for set intersection, as well as LCS, digit sum and CVC are difficult due to the lower bounds of [2,1,14] apply.

We also show that the above results hold even if, in addition to multiplicative error, one also allows a polylogarithmic size additive error of approximation. We finally show that an acceptable $O(1)$-ANN algorithm is possible for set intersection if and only if k-ANN algorithm are possible for all $k = O(1)$ and for all distance measures that are reducible to set intersection. We begin by introducing the following geometric problem.

Definition 13. *Let S be a set of real numbers. Then orthogonality problem on S, denoted OQ_S can be defined as follows. Given a set of vectors $A_1, ..., A_p \in S^n$, build a data structure in time $O(poly(n, p))$, such that, given a query vector $Q \in S^n$, do the following in time $O(poly(n, \log p))$:*
if there exists $1 \leq i \leq p$ such that $A_i \cdot Q = 0$ (dot product of two vectors) then return one A_j for which $A_j \cdot Q = 0$. If no such i exists, then return an arbitrary A_j.

Lemma 15. *The $OQ_{\{-1,0,1\}}$ problem is solvable only if there exists an acceptable exact nearest neighbors algorithm under set intersection.*

Proof. We prove the lemma by constructing an acceptable 1-NN algorithm for set intersection by the use of a solution to the $OQ_{\{-1,0,1\}}$ problem. Suppose we are given $A_1...., A_p \in \{0,1\}^n$. Construct $n+1$ data structures $Ds_0, \ldots, Ds_L, Ds_n$, for $OQ_{\{-1,0,1\}}$, for each $0 \leq L \leq n$, as follows: for a given L, let $A_{L,i} \in \{0,1\}^{n+L}$ be A_i concatenated with 1^L, and let Q_L be Q concatenated with -1^L.

Notice that if for a given A_i, we have $A_i \cdot Q = L$, then within data structure Ds_L, we must have $A_{L,i} \cdot Q_L = 0$. Thus, $min\{A_i \cdot Q\} = min\{L : A_{L,i} \cdot Q_L = 0\}$. We can thus find the exact nearest neighbor of Q by querying the data structures Ds_L in increasing order of L, and returning the first $A_{L,i}$ such that $A_{L,i} \cdot Q_L = 0$ will give A_i that minimizes $A_i \cdot Q$. ∎

Lemma 16. *The $OQ_{\{0,1\}}$ problem is solvable only if the $OQ_{\{-1,0,1\}}$ problem is solvable.*

Proof. We prove the lemma by producing a solution to the $OQ_{\{-1,0,1\}}$ problem through the use of a solution to the $OQ_{\{0,1\}}$ problem. Consider the function $f : \{-1,0,1\}^n \rightarrow \{-1,0,1\}^{n^2}$ defined as follows: given a vector $V = (v_1, ..., v_n) \in \{-1,0,1\}^n$, let $f(V)$ be the concatenation of $v_1 \cdot V, v_2 \cdot V, \ldots v_n \cdot V$ (here, $v_i \cdot V$ denotes multiplication of vector by a scalar).
We first observe that for any $X, Y \in \{-1,0,1\}^n$, $f(X) \cdot f(Y) = (X \cdot Y)^2$. This can be easily seen by letting $a = |\{i : x_i y_i = 1\}|$, and $b = |\{i : x_i y_i = -1\}|$. Then $X \cdot Y = a - b$. But $f(X) \cdot f(Y) = a(X \cdot Y) - b(X \cdot Y) = (X \cdot Y)^2$.

This implies that for any $X, Y \in \{-1, 0, 1\}^n$, $f(X) \cdot f(Y) = (X \cdot Y)^2 \geq 0$. Thus, for any such vectors X and Y, the angle between $f(X)$ and $f(Y)$ is at most 90 degrees. Notice that, whenever we have a finite set of vectors in R^{n^2} such that the angle between any pair of vectors is at most 90 degrees, then there must exist an n^2-dimensional cube with one of the vertices at the origin, such that it contains every vector in this set. (In other words, we can shift the coordinate axis so that each vector lie in the positive quadrant of the resulting n^2 dimensional space.)

Note that such a cube is only a function of n, and no other inputs and is uniquely described by n^2 pairwise orthogonal vectors. Let $C_1, ..., C_{n^2}$ be the vectors that describe such a cube. Then these vectors define an alternative orthogonal coordinate system. For any $X \in \{-1, 0, 1\}^n$, we can compute the coordinates of $f(X)$, call it $g(f(X))$, in this new coordinate system by computing the projections of $f(X)$ onto $C_1, ..., C_{n^2}$. Since the cube contains $f(X)$, $g(f(X))$ must have only non-negative coordinates. Consider the mapping $h : R^{n^2}_{\geq 0} \rightarrow \{0, 1\}^{n^2}$ defined as follows: let $V = (v_1, ..., v_{n^2}) \in R^{n^2}_{\geq 0}$, then $h(V) = V' = (v'_1, ..., v'_{n^2}) \in \{0, 1\}^{n^2}$ with $v'_i = 0$ whenever $v_i = 0$, and $v'_i = 1$ otherwise.

Let $U, V \in R^{n^2}_{\geq 0}$. Then we observe that $U \cdot V = 0$ if and only if $h(U) \cdot h(V) = 0$: this follows from the fact that U and V have only non-negative coordinates, and $U \cdot V = 0$ if and only if every positive coordinate gets multiplies by 0. Thus, the actual value of the positive coordinates does not matter. This concludes the proof of the lemma. ∎

Theorem 3. *For any k, there exists an acceptable k-ANN algorithm for set intersection if and only if there exists an acceptable exact nearest neighbor algorithm for set intersection.*

Proof. It is obvious that an exact nearest neighbor algorithm is also a k-ANN algorithm, thus we focus on the other direction.

Suppose there exists an acceptable k-ANN algorithm for set intersection. We construct an algorithm for $OQ_{\{0,1\}}$ as follows: given $A_1, ..., A_p \in \{0, 1\}^n$, we build the data structure for k-ANN search. Given a query $Q \in \{0, 1\}^n$, let A_j be the k-ANN of Q returned by the algorithm. Then, if there exists an A_i such that $A_i \cdot Q = 0$, we must have $A_j \cdot Q \leq k(A_i \cdot Q) = 0$, thus $A_j \cdot Q = 0$.

From Lemma 16 we know that an algorithm for $OQ_{\{-1,0,1\}}$ must exist, and thus from Lemma 15 we obtain an acceptable algorithm for exact nearest neighbors under set intersection. ∎

Now that we know that k-ANN problem is difficult for set intersection, a natural question is whether it is possible to obtain an efficient solution to the c + k-ANN problem for set intersection, where c is an additive approximation constant.

Definition 14. *Let U be a set, and let D be a distance measure on U. Given $A_1, ..., A_p, Q \in U$, the (k, c)-ANN problem asks to find A_i such that $D(A_i, Q) \leq k \cdot D(A_j, Q) + c$ for all A_j.*

Theorem 4. *For all k, and all $c = O(poly \log n)$, there exists an acceptable (k, c)-ANN algorithm for set intersection if and only if there exists an acceptable exact nearest neighbors algorithm for set intersection.*

Proof. Let $A_1, ..., A_p \in \{0,1\}^n$, and let $c = (\log n)^b$, $b = O(1)$. Suppose an acceptable (k, c)-ANN algorithm exists. We construct $A'_1, ..., A'_p \in \{0,1\}^{nc^2}$ by setting A'_i to be the concatenation of A_i with itself c^2 times. For a given query $Q \in \{0,1\}^n$, we construct Q' by concatenating Q with itself c^2 times. Suppose A'_i is the exact nearest neighbor of Q', and suppose the algorithm returns A'_j. Then $A'_j \cdot Q' = (\log n)^{2b}(A_j \cdot Q) \leq k(A'_i \cdot Q') + (\log(nc^2))^b = k(\log n)^{2b}(A_i \cdot Q) + (\log n + 2b \log\log n)^b$. We can divide the inequality by $(\log n)^{2b}$, and we get: $A_j \cdot Q \leq k(A_i \cdot Q) + \frac{(\log n + 2b \log\log n)^b}{(\log n)^{2b}} < k(A_i \cdot Q) + 1$. Thus, this solves the k-ANN problem. But from Theorem 3 we know that this implies a solution to the exact nearest neighbor problem.
The other direction is trivial. ∎

Corollary 7. *For any $k > 1$, and $c = O(polylog(n))$, the (k, c)-ANN problem under longest common subsequence, digit sum, and common vertex cover is harder than the exact nearest neighbor problem under set intersection.*

The preceding corollary has important implications about hardness of three very different groups of problems: digit sum (a vector problem), longest common subsequence (string similarity), and common vertex cover (graph similarity). Note that all of these problems are non-metrics. The relative complexity of metrics and non-metrics is illustrated in figure 1. The following lemma is parallel to inapproximability results in classical computational complexity.

Lemma 17. *Let $k = O(polylog(n))$. Then an acceptable k-ANN search algorithm for set intersection exists if and only if an acceptable $\sqrt{k}$-ANN search algorithm for set intersection exists.*

Proof. We employ the following reduction: suppose there exists an acceptable k-ANN algorithm for set intersection, and let $A_1, A_2, ..., A_p \in \{0,1\}^n$ be our input points. We construct a new set of points $A'_1, A'_2, ..., A'_p \in \{0,1\}^{n^2}$ as follows: for each A_i, we begin by setting A'_i to be an empty binary string. Then at step j, if the j-th bit of A_i is 1, we append a copy of A_i to A'_i. If the j-th bit of A_i is 0, we append n zeros to A'_i. For each query Q, we construct Q' in the same way. Notice that each intersection of A_i and Q contributes $|A_i \cap Q|$ to $|A'_i \cap Q'|$, and thus $|A'_i \cap Q'| = |A_i \cap Q|^2$. By assumption, there exists an acceptable k-ANN algorithm, thus we can build a data structure for $A'_1, ..., A'_n$ for k-ANN search. Suppose for a given Q', obtained from a given Q as above, the search returns A'_i. Then $|A'_i \cap Q'| \leq k \cdot |A'_j \cap Q'|$, which is equivalent to $|A_i \cap Q| \leq \sqrt{k} \cdot |A_j \cap Q|$. Thus, we obtain an acceptable $\sqrt{k}$-ANN algorithm by squaring the dimension. ∎

Corollary 8. *There exists an acceptable k-ANN algorithm for set intersection for some $k = \Theta(1)$ if and only if there exists an acceptable k-ANN algorithm for all $k = \Theta(1)$.*

Corollary 9. *There exists an acceptable k-ANN algorithm for set intersection for some $k = \Theta(poly(\log n))$ if and only if there exists an acceptable k-ANN algorithm for all $k = \Theta(poly(\log n))$.*

4.1 Hardness of Approximate Furthest Neighbors Problems

Longest common subsequence is an interesting measure of similarity between strings. The longer the common subsequence, the more similar two strings are. We have shown that the k-ANN problem under LCS is hard. In this section, we proceed to show that certain cases of the k-approximate furthest neighbor (k-AFN) are also hard. The hardness of k-AFN in LCS follows directly from the hardness of k-AFN under set intersection, and the observation that the reduction in Lemma 7 is exact. We begin by formally defining the k-AFN problem.

Definition 15. *Let U be a set, and D be a distance measure on U. Let $A_1, ..., A_p, Q \in U$. Then a* ***k-Approximate*** *Furthest Neighbor (denoted by* ***k-AFN****) is an element A_i such that $D(A_i, Q) \geq \frac{D(A_j,Q)}{k}$, for all $1 \leq j \leq p$.*

The following lemma shows that a better than constant approximate furthest neighbors algorithm under set intersection (and thus LCS) is hard.

Lemma 18. *Let $c > 0$ be fixed. Then there exists an acceptable $(1+\frac{1}{n^c})$-**AFN** algorithm for set intersection only if there exists an acceptable exact nearest neighbor algorithm for set intersection.*

Proof. Suppose there exists an acceptable $(1 + \frac{1}{n^c})$-**AFN** algorithm for set intersection. Let $A_1, ..., A_p, Q \in \{0,1\}^n$ be given. Let h be the smallest positive integer such that $\frac{1}{h+1} \leq c$ (for $c \geq 1$ the lemma is trivial), and let $m = n^{h+1}$. We construct $A'_1, ..., A'_p, Q' \in \{0,1\}^m$, where A'_i is A_i concatenated with itself n^h times. We construct Q' from Q in the same way. Note that $A_i \cdot Q = 0$ if and only if $A'_i \cdot Q' = 0$, and $A_i \cdot Q \geq 1$ if and only if $A'_i \cdot Q' \geq n^h = m^{\frac{h}{h+1}}$. Next, we obtain $A''_1, ..., A''_p, Q'' \in \{0,1\}^m$ by letting each bit of A''_i be the opposite of the corresponding bit in A'_i, i.e. we let A''_i be the complement of A'_i. We let $Q'' = Q'$. Observe that $A''_i \cdot Q'' = |Q''|$ if and only if $A'_i \cdot Q' = 0$; and $A''_i \cdot Q'' \leq |Q''| - m^{\frac{h}{h+1}}$ if and only if $A'_i \cdot Q' \geq m^{\frac{h}{h+1}}$, which happens if and only if $A_i \cdot Q \geq 1$. Thus, the minimum multiplicative discrepancy between the furthest neighbor of Q'' and another point is $\frac{|Q''|}{|Q''|-m^{\frac{h}{h+1}}} = (1 + \frac{m^{\frac{h}{h+1}}}{|Q''|-m^{\frac{h}{h+1}}}) > (1 + \frac{m^{\frac{h}{h+1}}}{m}) = (1 + \frac{1}{m^{\frac{1}{h+1}}})$. Thus, if we apply the acceptable $(1 + \frac{1}{m^c})$-**AFN** algorithm to $A''_1, ..., A''_p$ and Q'', then whenever there exists A_i such that $A_i \cdot Q = 0$, the algorithm will return A''_i (from which we can deduce the A_i). Therefore, this solves the $OQ_{\{0,1\}}$ problem. But Lemmas 15 and 16 show that this implies an acceptable solution to the exact nearest neighbor problem, thus the proof is complete. ∎

Lemma 19. *Suppose that for some $k = O(1)$, an acceptable* ***k-AFN*** *algorithm for set intersection exists. Then such an algorithm exists for all $k = O(1)$.*

Proof. The construction used in Lemma 17 proves this lemma as well. ∎

4.2 Hardness of ANN for Levenshtein Edit Distance

The Levenshtein edit distance between two strings x and y, denoted $LED(x, y)$ is defined as the minimum number of single character insertions and deletions to convert

one string to the other. One of our most interesting results is on the hardness of ANN under Levenshtein edit distance (with unbounded alphabet size).

Theorem 5. *For any $c > 0$, $(1+1/n^c)$-ANN under Levenshtein edit distance is as hard as ENN under Hamming Distance (and Set Intersection).*

Proof. Suppose we have an acceptable $(1 + 1/n^c)$-**ANN** algoritm for Levenshtein edit distance. Below, we show that this implies the existence of an acceptable $(1 + 1/n^{c'})$-**AFN** algorithm for LCS.

It is a well known fact that for any two strings x, y, $LCS(x,y) = (|x| + |y| - LED(x,y))/2$ [9]. Given a set of strings $X = \{x_1, \ldots, x_p\}$ and a query string q whose lengths are all n, suppose that $x_j = argmax_i LCS(q, x_i)$ and for all x_i we have $LCS(q, x_j) - LCS(q, x_i) \geq 2/3 \cdot n^{1-c'}$. In order to perform AFN queries on X under LCS we map X to a new set $X' = \{x'_1, \ldots, x'_p\}$ and q to q' such that x'_i (q') is the concatenation of x_i (q) with n βs where β is a new alphabet symbol. Clearly for any x'_i, $LCS(x'_i, q') = LCS(x_i, q) + n$. Let $c = c' + \log 3/2$. If we have an acceptable $(1 + 1/n^c)$-**ANN** algoritm for Levenshtein edit distance on X', given query q', this algorithm should return x_j as for any other x_i we have:

$$LED(q', x'_j) = 4 \cdot n - 2 \cdot LCS(q', x'_j) = 2 \cdot (n - LCS(q, x_j))$$

$$\leq 2 \cdot (n - LCS(q, x_i) - 2/3 \cdot n^{1-c'}) \leq 4 \cdot n - 2 \cdot LCS(q', x'_i) - 2/3 \cdot LED(q', x'_i) \cdot n^{-c'}$$

$$\leq LED(q', x'_i)(1 - 2/(3 \cdot n^{c'})) \leq LED(q', x_i)(1 - 1/n^c)$$

This implies $LED(q', x_i) \geq LED(q', x_j)(1 + 1/n^c)$ and proves our claim.

The existence of an acceptable $(1 + 1/n^{c'})$-**AFN** algorithm for LCS implies the existence of an acceptable $(1 + 1/n^{c'})$-**AFN** algorithm for SI (Corollary 4). This, in turn, implies the existence of an acceptable ENN algorithm for SI (Corollary 18) and thus an acceptable ENN algorithm for Hamming Distance (Lemma 4), which completes the proof of the theorem. ∎

4.3 Additional Evidence on Hardness of Set Intersection and Other Problems

In this section, we provide additional evidence that the k-**ANN** problem under set intersection (and other problems to which set intersection can be reduced) must be hard.

Lemma 20. *Let $c < 1$ be a constant, and $a, b \in \{0,1\}^n$ be vector representations of two sets a' and b'. Then there does not exist a communication protocol with complexity $O(polylog(n))$, which distinguishes between $|a' \cap b'| = 0$ and $|a' \cap b'| \geq n^{(1 - \frac{1}{\log^c n})}$ with probability at least $\frac{2}{3}$.*

Proof. Suppose there exists such a protocol. We can always pick a constant k such that $c < \frac{k}{k+1} < 1$. Let $a, b \in \{0,1\}^n$. We construct a protocol which distinguishes between $|a' \cap b'| = 0$ and $|a' \cap b'| > 0$ in $O(polylog(n))$ communication. Let $\hat{a}$ and $\hat{b}$ be the concatenation of $\log^k n$ copies of a and b respectively and let $m = |\hat{a}| = |\hat{b}|$, i.e., $m = n^{\log^k n}$. Also let $\hat{a}'$ and $\hat{b}'$ be the set representations of

binary vectors $\hat{a}$ and $\hat{b}$. If $|a' \cap b'| = 0$, then $|\hat{a}' \cap \hat{b}'| = 0$. On the other hand, if $|a' \cap b'| \geq 1$, then $|\hat{a}' \cap \hat{b}'| \geq n^{(\log^k n - 1)}$. But $m = n^{\log^k n} = 2^{\log^{k+1} n}$, so $\log n = (\log m)^{\frac{1}{k+1}}$. This implies that $n^{(\log^k n - 1)} = 2^{(\log^{k+1} n - \log n)} = 2^{(\log m - (\log m)^{\frac{1}{k+1}})}$ $= m^{(1-(\log m)^{\frac{1}{k+1}-1})} = m^{(1-\frac{1}{(\log m)^{\frac{k}{k+1}}})}$. But by our assumption, this can be distinguished in $O(polylog(m)) = O(polylog(n^{\log^k n})) = O(polylog(n))$ communication complexity. However the probabilistic communication complexity of set intersection is $\Theta(n)$ [15], which gives us a contradiction. ∎

Lemma 21. *Suppose Alice has $a \in \{0,1\}^n$, and Bob has $x, y \in \{0,1\}^n$. Let $k = O(polylog(n))$. Then the communication complexity of correctly answering the question " is it the case that $|a' \cap x'| \leq k \cdot |a' \cap y'|$?" with probability at least $\frac{2}{3}$ is $\Theta(n)$.*

Proof. Suppose that the communication complexity of the problem is $o(n)$. We reach a contradiction using the following steps. Suppose Alice and Bob have vectors a and b, respectively, both in $\{0,1\}^m$. Alice constructs $\hat{a} = a * 1^k$, and Bob constructs $x = b * 1^k$, and $y = 0^{m+k-1} * 1$ (where $*$ denotes concatenation) which are all binary vectors of size $n = m + k$. Clearly, $n = m + k < 2 \cdot m$, since $k = O(polylog(m))$. If $|a' \cap b'| = 0$, then $|\hat{a}' \cap x'| = k$, and $|\hat{a}' \cap y'| = 1$. Thus, $|\hat{a}' \cap x'| \leq k \cdot |\hat{a}' \cap t'|$. On the other hand, if $|a' \cap b'| > 0$, then $|\hat{a}' \cap x'| = k + |a' \cap b'| > k$, and $|\hat{a}' \cap y'| = 1$. Thus, $|\hat{a}' \cap x'| > k \cdot |\hat{a}' \cap y'|$. However, we know that we cannot distinguish between $|a' \cap b'| = 0$ and $|a' \cap b'| > 0$ in $o(n)$ communication (see [15]) which gives a contradiction. ∎

Lemma 22. *Let $c = O(1)$, then there does not exist an acceptable $(1 + \frac{c}{n})$-ANN algorithm for set intersection in the cell probe model.*

Proof. Suppose there is such an algorithm. For large enough n, $(1 + \frac{c}{n}) = (1 + \frac{1}{n})^c$, and we can apply the reduction of Lemma 17 $\log c$ times to obtain an acceptable $(1 + \frac{1}{n})$-ANN algorithm. Notice that the smallest discrepancy in distance between two points can be $\frac{n}{n-1} = 1 + \frac{1}{n-1} > 1 + \frac{1}{n}$, thus a $(1 + \frac{1}{n})$-ANN algorithm is actually an exact nearest neighbor algorithm. But [2] shows that there is no acceptable exact nearest neighbor algorithm for set intersection in the cell probe model, thus we derive a contradiction. ∎

References

[1] Omer Barkol and Yuval Rabani. Tighter lower bounds for nearest neighbor search and related problems in the cell probe model, *Proc. of STOC,* 2000.

[2] A. Borodin, R. Ostrovsky, Y. Rabani. Lower bounds for high-dimensional nearest neighbor search and related problems, *Proc. of STOC,* 1999.

[3] J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics, 52:46-52,* 1985.

[4] Amit Chakrabarti, Bernard Chazelle, Benjamin Gum, Alexey Lvov. A Lower Bound on the Complexity of Approximate Nearest-Neighbor Searching on the Hamming Cube. *Proc. ACM STOC,* 1999.

[5] A. Chakrabarti, O. Regev. An optimal randomized cell probe lower bound for approximate nearest neighbor searching. *ECCC,* 2003

[6] G. Cormode, M. Paterson, S. C. Sahinalp and U. Vishkin. Communication Complexity of Document Exchange. *Proc. ACM-SIAM Symp. on Discrete Algorithms,* 2000.

[7] G. Cormode, S. Muthukrishnan, S. C. Sahinalp. Permutation Edit Distance and Matching via Embeddings. *Proc. ICALP,* 2001.
[8] M. Farach-Colton, P. Indyk. Approximate nearest neighbor algorithms for Hausdorff metrics via embeddings. *Proc. of FOCS,* 1999.
[9] D. Hirschberg. Serial Computations of Levenshtein Distances. *Pattern Matching Algorithms,* edited by Apostolico and Galil, Oxford Univ. Press, 1997.
[10] P. Indyk. Approximate nearest neighbors in l_∞. *Proc. of FOCS,* 1998.
[11] P. Indyk. Approximate nearest neighbor algorithms for Frechet metric via product metrics. *Proc. of Symp. on Computational Geometry,* 2002.
[12] P. Indyk. Better Algorithms for High-dimensional Proximity Problems via Asymmetric Embeddings. *Proc. of 14th SODA,* 2003.
[13] P. Indyk, R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. *Proc. of 30th STOC,* 1998.
[14] T.S. Jayram, S. Khot, R. Kumar, Y. Rabani. Cell-Probe Lower Bounds for the Partial Match Problem. *Proc. of STOC,* 2003.
[15] B. Kalyanasundaram, G. Schnitger. The Probabilistic Communication Complexity of Set Intersection. *SIAM Journal on Discrete Mathematics, vol.5, pp. 545-557,* 1992.
[16] E. Kushilevitz, R. Ostrovsky, Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces *Proc. of 30th STOC,* 1998
[17] N. Linial, E. London, Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica, 15p. 215-245,* 1995.
[18] D. Liu. A strong lower bound for approximate nearest neighbor searching in the cell probe model. *Manuscript,* 2003
[19] P. B. Miltersen. Lower bounds for union-split-find related problems on random access machines. *Proc. of 26th STOC,* 1994
[20] P. B. Miltersen, N. Nisan, S. Safra, A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences, 57(1):37-49,* 1998
[21] S. Muthukrishnan, C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. *Proc. of 32nd STOC,* 2000.

A Syntactic Characterization of Distributive LTL Queries*

Marko Samer[1] and Helmut Veith[2]

[1] Institute of Information Systems (DBAI)
Vienna University of Technology, Austria
samer@dbai.tuwien.ac.at
[2] Institut für Informatik (I7)
Technische Universität München, Germany
veith@in.tum.de

Abstract. Temporal logic query solving, as introduced by Chan in 2000, is a variation of model checking where one subformula of the specification is left open and has to be filled in by the model checker in such a way that the specification becomes true. Motivated by symbolic query solving algorithms for CTL, Chan focused on a class of CTL queries which have one strongest solution, but no syntactic characterization of this class has been found yet. In this paper, we provide a syntactic characterization for the simpler case of LTL queries. We present a context-free grammar of LTL query templates $\mathbf{LTLQ}^{x}$ which guarantees that (i) all queries in $\mathbf{LTLQ}^{x}$ have at most one strongest solution, and (ii) all query templates not in $\mathbf{LTLQ}^{x}$ have simple instantiations with incomparable strongest solutions. While the LTL case appears to be simpler than the case of CTL, we believe that the proof method exhibited here can be extended to CTL as well.

1 Introduction and Overview

Given a system model $\mathfrak{M}$ and a temporal logic formula φ, a model checker is an efficient procedure for deciding $\mathfrak{M} \models \varphi$ [5,9,14]. The textbook understanding of model checking implicitly presupposes that the specification φ be known in advance. In practice however, model checkers are also used to *infer* properties about systems with unknown behavior by modifying φ and looking for the strongest variant of φ which holds true. This approach has been systematized by Chan in his seminal paper [3]. Chan introduced temporal logic queries γ, i.e., temporal logic formulas containing one occurrence of a distinguished proposition "?" which is considered a placeholder. The task of the query solver then is to find prepositional solutions φ satisfying $\mathfrak{M} \models \gamma[\varphi]$. Concentrating on CTL queries, Chan presented a symbolic algorithm which exploits the fact that certain queries always have one strongest solution and proposed a syntactic class of CTL queries with this property. Note that in general, queries may have multiple incomparable

* This work was supported by the European Community Research Training Network "Games and Automata for Synthesis and Validation" (GAMES), the EU Project ECRYPT, and by the Austrian Science Fund Project Z29-N04. The results presented in this paper have been developed as part of [15].

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1099–1110, 2004.

solutions; algorithms for computing multiple solutions as well as extensions to multiple placeholders have been the subject of related work [1,4,11,12,13].

In this paper, we return to Chan's original intention of designing a query language with unique strongest solutions. We will refer to such queries as "exact" queries. The fragment given by Chan is easily seen not to cover all exact CTL queries. Indeed, in previous work [16], we have corrected an error in Chan's original definition and extended his class to a richer fragment, but no syntactic characterization of exact CTL queries has been achieved so far despite significant efforts. We will therefore concentrate on the apparently simpler question of LTL queries in the current paper. We believe that the proof methods developed for LTL can potentially be extended to CTL as well; we will discuss this in the conclusion.

It was noticed already by Chan that the property

$$\gamma[\varphi \wedge \psi] \;\Leftrightarrow\; \gamma[\varphi] \wedge \gamma[\psi] \qquad \textbf{(Distributivity)}$$

is closely related to exactness. In fact, by Theorem 1 below, distributivity even characterizes the exact queries. As conjunction in general distributes over universal quantification only, the distributivity of a query γ gives evidence that the occurrence of the placeholder is governed by a universal quantification. The simplest instance of this observation is given by the the query $\mathbf{G}\,?$ whose exactness amounts to the equivalence of $\mathbf{G}(\varphi \wedge \psi)$ and $\mathbf{G}\,\varphi \wedge \mathbf{G}\,\psi$.

Distributivity has the potential to be exploited in model checking algorithms. Suppose that a specification φ can be written in the form $\gamma[\bigwedge_i \psi_i]$ where γ is exact. Then model checking φ can be reduced to a sequence of model checker calls for the properties $\gamma[\psi_i]$. In most cases, it is not to be expected that the reduced formula size will significantly effect the performance of the model checker. It is however quite likely that each of the formulas $\gamma[\psi_i]$ ranges over a significantly fewer number of atomic propositions, e.g., for loosely coupled concurrent systems with limited communication and many internal variables. This situation can be exploited by a model checker using existential abstraction, since a smaller number of variables in the specification will in general reduce the size of the abstract model [6,8]. On a more general note, a syntactic criterion for distributivity can also be used to enhance axiomatic systems for temporal logics by extending the known class of simple equivalences between temporal logic formulas [10].

Finding a syntactic criterion for exactness however appears to be hard as indicated by the fact that deciding exactness is EXPTIME-complete for CTL queries and PSPACE-complete for LTL queries. Consequently, there can be no context-free or other reasonably simple grammar which recognizes the exact queries. Notwithstanding this principal limitation, the main result of the current paper is a syntactic characterization of exact LTL queries in the following sense:

We consider *query templates,* i.e., query classes where a "wildcard" $\star$ is used to describe ordinary LTL subformulas which do not contain the placeholder. For example, the template $(\mathbf{X}\,?)\,\mathbf{U}\star$ stands for all queries $(\mathbf{X}\,?)\mathbf{U}\,\varphi$, where φ is an LTL formula. The *deterministic* grammar of Table 1 defines two template languages $\mathbf{LTLQ}^{x}$ and $\overline{\mathbf{LTLQ}^{x}}$, where $\mathbf{LTLQ}^{x}$ is derived from $\langle EX \rangle$ and $\overline{\mathbf{LTLQ}^{x}}$ is derived from $\langle NX \rangle$. When the context is clear we will say that a query is contained in a template language (or write $\gamma \in \mathcal{L}$) if γ is an instantiation of a query in $\mathcal{L}$. A query is positive, if the placeholder is not in the

Table 1. LTLQ production rules. The non-terminal $\langle EX \rangle$ defines the exact query templates, and the non-terminal $\langle NX \rangle$ defines the non-exact query templates.

$\langle Q1 \rangle$	::=	$?$	$\mid$	$\star \wedge \langle Q2 \rangle$	$\mid$	$\langle Q2 \rangle\ \mathring{\mathbf{U}}\ \star$	$\mid$	$\star\ \mathring{\mathbf{U}}\ \langle Q2 \rangle$	$\mid$
		$\star\ \bar{\mathbf{U}}\ \langle Q2 \rangle$	$\mid$	$\langle Q2 \rangle\ \mathring{\mathbf{W}}\ \star$	$\mid$	$\star\ \mathring{\mathbf{W}}\ \langle Q2 \rangle$	$\mid$	$\star\ \bar{\mathbf{W}}\ \langle Q2 \rangle$	$\mid$
		$\star \wedge \langle Q1 \rangle$	$\mid$	$\star \vee \langle Q1 \rangle$	$\mid$	$\mathbf{X}\ \langle Q1 \rangle$	$\mid$	$\langle Q1 \rangle\ \mathring{\mathbf{U}}\ \star$	$\mid$
		$\star\ \bar{\mathbf{U}}\ \langle Q1 \rangle$	$\mid$	$\langle Q1 \rangle\ \mathring{\mathbf{W}}\ \star$	$\mid$	$\star\ \bar{\mathbf{W}}\ \langle Q1 \rangle$	;		
$\langle Q2 \rangle$	::=	$\langle Q1 \rangle\ \mathbf{U}\ \star$	$\mid$	$\langle Q1 \rangle\ \mathbf{W}\ \star$	$\mid$	$\star \vee \langle Q2 \rangle$	$\mid$	$\mathbf{X}\ \langle Q2 \rangle$	$\mid$
		$\langle Q2 \rangle\ \mathbf{U}\ \star$	$\mid$	$\star\ \mathbf{U}\ \langle Q2 \rangle$	$\mid$	$\langle Q2 \rangle\ \mathbf{W}\ \star$	$\mid$	$\star\ \mathbf{W}\ \langle Q2 \rangle$	;
$\langle Q3 \rangle$	::=	$\mathbf{F}\ \langle Q1 \rangle$	$\mid$	$\mathbf{F}\ \langle Q5 \rangle$	$\mid$	$\mathbf{F}\ \langle Q6 \rangle$	$\mid$	$\mathbf{G}\ \langle Q4 \rangle$	$\mid$
		$\mathbf{G}\ \langle Q6 \rangle$	$\mid$	$\langle Q4 \rangle\ \mathring{\mathbf{U}}\ \star$	$\mid$	$\star\ \mathbf{U}\ \langle Q1 \rangle$	$\mid$	$\star\ \mathbf{U}\ \langle Q5 \rangle$	$\mid$
		$\star\ \mathbf{U}\ \langle Q6 \rangle$	$\mid$	$\star\ \mathring{\mathbf{U}}\ \langle Q1 \rangle$	$\mid$	$\star\ \mathring{\mathbf{U}}\ \langle Q5 \rangle$	$\mid$	$\star\ \mathring{\mathbf{U}}\ \langle Q6 \rangle$	$\mid$
		$\star\ \bar{\mathbf{U}}\ \langle Q4 \rangle$	$\mid$	$\star\ \bar{\mathbf{U}}\ \langle Q5 \rangle$	$\mid$	$\star\ \bar{\mathbf{U}}\ \langle Q6 \rangle$	$\mid$	$\langle Q4 \rangle\ \mathbf{W}\ \star$	$\mid$
		$\langle Q6 \rangle\ \mathbf{W}\ \star$	$\mid$	$\langle Q4 \rangle\ \mathring{\mathbf{W}}\ \star$	$\mid$	$\star\ \mathbf{W}\ \langle Q5 \rangle$	$\mid$	$\star\ \mathbf{W}\ \langle Q6 \rangle$	$\mid$
		$\star\ \bar{\mathbf{W}}\ \langle Q4 \rangle$	$\mid$	$\star\ \bar{\mathbf{W}}\ \langle Q5 \rangle$	$\mid$	$\star\ \bar{\mathbf{W}}\ \langle Q6 \rangle$	$\mid$	$\star \wedge \langle Q3 \rangle$	$\mid$
		$\star \vee \langle Q3 \rangle$	$\mid$	$\mathbf{X}\ \langle Q3 \rangle$	$\mid$	$\mathbf{F}\ \langle Q3 \rangle$	$\mid$	$\mathbf{G}\ \langle Q3 \rangle$	$\mid$
		$\langle Q3 \rangle\ \mathring{\mathbf{U}}\ \star$	$\mid$	$\star\ \mathbf{U}\ \langle Q3 \rangle$	$\mid$	$\star\ \mathring{\mathbf{U}}\ \langle Q3 \rangle$	$\mid$	$\star\ \bar{\mathbf{U}}\ \langle Q3 \rangle$	$\mid$
		$\langle Q3 \rangle\ \mathbf{W}\ \star$	$\mid$	$\langle Q3 \rangle\ \mathring{\mathbf{W}}\ \star$	$\mid$	$\star\ \mathbf{W}\ \langle Q3 \rangle$	$\mid$	$\star\ \bar{\mathbf{W}}\ \langle Q3 \rangle$	;
$\langle Q4 \rangle$	::=	$\langle Q3 \rangle\ \mathbf{U}\ \star$	$\mid$	$\langle Q5 \rangle\ \mathbf{U}\ \star$	$\mid$	$\langle Q6 \rangle\ \mathbf{U}\ \star$	$\mid$	$\langle Q5 \rangle\ \mathbf{W}\ \star$	$\mid$
		$\star \vee \langle Q4 \rangle$	$\mid$	$\mathbf{X}\ \langle Q4 \rangle$	$\mid$	$\langle Q4 \rangle\ \mathbf{U}\ \star$	$\mid$	$\star\ \mathbf{U}\ \langle Q4 \rangle$	$\mid$
		$\star\ \mathbf{W}\ \langle Q4 \rangle$	;						
$\langle Q5 \rangle$	::=	$\star\ \mathring{\mathbf{U}}\ \langle Q4 \rangle$	$\mid$	$\star\ \mathbf{W}\ \langle Q1 \rangle$	$\mid$	$\star\ \mathring{\mathbf{W}}\ \langle Q1 \rangle$	$\mid$	$\star\ \mathring{\mathbf{W}}\ \langle Q3 \rangle$	$\mid$
		$\star\ \mathring{\mathbf{W}}\ \langle Q4 \rangle$	$\mid$	$\star\ \mathring{\mathbf{W}}\ \langle Q6 \rangle$	$\mid$	$\star \wedge \langle Q5 \rangle$	$\mid$	$\mathbf{X}\ \langle Q5 \rangle$	$\mid$
		$\langle Q5 \rangle\ \mathring{\mathbf{U}}\ \star$	$\mid$	$\langle Q5 \rangle\ \mathring{\mathbf{W}}\ \star$	$\mid$	$\star\ \mathring{\mathbf{W}}\ \langle Q5 \rangle$	;		
$\langle Q6 \rangle$	::=	$\star \wedge \langle Q4 \rangle$	$\mid$	$\star \vee \langle Q5 \rangle$	$\mid$	$\star \wedge \langle Q6 \rangle$	$\mid$	$\star \vee \langle Q6 \rangle$	$\mid$
		$\mathbf{X}\ \langle Q6 \rangle$	$\mid$	$\langle Q6 \rangle\ \mathring{\mathbf{U}}\ \star$	$\mid$	$\langle Q6 \rangle\ \mathring{\mathbf{W}}\ \star$	;		
$\langle Q7 \rangle$	::=	$\mathbf{F}\ \langle Q2 \rangle$	$\mid$	$\mathbf{F}\ \langle Q4 \rangle$	$\mid$	$\mathbf{G}\ \langle Q1 \rangle$	$\mid$	$\mathbf{G}\ \langle Q2 \rangle$	$\mid$
		$\mathbf{G}\ \langle Q5 \rangle$	$\mid$	$\star \wedge \langle Q7 \rangle$	$\mid$	$\star \vee \langle Q7 \rangle$	$\mid$	$\mathbf{X}\ \langle Q7 \rangle$	$\mid$
		$\mathbf{F}\ \langle Q7 \rangle$	$\mid$	$\mathbf{G}\ \langle Q7 \rangle$	$\mid$	$\langle Q7 \rangle\ \mathbf{U}\ \star$	$\mid$	$\langle Q7 \rangle\ \mathring{\mathbf{U}}\ \star$	$\mid$
		$\star\ \mathbf{U}\ \langle Q7 \rangle$	$\mid$	$\star\ \mathring{\mathbf{U}}\ \langle Q7 \rangle$	$\mid$	$\star\ \bar{\mathbf{U}}\ \langle Q7 \rangle$	$\mid$	$\langle Q7 \rangle\ \mathbf{W}\ \star$	$\mid$
		$\langle Q7 \rangle\ \mathring{\mathbf{W}}\ \star$	$\mid$	$\star\ \mathbf{W}\ \langle Q7 \rangle$	$\mid$	$\star\ \mathring{\mathbf{W}}\ \langle Q7 \rangle$	$\mid$	$\star\ \bar{\mathbf{W}}\ \langle Q7 \rangle$	;
$\langle EX \rangle$	::=	$\langle Q1 \rangle$	$\mid$	$\langle Q2 \rangle$	$\mid$	$\langle Q7 \rangle$	;		
$\langle NX \rangle$	::=	$\langle Q3 \rangle$	$\mid$	$\langle Q4 \rangle$	$\mid$	$\langle Q5 \rangle$	$\mid$	$\langle Q6 \rangle$	;

scope of a negation, i.e., we assume negation normal form by default. Our main result is then stated as follows:

Classification Theorem

(i) All queries derived from templates in LTLQ^x *are exact.*
(ii) Each template in $\overline{\mathit{LTLQ}^x}$ *has a non-exact instance. Moreover, the non-exact instances are obtained by instantiating the wildcards by atomic propositions.*
(iii) Together, LTLQ^x *and* $\overline{\mathit{LTLQ}^x}$ *constitute all positive single-placeholder LTL queries over the set of temporal operators* $\mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{U}, \mathring{\mathbf{U}}, \bar{\mathbf{U}}, \mathbf{W}, \mathring{\mathbf{W}}, \bar{\mathbf{W}}$.

Several comments are in place to explain the range and restrictions of this result:

- It is well-known that the fragment of LTL based on $\mathbf{X}$ and $\mathbf{U}$ already achieves the expressive power of LTL. This is not the case for LTL query languages (or CTL query languages for that matter). For example, adding the operator $\varphi \mathring{\mathbf{U}} \psi = \varphi \mathbf{U} (\varphi \wedge \psi)$ makes it possible to formulate a query $? \mathring{\mathbf{U}} \psi$, although $? \mathbf{U} (? \wedge \psi)$ has two occurrences of $?$, and is therefore not allowed. The concrete choice of temporal operators follows Chan's paper. We believe that for all LTL definable operators, a similar classification can be obtained.
- Our characterization also holds true for arbitrary subsets of the considered operators by removing the corresponding grammar elements.
- Since a negation in front of the placeholder can be removed without loss of generality, we can assume that all queries are positive; consequently, part (iii) of the Classification Theorem is no restriction of generality for all fragments of our query language allowing a negation normal form. Note however, that if we allowed multiple occurrences of the same placeholder, then we would have to distinguish between positive and negative occurrences.
- The only special case is given by the operators $\varphi \bar{\mathbf{U}} \psi \Leftrightarrow \varphi \mathbf{U} (\neg\varphi \wedge \psi)$ and $\varphi \bar{\mathbf{W}} \psi \Leftrightarrow \varphi \mathbf{W} (\neg\varphi \wedge \psi)$ which, in contrast to the other operators, are not monotone in their first argument. Consequently, we need to syntactically restrict them to positive occurrences of the placeholder.
- The dependency graph in Figure 1 shows the relationships between the rules in the grammar. Interestingly, an exact query can be extended to a non-exact query, and vice versa. In combination with the cyclicity of the dependency graph, this situation renders the inductive proofs required for parts (i) and (ii) of the theorem non-trivial.

Templates are a natural concept for the syntactic characterization of logic formulas. For example, Buccafurri et al. [2] have given a template characterization for the fragment of ACTL with linear counterexamples; more generally, all prefix characterizations which are very common in formal logic amount to a form of template characterization. Note however that a priori we cannot exclude the existence of a simple (e.g., context-free) syntactic class of LTL queries which contains all exact queries *up to logical equivalence.* Since deciding logical equivalence of LTL formulas is PSPACE-complete, such a characterization would not contradict the PSPACE-completeness of determining exactness of LTL queries. While such a characterization would be semantically stronger than ours, we believe that it will be hard to find and unnatural as a specification language.

The rest of this paper is organized as follows: Following some simple preliminaries in Section 2, we state basic properties of temporal logic query languages in Section 3. Section 4 finally contains the technical result. As the grammar indicates, the inductive proofs are nested and too large for inclusion in this extended abstract; we will therefore focus on the proof structure rather than on details. The important part of the exactness proof in Section 4.1 is finding the intermediate induction properties which are necessary to make the proof go through. Section 4.2 describes the inductive construction of paths for which the exactness of $\overline{\mathbf{LTLQ}^{x}}$ queries fails. Section 5 concludes the paper.

2 Preliminaries

We assume the reader is familiar with the basics of model checking, i.e., Kripke structures and the linear temporal logic LTL based on the temporal operators **X** ("next"), **F** ("future"), **G** ("global"), and **U** ("until").

Let $\mathfrak{K} = (\mathcal{Q}, \mathcal{Q}_0, \delta, \ell)$ be a *Kripke structure* over the set $\mathcal{A}$ of atomic propositions, where $\mathcal{Q}$ is a set of states, $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is the set of initial states, $\delta : \mathcal{Q} \to \wp(\mathcal{Q})$ is a total transition function, and $\ell : \mathcal{Q} \to \wp(\mathcal{A})$ is a total labeling function. A *computation path* π in $\mathfrak{K}$ is an infinite sequence of states $\pi : \mathbb{N} \to \mathcal{Q}$ such that $\pi(0) \in \mathcal{Q}_0$ and for all $s, s' \in \mathcal{Q}$ and $i \in \mathbb{N}$ it holds that $s' \in \delta(s)$ if $\pi(i) = s$ and $\pi(i+1) = s'$. An infinite sequence of states $\sigma : \mathbb{N} \to \mathcal{Q}$ is called a *computation path suffix* or simply a *path* iff there exists a computation path π and an $n \in \mathbb{N}$ such that $\pi(n+i) = \sigma(i)$ for all $i \in \mathbb{N}$. We write π^n to denote the computation path suffix of the computation path π iff $\pi(n+i) = \pi^n(i)$ for all $i \in \mathbb{N}$. As usual we write $\mathfrak{K}, \pi \models \varphi$ to denote that the LTL formula φ is satisfied on path π in $\mathfrak{K}$, and we write $\mathfrak{K} \models \varphi$ to denote $\mathfrak{K}, \pi \models \varphi$ for all paths π in $\mathfrak{K}$. For simplicity, we will omit $\mathfrak{K}$ if it is clear from the context, i.e., we will write $\pi \models \varphi$ instead of $\mathfrak{K}, \pi \models \varphi$.

The operators $\mathbf{F}\varphi \Leftrightarrow \top \mathbf{U} \varphi$ and $\varphi \to \psi \Leftrightarrow \neg\varphi \vee \psi$ are defined as usual. Since we will consider formulas in negation normal form only, all other operators are defined directly and not as abbreviations. Following Chan [3], we use some additional temporal operators. In particular, we use the weak until operator $\varphi \mathbf{W} \psi \Leftrightarrow (\mathbf{G}\varphi) \vee (\varphi \mathbf{U} \psi)$. The other operators are variants of the strong until operator **U** and the weak until operator **W**:

$$\varphi \mathring{\mathbf{U}} \psi \Leftrightarrow \varphi \mathbf{U} (\varphi \wedge \psi) \qquad \varphi \mathring{\mathbf{W}} \psi \Leftrightarrow \varphi \mathbf{W} (\varphi \wedge \psi)$$
$$\varphi \bar{\mathbf{U}} \psi \Leftrightarrow \varphi \mathbf{U} (\neg\varphi \wedge \psi) \qquad \varphi \bar{\mathbf{W}} \psi \Leftrightarrow \varphi \mathbf{W} (\neg\varphi \wedge \psi)$$

As argued above, the additional operators will give rise to stronger temporal logic queries although they do not increase the expressive power of LTL.

3 LTL Queries

In this section, we survey some basic properties of temporal logic queries and provide a characterization and complexity analysis of exactness. Although the proof is not hard, it has not been explicitly stated in the literature. Throughout this paper we will write γ, $\bar{\gamma}$, etc. to denote queries, and use all other Greek letters to denote formulas.

Definition 1 (LTL query). *An* LTL query *is an LTL formula where some subformulas are replaced by a special variable* ?, *called* placeholder. *We denote the set of all LTL queries by LTLQ.*

Definition 2 (Solution). *Let γ be a query, $\mathfrak{K}$ be a Kripke structure, and φ be a formula. We write $\gamma[\varphi]$ to denote the result of substituting all occurrences of the placeholder in γ by φ. If $\mathfrak{K} \models \gamma[\varphi]$, then we say that φ is a* solution *to γ in $\mathfrak{K}$. We denote the set of all solutions to γ in $\mathfrak{K}$ by $\mathit{sol}(\mathfrak{K}, \gamma) = \{\varphi \mid \mathfrak{K} \models \gamma[\varphi]\}$.*

A query γ can be viewed as a function $\gamma : \varphi \mapsto \gamma[\varphi]$ which maps formulas to formulas. Thus, an important and natural property is monotonicity, in particular, queries for which the corresponding function is monotonic increasing.

Definition 3 (Monotone query). *A query γ is* monotone *iff $\varphi \Rightarrow \psi$ implies $\gamma[\varphi] \Rightarrow \gamma[\psi]$ for all formulas φ and ψ.*

The set of solutions to a query forms a partially ordered set ordered by implication. Let $\mathcal{M}$ be the set of minimal elements (if they exist) of this poset of γ in $\mathfrak{K}$. Then it holds that $\mathbf{sol}(\mathfrak{K}, \gamma) \subseteq \{\varphi \mid \exists \mu \in \mathcal{M}.\ \mu \Rightarrow \varphi\}$. It can be easily shown that for monotone queries also the converse inclusion holds. In other words, if γ is monotone, it holds that $\mathbf{sol}(\mathfrak{K}, \gamma) = \{\varphi \mid \exists \mu \in \mathcal{M}.\ \mu \Rightarrow \varphi\}$.

Definition 4 (Exact solution). *A solution ξ to a query γ in a Kripke structure $\mathfrak{K}$ is* exact *iff it holds that $\mathit{sol}(\mathfrak{K}, \gamma) = \{\varphi \mid \xi \Rightarrow \varphi\}$*

We are now able to give a formal definition of exact queries.

Definition 5 (Exact query). *A query is* exact *iff it has an exact solution in every model where the set of solutions is not empty.*

For the two directions of distributivity we introduce special terminology as well:

Definition 6 (Collecting, Separating). *A query γ is* collecting *iff it satisfies $\gamma[\varphi] \wedge \gamma[\psi] \Rightarrow \gamma[\varphi \wedge \psi]$, and* separating *iff it satisfies $\gamma[\varphi \wedge \psi] \Rightarrow \gamma[\varphi] \wedge \gamma[\psi]$ for all formulas φ and ψ. A query is* distributive *iff it is collecting and separating.*

Item 3 of the following characterization will be the stepping stone for proving exactness:

Theorem 1. *Let γ be any query. Then the following are equivalent:*

1. *γ is exact.*
2. *γ is distributive.*
3. *γ is monotone and collecting.*

In analogy to Chan's EXPTIME-completeness proof for deciding validity of CTL queries [3] we can determine the complexity of deciding exactness of LTL queries by reduction from and to the validity of LTL formulas:

Theorem 2. *Deciding whether an LTL query is exact is* PSPACE*-complete.*

As argued above, the main implication of this theorem is the fact that no simple grammar can recognize the exact LTL queries.

Remark. Note that with the exception of Theorem 2, all results in this section hold generally for all classes of CTL* queries regardless of the number and polarity of placeholders.

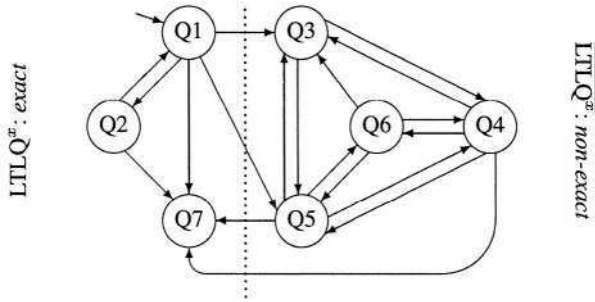


Fig. 1. LTLQ dependence diagram

4 The Characterization of Exact LTL Queries

In this section, we analyze $\mathbf{LTLQ}^x$. Recall that we are interested only in queries with a single placeholder. By Theorem 1, to prove exactness of a query, it suffices to show that the query is monotone and collecting. We begin with

Definition 7 ($\mathbf{LTLQ}^m$). *The language* $\mathbf{LTLQ}^m$ *is the largest set of LTL queries which do not contain a subquery of the form* $\gamma\,\bar{\mathbf{U}}\,\varphi$ *or* $\gamma\,\bar{\mathbf{W}}\,\varphi$.

For example, the query $\mathbf{X}\,(\varphi\,\bar{\mathbf{U}}\,\mathbf{G}\,?)$ is in $\mathbf{LTLQ}^m$, whereas $\mathbf{X}\,((\varphi \vee ?)\,\bar{\mathbf{U}}\,\psi)$ is not in $\mathbf{LTLQ}^m$. The intuition behind the definition of $\mathbf{LTLQ}^m$ above is simply to restrict our considerations to positive LTL queries, i.e., queries in which the placeholder appears under an even number of negations.

Lemma 1. *Every query in* $\mathbf{LTLQ}^m$ *is monotone.*

In the following we write $\mathbf{LTLQ}^1$ for the language derived from non-terminal $\langle Q1\rangle$, $\mathbf{LTLQ}^2$ for the language derived from $\langle Q2\rangle$, and so on. It is easy to see that $\mathbf{LTLQ}^m = \bigcup_{i=1}^{7} \mathbf{LTLQ}^i$, since every operator allowed in $\mathbf{LTLQ}^m$ occurs in combination with every non-terminal. In fact, the rules in Table 1 provide a deterministic context-free grammar for $\mathbf{LTLQ}^m$. The language $\mathbf{LTLQ}^x$ is defined as $\mathbf{LTLQ}^x = \mathbf{LTLQ}^1 \cup \mathbf{LTLQ}^2 \cup \mathbf{LTLQ}^7$. Its complement is given by $\overline{\mathbf{LTLQ}^x} = \mathbf{LTLQ}^m \backslash \mathbf{LTLQ}^x = \mathbf{LTLQ}^3 \cup \mathbf{LTLQ}^4 \cup \mathbf{LTLQ}^5 \cup \mathbf{LTLQ}^6$.

The dependencies between the non-terminals in the above grammar are illustrated in Figure 1. For example, there is an edge from vertex Q1 to vertex Q7 because in the definition of non-terminal $\langle Q7\rangle$ there appears $\mathbf{G}\,\langle Q1\rangle$, i.e., there is an operator that leads from non-terminal $\langle Q1\rangle$ to non-terminal $\langle Q7\rangle$. Since the grammar is deterministic, each query can be uniquely assigned to a vertex in the graph. For example, it can be easily checked that the query $(b\,\mathbf{U}\,(a \wedge ?))\,\mathbf{U}\,c$ belongs to vertex Q4. The vertices on the left hand side of the dotted line represent $\mathbf{LTLQ}^x$ and the vertices on the right hand side represent $\overline{\mathbf{LTLQ}^x}$.

The proofs of the following lemmas are structural inductions on queries in $\mathbf{LTLQ}^m$. In particular, we will show that all queries in $\mathbf{LTLQ}^x$ are exact and that in each sub-language $\mathbf{LTLQ}^3$, $\mathbf{LTLQ}^4$, $\mathbf{LTLQ}^5$, and $\mathbf{LTLQ}^6$ of its complement $\overline{\mathbf{LTLQ}^x}$ there exists

a natural class of queries that are not exact. Therefore, the non-trivial dependencies between these sublanguages, i.e., the non-terminals, as shown in Figure 1 shape the form of our inductive proofs. As we shall see soon, a major complication in the proofs arises from the fact that the dependencies are circular and that different induction properties are needed for the different $\mathbf{LTLQ}^i$.

4.1 Proving Exactness of $\mathbf{LTLQ}^x$

This section is devoted to the proof of our first main result:

Theorem 3. *Every query in $\mathbf{LTLQ}^x$ is exact.*

By Theorem 1, it suffices to show that all queries in $\mathbf{LTLQ}^1$, $\mathbf{LTLQ}^2$, and $\mathbf{LTLQ}^7$ are collecting. Unfortunately, it is not possible to show the collecting property directly by induction, because as an induction property, the collecting property contains too few information. (By the induction property of an inductive proof we understand the concrete property which is proven.) Instead, we need to use the following strengthenings of the collecting property:

Definition 8. *Let $\gamma \in \mathbf{LTLQ}$. Then we define the following collecting properties:*

γ *is* **strong collecting:**
If $\pi \models \gamma[\varphi]$ *and* $\pi^n \models \gamma[\psi]$ *then* $\pi^n \models \gamma[\varphi \wedge \psi]$.

γ ***is* boundary collecting:**
If $\pi \models \gamma[\varphi]$ *and* $\pi^n \models \gamma[\psi]$ *then* $\pi \models \gamma[\bot]$ *or* $\pi^n \models \gamma[\varphi \wedge \psi]$.

γ *is* **intermediate collecting:**
If $\pi \models \gamma[\varphi]$ *and* $\pi^n \models \gamma[\psi]$ *then* $\pi^n \models \gamma[\varphi \wedge \psi]$ *or there exists an* $r < n$ *such that* $\pi^r \models \gamma[\bot]$.

γ *is* **weak collecting** *iff it is collecting.*

It follows from the definition that every strong collecting query is also boundary collecting, every boundary collecting query is also intermediate collecting ($r = 0$), and every intermediate collecting query is also weak collecting ($n = 0$).

The following lemma can be shown by inductive proofs on the structure and the number of queries in $\mathbf{LTLQ}^1$ and $\mathbf{LTLQ}^2$.

Lemma 2. *Every query in $\mathbf{LTLQ}^1$ is collecting, and every query in $\mathbf{LTLQ}^2$ is intermediate collecting.*

Note that every query in $\mathbf{LTLQ}^7$ contains a subquery of the form $\mathbf{F}\,\gamma$, where $\gamma \in \mathbf{LTLQ}^2 \cup \mathbf{LTLQ}^4$, or of the form $\mathbf{G}\,\gamma$, where $\gamma \in \mathbf{LTLQ}^1 \cup \mathbf{LTLQ}^2 \cup \mathbf{LTLQ}^5$. Therefore, we need the following auxiliary result to prove the collecting property of queries in $\mathbf{LTLQ}^7$. It can be shown by inductive proofs on the structure and the number of queries in $\mathbf{LTLQ}^4$ and $\mathbf{LTLQ}^5$.

Lemma 3. *If $\gamma \in \mathbf{LTLQ}^4$, then $\mathbf{F}\,\gamma$ is boundary collecting; if $\gamma \in \mathbf{LTLQ}^5$, then $\mathbf{G}\,\gamma$ is strong collecting.*

Now, the collecting property of queries in $\mathbf{LTLQ}^7$ can be shown by structural induction. As induction start we can use Lemma 2 and Lemma 3.

Lemma 4. *Every query in* $\mathbf{LTLQ}^7$ *is boundary collecting.*

Remark. Note that we use Lemma 3 in the induction start for proving Lemma 4, i.e., we use *non-exact* subqueries as induction start to prove the collecting property of queries in $\mathbf{LTLQ}^7$. From the dependency graph in Figure 1 it is easy to see that by adding appropriate operators to non-exact queries we can always obtain exact queries, because from Q4 and Q5 we can reach Q7 through a single edge, and from Q3 and Q6 we can reach Q7 in two steps.

Theorem 3 now follows from Theorem 1 by putting together the facts that (i) $\mathbf{LTLQ}^x$ is defined as the union of $\mathbf{LTLQ}^1$, $\mathbf{LTLQ}^2$, and $\mathbf{LTLQ}^7$, (ii) that $\mathbf{LTLQ}^m$ is monotone, and (iii) the fact that intermediate collecting queries are also boundary collecting, and boundary collecting queries are also (weak) collecting.

4.2 Constructing Counterexamples for $\overline{\mathbf{LTLQ}^x}$

In this section, we will show that our fragment $\mathbf{LTLQ}^x$ is maximal in the sense that all templates in $\overline{\mathbf{LTLQ}^x}$ have simple instantiations which are not exact. By Theorem 1, it suffices to show that the templates in each sublanguage $\mathbf{LTLQ}^3$, $\mathbf{LTLQ}^4$, $\mathbf{LTLQ}^5$, and $\mathbf{LTLQ}^6$ of $\overline{\mathbf{LTLQ}^x}$ contain queries which are not collecting. To this end, let us define a formal notion of simple queries.

Definition 9 (Simple query). *An LTL query* γ *is* simple *iff every ordinary LTL subformula (i.e., every subformula without a placeholder) of* γ *is atomic and occurs only once in* γ.

In particular, we can obtain a simple query from a template by replacing each wildcard $\star$ in the template by a new atomic proposition. For example, the query $\mathbf{G}\,(a\,\mathbf{U}\,(b \wedge \mathbf{X}\,?))$ is simple, whereas $\mathbf{G}\,((a \vee b)\,\mathbf{U}\,\mathbf{X}\,?)$ is not simple because of $a \vee b$, and neither is $\mathbf{G}\,(a\,\mathbf{U}\,\mathbf{X}\,(a \wedge ?))$ because a occurs twice. From the definition of simplicity, we know that all subformulas of a simple query are independent from each other, as they are different atomic propositions. This will be important in the inductive construction of a counterexample path. In this way, we are able to prove our second main result:

Theorem 4. *All simple queries in* $\overline{\mathbf{LTLQ}^x}$ *violate distributivity and are therefore not exact.*

Given a simple query γ obtained from a template in $\overline{\mathbf{LTLQ}^x}$, our proof strategy is to inductively construct a path π such that

$$\pi \models \gamma[p] \wedge \gamma[q] \qquad \text{but} \qquad \pi \not\models \gamma[p \wedge q]. \tag{$*$}$$

This is exemplified in the following:

Example 1. Consider the simple query $\gamma = (b\,\mathbf{U}\,(a \wedge ?))\,\mathbf{U}\,c$. Since $\gamma \in \mathbf{LTLQ}^4$, there exists a path π such that $\pi \models \gamma[p] \wedge \gamma[q]$ but $\pi \not\models \gamma[p \wedge q]$ for any atomic propositions p and q not occurring in γ. The construction of π for this simple example according to our proof is illustrated in Figure 2. We start with an initial path on which for all $n \in \mathbb{N}$ it holds that $\pi^{4n} \models p$, $\pi^{4n+2} \models q$, and $\pi \models \mathbf{G}\,\neg(p \wedge q)$. Then, according to the structure

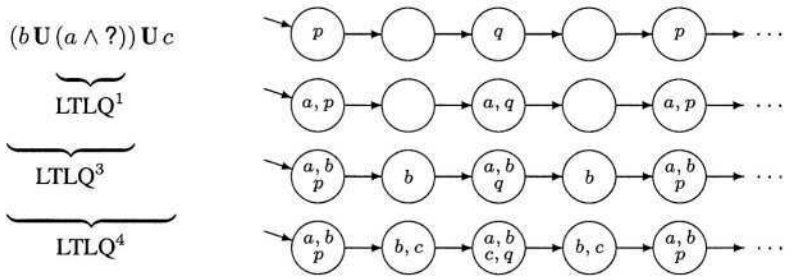


Fig. 2. Counterexample construction

of γ, we successively add labels to the states of π in such a way that we can always achieve resp. preserve a counterexample by adding further labels. It is easy to see that the resulting path in Figure 2 is indeed a counterexample to the collecting property. Note that there exist simpler ad hoc counterexamples for the query γ. The construction in Figure 2 however illustrates the general method that works for all queries.

Similarly as with the proof of exactness in the previous section, the tricky part of this proof is to find suitable induction properties which are stronger than $(*)$. As the detailed induction proof is even longer than the proof of exactness, we omit the proof details and concentrate on the general structure.

Note that every query in $\overline{\mathbf{LTLQ}^x}$ must contain an exact subquery in $\mathbf{LTLQ}^1$ such that there is no exact subquery in $\mathbf{LTLQ}^x$ between them. By induction on the structure and the number of queries in $\mathbf{LTLQ}^1$and $\mathbf{LTLQ}^2$, we can prove the following property:

Lemma 5. *Let $\gamma \in \mathbf{LTLQ}^1$ be simple, and let p and q be atomic propositions not occurring in γ. Then there exists a path π such that for all $n \in \mathbb{N}$ it holds that $\pi^{4n} \models \gamma[p]$, $\pi^{4n+2} \models \gamma[q]$, and $\pi \models \mathbf{G}\,\neg\gamma[p \wedge q]$.*

Note that the property proven in Lemma 5 does *not* have $(*)$ as a special case (not surprisingly, given the fact that $\mathbf{LTLQ}^1$ is exact). The property shown is however instrumental as an induction start for the construction of counterexample paths. (Recall that all paths in Figure 1 leading into $\overline{\mathbf{LTLQ}^x}$ must pass through $\mathbf{LTLQ}^1$.)

The following lemma, in combination with Table 2, summarizes a sequence of lemmas which describe the structure of the counterexample paths. Since the dependencies are circular, the proof of these properties requires a complicated nested induction on the structure and the number of queries in $\mathbf{LTLQ}^3$, $\mathbf{LTLQ}^4$, $\mathbf{LTLQ}^5$, and $\mathbf{LTLQ}^6$.

Lemma 6. *Let $\gamma \in \overline{\mathbf{LTLQ}^x}$ be simple, and let p and q be atomic propositions not occurring in γ. Then there exists a path π such that for all $n \in \mathbb{N}$ it holds that*

$$\pi^{f_p^\gamma(n)} \models \gamma[p] \quad \textit{and} \quad \pi^{f_q^\gamma(n)} \models \gamma[q], \quad \textit{but} \quad \pi^{f_{p\wedge q}^\gamma(n)} \not\models \gamma[p \wedge q]$$

where $f_\varphi^\gamma : \mathbb{N} \to \mathbb{N}$ is a function in n defined in Table 2.

For example, if $\gamma \in \mathbf{LTLQ}^4$, the above lemma says that there exists a path π such that $\pi \models \mathbf{G}\,\gamma[p] \wedge \mathbf{G}\,\gamma[q]$ but $\pi^{4n} \not\models \gamma[p \wedge q]$ for all $n \in \mathbb{N}$. It is easy to see that the

Table 2. Structure of counterexample paths

f_φ^γ	$\varphi = p$	$\varphi = q$	$\varphi = p \wedge q$
$\gamma \in \mathrm{LTLQ}^3$	n	n	n
$\gamma \in \mathrm{LTLQ}^4$	n	n	$4n$
$\gamma \in \mathrm{LTLQ}^5$	$4n$	$4n$	n
$\gamma \in \mathrm{LTLQ}^6$	$4n$	$4n$	n

properties proven in Lemma 6 have property $(*)$ as special cases, whence we obtain the following corollary.

Corollary 1. *Let $\gamma \in \overline{LTLQ^x}$ be simple. Then there exists a path on which the collecting property of γ is violated. Consequently, γ is not distributive.*

Together with Theorem 1, this immediately implies Theorem 4.

Remark. Similarly as with prefix characterizations, our results do not exclude that *certain* non-simple queries in $\overline{\mathrm{LTLQ}^x}$ are exact. For example, the queries $a\,\mathbf{U}\,(b \wedge (?\,\mathbf{U}\,\mathbf{G}\,c)) \in \mathrm{LTLQ}^3$ and $((b\,\mathbf{U}\,?) \wedge \mathbf{X}\neg a)\,\mathbf{U}\,a \in \mathrm{LTLQ}^4$ are exact. However, it is obviously not the case that all non-simple queries in $\overline{\mathrm{LTLQ}^x}$ are exact. For example, the query $a\,\mathbf{U}\,(\mathbf{X}\,b \wedge (?\,\mathbf{U}\,c)) \in \mathrm{LTLQ}^3$ is not simple and not exact.

5 Conclusion

In this paper, we have shown a syntactic characterization of exact/distributive LTL queries using query templates. For the proof of exactness, we have introduced stronger versions of the collecting property (strong, boundary, and intermediate collecting) which give us additional knowledge of the queries in the classes LTLQ^i. We do not only know that every query in LTLQ^x is exact, but we also know that every query in LTLQ^1 is collecting, every query in LTLQ^2 is intermediate collecting, and every query in LTLQ^7 is boundary collecting. We believe that this information can be used in order to obtain more determinism when solving queries or evaluating formulas. Although not explicitly stated in his paper, Chan's symbolic algorithm for solving CTL queries is based on this principle. In future work, we plan to experiment with optimizations of this kind.

We are currently also working on a syntactic characterization of exact CTL queries, i.e., an extension of our fragment presented in [16]. The difference to the proof method for LTL presented here is that the inductive proofs for CTL have to work on trees instead of paths which makes things much more complicated. In fact, we have found an extension of our fragment in [16] which is described by a context-free grammar consisting of 10 non-terminals. However, we could not find a proof for the maximality of this fragment yet.

References

1. Glenn Bruns and Patrice Godefroid. Temporal logic query checking. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS),* pages 409–417. IEEE Computer Society Press, 2001.

2. Francesco Buccafurri, Thomas Eiter, Georg Gottlob, and Nicola Leone. On ACTL formulas having linear counterexamples. *Journal of Computer and System Sciences,* 62(3):463–515, 2001.
3. William Chan. Temporal-logic queries. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV),* volume 1855 of *Lecture Notes in Computer Science,* pages 450–463. Springer-Verlag, 2000.
4. Marsha Chechik and Arie Gurfinkel. TLQSolver: A temporal logic query checker. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV),* volume 2725 of *Lecture Notes in Computer Science,* pages 210–214. Springer-Verlag, 2003.
5. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Workshop on Logics of Programs,* volume 131 of *Lecture Notes in Computer Science,* pages 52–71. Springer-Verlag, 1981.
6. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV),* volume 1855 of *Lecture Notes in Computer Science,* pages 154–169. Springer-Verlag, 2000. Extended version available as [7].
7. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM,* 50(5):752–794,2003.
8. Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS),* 16(5): 1521–1542, 1994.
9. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking.* MIT Press, 1999.
10. E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science,* volume B: Formal Methods and Semantics, pages 995–1067. Elsevier, 1990.
11. Arie Gurfinkel, Marsha Chechik, and Benet Devereux. Temporal logic query checking: A tool for model exploration. *IEEE Transactions on Software Engineering,* 29(10):898–914, 2003.
12. Arie Gurfinkel, Benet Devereux, and Marsha Chechik. Model exploration with temporal logic query checking. In *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE),* pages 139–148. ACM Press, 2002.
13. Samuel Hornus and Philippe Schnoebelen. On solving temporal logic queries. In *Proceedings of the 9th International Conference on Algebraic Methodology and Software Technology (AMAST),* volume 2422 of *Lecture Notes in Computer Science,* pages 163–177. Springer-Verlag, 2002.
14. Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on Programming,* volume 137 of *Lecture Notes in Computer Science,* pages 337–350. Springer-Verlag, 1982.
15. Marko Samer. PhD thesis, Vienna University of Technology. In preparation.
16. Marko Samer and Helmut Veith. Validity of CTL queries revisited. In *Proceedings of the 12th Annual Conference of the European Association for Computer Science Logic (CSL),* volume 2803 of *Lecture Notes in Computer Science,* pages 470–483. Springer-Verlag, 2003.

Online Scheduling with Bounded Migration*

Peter Sanders, Naveen Sivadasan, and Martin Skutella

Max-Planck-Institut für Informatik, Saarbrücken, Germany,
{sanders,ns,skutella}@mpi-sb.mpg.de

Abstract. Consider the classical online scheduling problem where jobs that arrive one by one are assigned to identical parallel machines with the objective of minimizing the makespan. We generalize this problem by allowing the current assignment to be changed whenever a new job arrives, subject to the constraint that the total size of moved jobs is bounded by β times the size of the arriving job.

Our main result is a linear time 'online approximation scheme', that is, a family of online algorithms with competitive ratio $1+\epsilon$ and constant migration factor $\beta(\epsilon)$, for any fixed $\epsilon > 0$. This result is of particular importance if considered in the context of sensitivity analysis: While a newly arriving job may force a complete change of the entire structure of an optimal schedule, only very limited 'local' changes suffice to preserve near-optimal solutions. We believe that this concept will find wide application in its own right.

We also present simple deterministic online algorithms with migration factors $\beta = 2$ and $\beta = 4/3$, respectively. Their competitive ratio 3/2 beats the lower bound on the performance of any online algorithm in the classical setting without migration. We also present improved algorithms and similar results for closely related problems. In particular, there is a short discussion of corresponding results for the objective to maximize the minimum load of a machine. The latter problem has an application for configuring storage servers that was the original motivation for this work.

1 Introduction

A classical scheduling problem. One of the most fundamental scheduling problems asks for an assignment of jobs to m identical parallel machines so as to minimize the makespan. (The makespan is the completion time of the last job that finishes in the schedule; it also equals the maximum machine load.) This problem is well known to be strongly NP-hard [8].

The *offline* variant of this problem assumes that all jobs are known in advance whereas in the *online* variant the jobs are incrementally revealed by an adversary

* This work was partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and by the EU Thematic Network APPOL II, Approximation and Online Algorithms, IST-2001-30012.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1111–1122, 2004.

and the online algorithm can only choose the machine for the new job without being allowed to move other jobs. Note that dropping this radical constraint on the online algorithm yields the offline situation.

A new online scheduling paradigm. We study a natural generalization of both offline and online problems. Jobs arrive incrementally but, upon arrival of a new job j, we are allowed to migrate *some* previous jobs to other machines. The total size of the migrated jobs however must be bounded by βp_j where p_j is the size of the new job. For *migration factor* $\beta = 0$ we get the online setting and for $\beta = \infty$ we get the offline setting.

Approximation algorithms. For an offline optimization problem, an *approximation algorithm* efficiently (in polynomial time) constructs schedules whose values are within a constant factor $\alpha \geq 1$ of the optimum solution value. The number α is called *performance guarantee* or *performance ratio* of the approximation algorithm. A family of polynomial time approximation algorithms with performance guarantee $1 + \epsilon$ for all fixed $\epsilon > 0$ is called a *polynomial time approximation scheme* (PTAS).

Competitive analysis. In a similar way, *competitive analysis* evaluates solutions computed in the online setting. An online algorithm achieves *competitive ratio* $\alpha \geq 1$ if it always maintains solutions whose objective values are within a factor α of the offline optimum. Here, in contrast to offline approximation results, the achievable values α are not determined by limited computing power but by the apparent lack of information about parts of the input that will only be revealed in the future. As a consequence, for all interesting classical online problems it is rather easy to come up with lower bounds that create a gap between the best possible competitive ratio α and 1. In particular, it is usually impossible to construct a family of $(1+\epsilon)$-competitive online algorithms for such problems.

Related Work

For the online machine scheduling problem, Graham's *list scheduling* algorithm keeps the makespan within a factor $2 - 1/m$ of the offline optimum [9]: Schedule a newly arriving job on the least loaded machine. It can also easily be seen that this bound is tight.

For the offline setting, Graham showed three years later that sorting the jobs in the order of non-increasing size before feeding them to the list scheduling algorithm yields an approximation algorithm with performance ratio $4/3 - 1/(3m)$. After a series of improvements, finally, a polynomial time approximation schemes for an arbitrary number of machines were given by Hochbaum and Shmoys [10].

In a series of papers, increasingly complicated online algorithms with better and better competitive ratios beating the Graham bound 2 have been developed [5,11,1]. The best result known to date is a 1.9201-competitive algorithm due to Fleischer and Wahl [7]. The best lower bound 1.88 on the competitive ratio of any deterministic online algorithm currently known is due to Rudin [12]. For randomized online algorithms there is a lower bound of $e/(e-1) \approx 1.58$ [6,

16]. For more results on online algorithms for scheduling we refer to the recent survey articles by Albers [2] and Sgall [17].

Strategies that reassign jobs were studied in the context of online load balancing, jobs arrive in and depart from a system of m machines online and the scheduler has to assign each incoming job to one of the machines. Deviating from the usual approach of comparing against the optimal *peak load* seen so far, Westbrook [18] introduced the notion of competitiveness against *current load*: An algorithm is α–competitive if after every round the makespan is within α factor of the optimal makespan for the current set of jobs. Each incoming job u has size p_u and reassignment cost r_u. For a job, the reassignment cost has to be paid for its initial assignment and then every time it is reassigned. Observe that the optimal strategy has to pay this cost once for each job for its initial assignment. Thus the optimal (re) assignment cost S is simply the sum of reassignment costs of all jobs scheduled till now. Westbrook showed a 6-competitive strategy for identical machines with reassignment cost $3S$ for proportional reassignments, i.e., r_u is proportional to p_u, and $2S$ for unit reassignments, i.e., $r_u = 1$ for all jobs. Later Andrews et al. [3] improved it to 3.5981 with the same reassignment factors. They also showed $3 + \epsilon$ and $2 + \epsilon$ competitive strategies respectively for the proportional and unit case, the reassignment factor depending only on ϵ. For arbitrary reassignment costs they achieve 3.5981 competitiveness with 6.8285 reassignment factor. They also present a 32–competitive strategy with constant reassignment factor for related machines. Job deletions is an aspect that we do not consider in our work, our focus is primarily on achieving competitive ratios close to 1. Our results can also be interpreted in this framework of online load balancing, with proportional reassignments and without job deletions. We show strategies with better competitive ratios, at the same time achieving reassignment factor strictly less than three. We also show $(1 + \epsilon)$–competitive strategies, for any $\epsilon > 0$, with constant reassignment factor $f(\epsilon)$. Our results are also stronger in the sense that a strategy with reassignment factor β ensures that when a job u arrives, the total reassignment cost incurred (for scheduling it) is at most βr_u. This is different from the more relaxed constraint that after t rounds, the total reassignment cost incurred is at most $\beta \sum r_u$ (summing over all jobs seen till round t). Most of our strategies are *robust,* they convert *any* α–competitive schedule to an α–competitive schedule after assigning the newly arrived job, whereas in [18,3] it is required that the schedule so far is carefully constructed in order to ensure the competitiveness after assigning/deleting a job in the next round.

2 Our Contribution

In Section 4 we describe a simple online algorithm which achieves approximation ratio 3/2 using a moderate migration factor $\beta = 2$. Notice that already this result beats the lower bound 1.88 (1.58) on the competitive ratio of any classical (randomized) online algorithm without migration. Using a more sophisticated analysis, the migration factor can be decreased to 4/3 while maintaining com-

petitive ratio 3/2. On the other hand we show that our approach does not allow for migration factor 1 and competitive ratio 3/2. Furthermore, an improved competitive ratio 4/3 can be achieved with migration factor 4. For two machines, we can achieve competitive ratio 7/6 with a migration factor of one. This ratio is tight for migration factor one.

Our main result can be found in Section 5. We present a family of online algorithms with competitive ratio $1+\epsilon$ and constant migration factor $\beta(\epsilon)$, for any fixed $\epsilon > 0$. On the negative side, no constant migration factor suffices to maintain competitive ratio one, i.e., optimality. We provide interpretations of these results in several different contexts:

Online algorithms. Online scheduling with bounded job migration is a relaxation of the classical online paradigm. Obviously, there is a tradeoff between the desire for high quality solutions and the requirement to compute them online, that is, to deal with a lack of information. Our result can be interpreted in terms of the corresponding tradeoff curve: Any desired quality can be guaranteed while relaxing the online paradigm only moderately by allowing for a constant migration factor.

Sensitivity analysis. Given an optimum solution to an instance of an optimization problem and a slightly modified instance, can the given solution be turned into an optimum solution for the modified instance without changing the solution too much? This is the impelling question in sensitivity analysis. As indicated above, for the scheduling problem under consideration one has to answer in the negative. Already one additional job can change the entire structure of an optimum schedule. However, our result implies that the answer is positive if we only require near-optimum solutions.

Approximation results. Our result yields a new PTAS for the scheduling problem under consideration. Due to its online background, this PTAS constructs the solution incrementally. That is, it reads the input little by little always maintaining a $(1+\epsilon)$-approximate solution. Indeed, it follows from the analysis of the algorithm that every update only takes constant time. In particular, the overall running time is linear and thus matches the previously best known approximation result.

We believe that each of these interpretations constitutes an interesting motivation for results like the one we present here in its own right and can therefore lead to interesting results for many other optimization problems.

The underlying details of the presented online approximation scheme have the same roots as the original PTAS by Hochbaum and Shmoys [10].We distinguish between small and large jobs; a job is called large if its size is of the same order of magnitude as the optimum makespan. Since this optimum can change when a new job arrives, the classification of jobs must be updated dynamically. The size of every large job is rounded such that the problem of computing an optimum schedule for the subset of large jobs can be formulated as an integer linear program of constant size. A newly arriving job causes a small change in the right hand side of this program. This enables us to use results from sensitivity analysis of integer programs in order to prove that the schedule of large jobs

needs to be changed only slightly. Our PTAS is very simple, it uses only this structural result and does not use any algorithms from integer programming theory.

In Section 6 we discuss an application of bounded migration to configuring storage servers. This was the original motivation for our work. In this application, the objective is to maximize the minimum load. It is well-known [4] that any online deterministic algorithm for this *machine covering problem* has competitive ratio at least m (the number of machines). There is also a lower bound of $\Omega(\sqrt{m})$ for any randomized online algorithm. We develop a simple deterministic online strategy which is 2-competitive already for migration factor $\beta = 1$.

Due to space limitation, proofs of some of the theorems are omitted. We refer the reader to [14] for a full version of the paper.

3 Preliminaries

Let the set of *machines* be denoted by $M = \{1, \ldots, m\}$. The set of *jobs* is $\{1, \ldots, n\}$ where job j arrives in round j. Let p_j denote the positive *processing time* or the *size* of job j. For a subset of jobs N, the *total processing time* of jobs in N is $p(N) := \sum_{j \in N} p_j$; let $p_{\max}(N) := \max_{j \in N} p_j$. For a subset of jobs N, let $\mathrm{opt}(N)$ denote the *optimal makespan*. If the subset of jobs N and a newly arrived job j are clear from the context, we sometimes also use the shorter notation $\mathrm{opt} := \mathrm{opt}(N)$ and $\mathrm{opt}' := \mathrm{opt}(N \cup \{j\})$. It is easy to observe that $\mathrm{lb}(N) := \max\{p(N)/m, p_{\max}(N)\}$ is a lower bound on $\mathrm{opt}(N)$ satisfying

$$\mathrm{lb}(N) \leq \mathrm{opt}(N) \leq 2\mathrm{lb}(N) . \tag{1}$$

The following well-known fact is used frequently in the subsequent sections.

Observation 1. *For a set of jobs N, consider an arbitrary schedule with makespan κ. Assigning a new job j to the least loaded machine yields a schedule with makespan at most* $\max\{\kappa, \mathrm{opt}(N \cup \{j\}) + (1 - 1/m)p_j\}$.

4 Strategies with Small Migration Factor

We consider the problem of scheduling jobs arriving one after another on m parallel machines so as to minimize the makespan. We first show a very simple 3/2-competitive algorithm with migration factor 2. The algorithm is as follows:
Procedure FILL_1:
Upon arrival of a new job j, choose one of the following two options minimizing the resulting makespan.

Option 1: Assign job j to the least loaded machine.
Option 2: Let i be the machine minimizing the maximum job size. Repeatedly remove jobs from this machine; stop before the total size of removed jobs exceeds $2p_j$. Assign job j to machine i. Assign the removed jobs successively to the least loaded machine.

Theorem 1. *Procedure* FILL_1 *is* $\left(\frac{3}{2} - \frac{1}{2m}\right)$*-competitive with migration factor* 2.

Proof. From the description of FILL_1, it is clear that the migration factor is at most 2. In order to prove competitiveness, we consider an arbitrary $\left(\frac{3}{2} - \frac{1}{2m}\right)$-approximate schedule for a set of jobs N and show that incorporating a new job j according to FILL_1 results in a new schedule which is still $\left(\frac{3}{2} - \frac{1}{2m}\right)$-approximate. In the following, a job is called *small* if its processing time is at most $\text{opt}'/2$, otherwise it is called *large*. If the new job j is small, then the first option yields makespan at most $\left(\frac{3}{2} - \frac{1}{2m}\right)\text{opt}'$ by Observation 1. Thus, we can assume from now on that j is large.

Since there can be at most m large jobs in $N \cup \{j\}$, all jobs on the machine chosen in the second option are small. Thus, after removing jobs from this machine as described above, the machine is either empty or the total size of removed jobs exceeds the size of the large job j. In both cases, assigning job j to this machine cannot increase its load above $\left(\frac{3}{2} - \frac{1}{2m}\right)\text{opt}'$. Thus, using the same argument as above, assigning the removed small jobs successively to the least loaded machine yields again a $\left(\frac{3}{2} - \frac{1}{2m}\right)$-approximate schedule. □

Next we show that the migration factor can be decreased to 4/3 without increasing the competitive ratio above 3/2. This result is achieved by carefully modifying FILL_1.

Procedure FILL_2 :
Upon arrival of j, choose the one of the following $m+1$ options that minimizes the resulting makespan. (Break ties in favor of option 0.)

Option 0: Assign job j to the least loaded machine.

Option i [for $i \in \{1, \ldots, m\}$]: Ignoring the largest job on machine i, consider the remaining jobs in the order of non-increasing size and remove them from the machine; stop before the total size of removed jobs exceeds $\frac{4}{3}p_j$. Assign job j to machine i. Assign the removed jobs successively to the least loaded machine.

Theorem 2. *Procedure* FILL_2 *is* $\frac{3}{2}$*-competitive with migration factor* $\frac{4}{3}$.

Robustness: Most of our scheduling strategies for minimizing the makespan discussed in this chapter are robust in the following sense. The only invariant that we require in their analyses is that before the arrival of a new job the current schedule is α-approximate. Job j can then be incorporated yielding again an α-approximate schedule. In other words, we do not require that the current schedule is carefully constructed so far, to maintain the competitiveness in the next round. Only for Procedure FILL_2, the schedule should additionally satisfy that, on any machine, the load excluding the largest job in it is at most the optimum makespan. We further remark that this is not an unreasonable requirement as even the simple list scheduling algorithm ensures this.

Negative Results

Theorems 1 and 2 raise the question of which migration factor is really necessary to achieve competitive ratio 3/2. We can prove that any robust strategy needs migration factor greater than 1 in order to maintain competitive ratio 3/2.

Lemma 1. *There exists a* 3/2*-approximate schedule such that, upon arrival of a particular job, migration factor* 1.114 *is needed to achieve* 3/2*-competitiveness. Moreover, migration factor* 1 *only allows for competitive ratio* 1.52 *in this situation.*

An additional feature of FILL_1 and FILL_2 is that they are *local* in the sense that they migrate jobs only from the machine where the newly arrived job is assigned to. There is a class of optimal schedules for which, upon arrival of a new job, it is not possible to achieve a better competitive ratio than 3/2 using only local migration. This holds even if an arbitrary migration factor is allowed. The following optimal schedule on m machines, upon the arrival of a new job, enforces a competitive ratio of at least $3/(2+\frac{2}{m})$ for any amount of migration. This bound converges to 3/2 for large m. The example looks as follows: Machines 1 and 2 each contain one job of size 1/2 and $m/2$ jobs of size $1/m$. All other machines contain a single job of size 1. The newly arriving job has size 1. The optimum makespan is $1+1/m$ and the makespan achievable by any local strategy is 3/2.

We conclude this section by stating two more results that improve the 3/2–competitive ratio.

Theorem 3. *There exists a* $\frac{4}{3}$*-competitive strategy with factor* 4 *migration.*

Theorem 4. *For the case of two machines, there exists a* $\frac{7}{6}$*-competitive strategy with factor 1 migration. Moreover, let A be any deterministic algorithm that is* c*-competitive with factor 1 migration, for two machines. Then* $c \geq \frac{7}{6(1+\epsilon)}$ *for any sufficiently small positive* $\epsilon \in \mathbb{R}^+$.

5 An Online Approximation Scheme with Constant Migration

The results presented in the last section raise the question how far the competitive ratio for online algorithms with constant migration factor can be decreased. We first prove that optimality (i.e., competitive ratio 1) cannot be achieved. However, for any fixed $\epsilon > 0$ we can get down to competitive ratio $1+\epsilon$.

Lemma 2. *Any online algorithm computing optimal solutions needs migration factor* $\Omega(m)$.

Proof (Sketch). Consider a scheduling instance with m machines and $2m-2$ jobs, two of size i/m for all $i = 1, \ldots, m-1$. Up to permutations of machines,

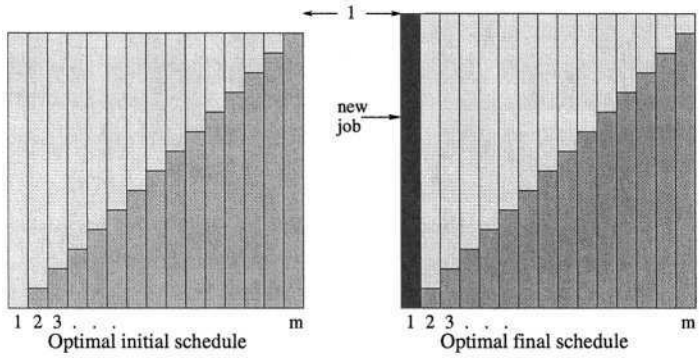


Fig. 1. An instance where all machine configurations have to change to maintain optimality.

any optimum schedule has the structure depicted in the left part of Figure 1. The optimum makespan is $(m-1)/m$. When a new job of size 1 arrives, the optimum makespan increases to 1; see the right hand side of Figure 1. A short calculation shows that the minimum total size of the jobs to be migrated is $\Omega(m)$. □

In the following, $\epsilon > 0$ is a fixed constant. We assume without loss of generality that $\epsilon \leq 1$. The following observation belongs by now to the folklore in the field of scheduling.

Observation 2. *Rounding up each job's processing time to the nearest integer power of* $1+\epsilon$ *increases the makespan of an arbitrary schedule at most by a factor* $1+\epsilon$. *In particular, in specifying a* $(1+O(\epsilon))$*-competitive algorithm we can assume that all processing times are integer powers of* $1+\epsilon$.

The current set of jobs is denoted by N. A job in N is called *large* if its processing time is at least $\epsilon \operatorname{lb}(N)$; otherwise, it is called *small*. The subset of large and small jobs is denoted by N_L and N_S, respectively. We partition N into classes N_i, $i \in \mathbb{Z}$, with

$$N_i := \{j \in N \mid p_j = (1+\epsilon)^i\} .$$

Let $I := \{i \in \mathbb{Z} \mid \epsilon \operatorname{lb}(N) \leq (1+\epsilon)^i \leq p_{\max}(N)\}$ such that $N_L = \bigcup_{i \in I} N_i$. Thus, the number of different job sizes for large jobs is bounded by $|I|$ and therefore constant:

$$|I| \leq 1 + \log_{1+\epsilon} \frac{p_{\max}(N)}{\epsilon \operatorname{lb}(N)} \leq 1 + \log_{1+\epsilon} \frac{1}{\epsilon} \leq \frac{2}{\epsilon} \log\left(\frac{1+\epsilon}{\epsilon}\right) . \qquad (2)$$

Given an assignment of jobs N_L to machines, we say that a particular machine obeys *configuration* $k : I \to \mathbb{N}_0$ if, for all $i \in I$, exactly $k(i)$ jobs from N_i are assigned to this machine. The set of configurations that can occur in any schedule for N_L is

$$\mathbf{K} := \{k : I \to \mathbb{N}_0 \mid k(i) \leq |N_i| \text{ for all } i \in I\} .$$

Up to permutations of machines, an arbitrary schedule for N_L can be described by specifying, for each $k \in \mathbf{K}$, the number y_k of machines that obey configuration k. Conversely, a vector $y \in \mathbb{N}_0^{\mathbf{K}}$ specifies a feasible m-machine-schedule for N_L if and only if

$$\sum_{k \in \mathbf{K}} y_k = m \qquad \text{and} \tag{3}$$

$$\sum_{k \in \mathbf{K}} k(i)\, y_k = |N_i| \qquad \text{for all } i \in I. \tag{4}$$

We denote the set of vectors $y \in \mathbb{N}_0^{\mathbf{K}}$ satisfying (3) and (4) by $\mathbf{S}$. Thus, $\mathbf{S}$ represents the set of all schedules (up to permutations of machines and up to permutations of equal size jobs) for N_L. For a configuration $k \in \mathbf{K}$ let

$$\text{load}(k) := \sum_{i \in I} (1+\epsilon)^i\, k(i)$$

denote the load of a machine obeying configuration k. The makespan of a schedule $y \in \mathbf{S}$ is equal to $\max\{\text{load}(k) \mid y_k > 0\}$. For $\mu \geq 0$, let

$$\mathbf{K}(\mu) := \{k \in \mathbf{K} \mid \text{load}(k) \leq \mu\} .$$

The set of all schedules with makespan at most μ is denoted by

$$\mathbf{S}(\mu) := \{y \in \mathbf{S} \mid y_k = 0 \text{ if } \text{load}(k) > \mu\} .$$

In the following, we usually interpret a schedule $y \in \mathbf{S}(\mu)$ as a vector in $\mathbb{N}_0^{\mathbf{K}(\mu)}$ by ignoring all zero-entries corresponding to configurations not contained in $\mathbf{K}(\mu)$.

The minimum makespan for N_L can be obtained by determining the minimum value μ with $\mathbf{S}(\mu) \neq \emptyset$. Checking whether $\mathbf{S}(\mu)$ is empty and, otherwise, finding a schedule $y \in \mathbf{S}(\mu)$ can be done by finding a feasible solution to an integer linear program. We can write

$$\mathbf{S}(\mu) = \{y \in \mathbb{N}_0^{\mathbf{K}(\mu)} \mid A(\mu)\, y = b\} ,$$

where $A(\mu)$ is a matrix in $\mathbb{N}_0^{(1+|I|) \times |\mathbf{K}(\mu)|}$ and b is a vector in $\mathbb{N}_0^{1+|I|}$. The first row of the linear system $A(\mu)\, y = b$ corresponds to constraint (3); the remaining $|I|$ rows correspond to constraints (4).

Lemma 3. *Let N be a set of jobs and let j be a new job of size $p_j \geq \epsilon\, \text{lb}(N)$. Any schedule for N_L with makespan $\mu \leq (1+\epsilon)\text{opt}(N)$ can be turned into a schedule for $N_L \cup \{j\}$ by touching only a constant number of machines such that the makespan of the resulting schedule is at most $\max\{\mu, \text{opt}(N_L \cup \{j\})\}$.*

Proof. We distinguish two cases. If $p_j \geq 2\mu$, then it is easy to observe that $\text{opt}(N_L \cup \{j\}) = p_j$ and an optimal schedule for $N_L \cup \{j\}$ can be obtained by assigning job j to an arbitrary machine and moving all jobs that are currently on this machine to any other machine.

In the remainder of the proof we can assume that $p_j = (1+\epsilon)^{i'} < 2\mu$ and therefore $\mathrm{opt}(N_L \cup \{j\}) \le 2\mu$. Let $y \in \mathbf{S}(\mu)$ denote the given schedule for N_L. Then, y satisfies

$$A(\mu)\, y \;=\; b \;, \qquad y \in \mathbb{N}_0^{\mathbf{K}(\mu)} \;. \tag{5}$$

Let $I' := I \cup \{i'\}$ and let $\mathbf{K}'$ denote the set of configurations $k : I' \to \mathbb{N}_0$ that can occur in any schedule for $N_L \cup \{j\}$. Then, $\mathbf{K}'(\mu)$, $\mathbf{S}'(\mu)$, $A'(\mu)$, and b' are defined analogously to $\mathbf{K}(\mu)$, $\mathbf{S}(\mu)$, $A(\mu)$, and b, respectively, with $\mathbf{K}$ replaced by $\mathbf{K}'$ and I replaced by I'.

Let $\mu' := \max\{\mu, \mathrm{opt}(N_L \cup \{j\})\} \le 2\mu$. We are looking for a schedule $y' \in \mathbf{S}'(\mu')$, that is, y' must satisfy

$$A'(\mu')\, y' \;=\; b' \;, \qquad y' \in \mathbb{N}_0^{\mathbf{K}'(\mu')} \;. \tag{6}$$

Moreover, y' should be 'similar' to y. In order to compare the two vectors, we first 'lift' y to a vector in $\mathbb{N}_0^{\mathbf{K}'(\mu')}$ as follows. A configuration $k \in \mathbf{K}(\mu)$ can be interpreted as an element of $\mathbf{K}'(\mu')$ by defining $k(i) := 0$ for all $i \in I' \setminus I$. We then define $y_k := 0$ for all $k \in \mathbf{K}'(\mu') \setminus \mathbf{K}(\mu)$. It follows from (5) that the extended vector y satisfies

$$A'(\mu')\, y \;=\; \hat{b} \;, \qquad y \in \mathbb{N}_0^{\mathbf{K}'(\mu')} \;. \tag{7}$$

The right hand side $\hat{b} \in \mathbb{N}_0^{1+|I'|}$ is defined as follows: If $I' = I$, then $\hat{b} = b$; otherwise, $I' = I \cup \{i'\}$ and we define the entry of vector $\hat{b}$ corresponding to i' to be zero and all other entries as in vector b.

Thus, y and y' are solutions to essentially the same integer linear program ((7) and (6), respectively) with slightly different right hand sides $\hat{b}$ and b', respectively. More precisely, the right hand sides are equal for all but one entry (the one corresponding to i') where they differ by 1.

Using a sensitivity result from integer linear programming (see, e.g., [15, Corollary 17.2a]), there exists a solution y' to (6) satisfying

$$\|y - y'\|_\infty \;\le\; 3\left|\mathbf{K}'(\mu')\right| \Delta \;, \tag{8}$$

where Δ is an upper bound on the absolute value of any sub-determinant of the matrix $A'(\mu')$. To complete the proof, we have to show that the right hand side of (8) is constant.

First we give an upper bound on the number of columns $|\mathbf{K}'(\mu')|$, i.e., on the number of machine configurations with load at most μ'. Since each job has size at least

$$\epsilon\, \mathrm{lb}(N) \;\overset{(1)}{\ge}\; \frac{\epsilon}{2}\,\mathrm{opt}(N) \;\ge\; \frac{\epsilon}{2(1+\epsilon)}\,\mu \;\ge\; \frac{\epsilon}{4(1+\epsilon)}\,\mu' \;,$$

there are at most $\gamma := \lfloor 4(1+\epsilon)/\epsilon \rfloor \le \frac{8}{\epsilon}$ jobs in any configuration $k \in \mathbf{K}'(\mu')$. In particular, $k(i) \le \gamma$ for all $i \in I'$. This yields

$$\left|\mathbf{K}'(\mu')\right| \;\le\; \gamma^{|I'|} \;\le\; \gamma^{|I|+1} \le \left(\frac{8}{\epsilon}\right)^{|I|+1} \;\overset{(2)}{\le}\; \left(\frac{8}{\epsilon}\right)^{\frac{3}{\epsilon}\log\left(\frac{1+\epsilon}{\epsilon}\right)} \;\le\; \left(\frac{1+\epsilon}{\epsilon}\right)^{\frac{3}{\epsilon}\log\left(\frac{8}{\epsilon}\right)} .$$

Finally, all entries in the first row of $A'(\mu')$ are 1 and the remaining entries are of the form $k(i) \leq \gamma$. Hence we bound Δ as follows. The maximum dimension of a square sub-matrix inside $A'(\mu')$ is at most the number of rows, i.e., $2 + |I|$ and, each entry in it has value at most γ. Hence the value of its determinant is upper-bounded by $((2 + |I|)\gamma)^{2+|I|}$. Thus

$$\Delta \leq \left((2 + |I|)\,\gamma\right)^{2+|I|} \overset{(2)}{\leq} \left(\frac{8}{\epsilon^2}\right)^{2+|I|} \cdot \left(\frac{8}{\epsilon}\right)^{2+|I|} \overset{(2)}{\leq} \left(\frac{1+\epsilon}{\epsilon}\right)^{\frac{12}{\epsilon}\log(\frac{4}{\epsilon})}$$

Thus the number of machines touched, which is at most $3\left|\mathbf{K}'(\mu')\right|\Delta$ by (8), is a constant. This concludes the proof. □

Theorem 5. *Let N be a set of jobs and let j be a new job not contained in N. Any $(1+\epsilon)$-approximate schedule for N can be turned into a $(1+\epsilon)$-approximate schedule for $N \cup \{j\}$ such that the total size of jobs that have to be moved is bounded by a constant $\beta(\epsilon)$ times p_j.*

Theorem 6. *There exists a $(1+\epsilon)$-competitive online algorithm with constant migration factor $\beta(\epsilon)$ such that the running time for scheduling a newly arrived job is constant.*

In particular, it follows from the last property mentioned in the theorem that the algorithm has linear running time.

6 Maximizing the Minimum Machine Load

An alternative, yet less frequently used objective for machine scheduling is to maximize the minimum load. However, we have a concrete application using this objective function that was the original motivation for our interest in bounded migration: Storage area networks (SAN) usually connect many disks of different capacity and grow over time. A convenient way to hide the complexity of a SAN is to treat it as a single big, fault tolerant disk of huge capacity and throughput. A simple scheme with many nice properties implements this idea if we manage to partition the SAN into several sub-servers of about equal size [13]. Mapping to the scheduling paradigm, the contents of disks correspond to jobs and the sub-servers correspond to machines. Each sub-server stores the same amount of data. For example, if we have two sub-servers, each of them stores all the data to achieve a fault tolerance comparable to mirroring in ordinary RAID level 0 arrays. More sub-servers allow for a more flexible tradeoff between fault tolerance, redundancy, and access granularity. In any case, the capacity of the server is determined by the *minimum* capacity of a sub-server. Moreover, it is not acceptable to completely reconfigure the system when a new disk is added to the system or when a disk fails. Rather, the user expects a "proportionate response", i.e., if she adds a disk of x GByte she will not be astonished if the system moves data of this order of magnitude but she would complain if much more is moved.

Our theoretical investigation confirms that this 'common sense[1] expectation is indeed reasonable. For the case without disk failures (job departures) we obtain the following result using a simple greedy strategy.

Theorem 7. *There exists a* 2*–competitive strategy with migration factor* 1.

Acknowledgments. We thank Gerhard Woeginger for interesting discussions and helpful comments on the topic of this paper. We also thank the referees for pointing us to [3,18].

References

1. S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing,* 29:459–473, 1999.
2. S. Albers. Online algorithms: a survey. *Mathematical Programming,* 97:3–26, 2003.
3. M. Andrews, M.X. Goemans, and L. Zhang. Improved bounds for on-line load balancing. *Algorithmica,* 23:278–301, 1999.
4. Y. Azar and L. Epstein. On-line machine covering. *Journal of Algorithms,* 1:67–77, 1998.
5. Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences,* 51:359–366, 1995.
6. B. Chen, A. van Vliet, and G. J. Woeginger. Lower bounds for randomized online scheduling. *Information Processing Letters,* 51:219–222, 1994.
7. R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling,* 3:343–353, 2000.
8. M. R. Garey and D. S. Johnson. Strong np-completeness results: Motivation, examples and implications. *Journal of the ACM,* 25:499–508, 1978.
9. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal,* 45:1563–1581, 1966.
10. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM,* 34:144–162, 1987.
11. D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms,* 20:400–430, 1996.
12. J. F. Rudin III. *Improved bounds for the on-line scheduling problem.* PhD thesis, The University of Texas at Dallas, 2001.
13. P. Sanders. Algorithms for scalable storage servers. In *SOFSEM 2004: Theory and Practice of Computer Science,* volume 2932, pages 82–101, 2004.
14. P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. Research Report MPI-I-2004-1-004, Max-Planck-Institut für Informatik, Saarbrücken, Germany, April 2004.
15. A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester, 1986.
16. J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters,* 63(1):51–55, 14 July 1997.
17. J. Sgall. On-line scheduling — a survey. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art,* volume 1442 of *Lecture Notes in Computer Science,* pages 196–231. Springer, Berlin, 1998.
18. J. Westbrook. Load balancing for response time. *J. Algorithms,* 35(1):1–16, 2000.

On the Expressive Power of Monadic Least Fixed Point Logic

Nicole Schweikardt*

Institut für Informatik, Humboldt-Universität Berlin
Unter den Linden 6, D-10099 Berlin, Germany
schweika@informatik.hu-berlin.de
http://www.informatik.hu-berlin.de/ schweika

Abstract. Monadic least fixed point logic MLFP is a natural logic whose expressiveness lies between that of first-order logic FO and monadic second-order logic MSO. In this paper we take a closer look at the expressive power of MLFP. Our results are

1. MLFP can describe graph properties beyond any fixed level of the monadic second-order quantifier alternation hierarchy.
2. On strings with built-in addition, MLFP can describe at least all languages that belong to the linear time complexity class DLIN.
3. Settling the question whether

   addition-invariant MLFP $\stackrel{?}{=}$ addition-invariant MSO on finite strings

   would solve open problems in complexity theory: "=" would imply that PH = PTIME whereas "$\neq$" would imply that $\mathbf{DLIN} \neq \mathbf{LINH}$.

1 Introduction

A central topic in Finite Model Theory has always been the comparison of the expressive power of different logics on finite structures. One of the main motivations for such studies is an interest in the expressive power of query languages for *relational databases* or for *semi-structured data* such as XML-documents. Relational databases can be modeled as finite relational structures, whereas XML-documents can be modeled as finite labeled trees. Since *first-order logic* FO itself is too weak for expressing many interesting queries, various extensions of FO have been considered as query languages.

When restricting attention to strings and labeled trees, *monadic second-order logic* MSO seems to be "just right": it has been proposed as a yardstick for expressiveness of XML query languages [7] and, due to its connection to finite automata (cf., e.g., [25]), the model-checking problem for (Boolean and unary) MSO-queries on strings and labeled trees can be solved with polynomial time data complexity (cf., e.g., [6]). On finite relational structures in general, however, MSO can express complete problems for

* Parts of this work were done while the author was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD) in order to visit the Laboratory for Foundations of Computer Science, University of Edinburgh, Scotland, U.K.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1123–1135, 2004.

all levels of the polynomial time hierarchy [1], i.e., MSO can express queries that are believed to be far too difficult to allow efficient model-checking.

The main focus of the present paper lies on *monadic least fixed point logic* MLFP, which is an extension of first-order logic by a mechanism that allows to define unary relations *by induction.* Precisely, MLFP is obtained by restricting the least fixed point logic FO(LFP) (cf., e.g., [16,5]) to formulas in which only *unary* relation variables are allowed. The expressive power of MLFP lies between the expressive power of FO and the expressive power of MSO. On finite relational structures in general, MLFP has the nice properties that (1) the model-checking problem can be solved with polynomial time and linear space data complexity, and (2) MLFP is "on-spot" for the description of many important problems. For example, the transitive closure of a binary relation, or the set of winning positions in the geography game (cf., [5, Exercise 8.1.10]) can be specified by MLFP-formulas. And on strings and labeled trees, MLFP even has exactly the same expressiveness as MSO (with respect to Boolean and unary queries, cf., [25,7]). But for all that, the logic MLFP has received surprisingly little attention in recent years. Considerably more attention has already been paid to monadic fixed point extensions of *propositional modal logic,* which are used as languages for hardware and process specification and verification. A particularly important example of such a logic is the *modal* μ*-calculus* (cf., e.g., [2]), which can be viewed as the modal analogue of MLFP. *Monadic datalog,* the monadic fixed point extension of *conjunctive queries* (a subclass of FO), has recently been proposed as a database and XML query language that has a good trade-off between the expressive strength, on the one hand, and the complexity of query evaluation, on the other hand [7]. On relational structures in general, however, neither monadic datalog nor the modal μ-calculus can express all of FO, whereas all 3 logics are included in MLFP.

As already mentioned, the expressive power of MLFP ranges between that of FO and that of MSO. Dawar [3] has shown that *3-colorability* of finite graphs is not definable in infinitary logic $L^{\omega}_{\infty\omega}$. Since all of MLFP can be expressed in $L^{\omega}_{\infty\omega}$, this implies that the (NP-complete) 3-colorability problem is definable in MSO (even, in *existential* monadic second-order logic $\mathbf{Mon}\Sigma^1_1$), but not in MLFP. Grohe [13] also exposed a *polynomial time solvable* graph problem that is not MLFP-definable, but that is definable in FO(LFP) and, as a closer inspection shows, also in MSO. Both results show that on finite graphs MLFP is strictly less expressive than MSO. The first main result of the present paper states that, nevertheless, MLFP has a certain expressive strength, as it can define graph problems beyond any fixed level of the monadic second-order quantifier alternation hierarchy:

Theorem 1.1. *For each* $k \geqslant 1$ *there is an MLFP-definable graph problem that does not belong to the* k*-th level of the monadic second-order quantifier alternation hierarchy.*

When shifting attention from finite graphs to finite strings or labeled trees, the picture is entirely different: there, MLFP, MSO, and $\mathbf{Mon}\Sigma^1_1$ have the same expressive power, namely, of expressing exactly the *regular* languages (cf., [25]). To increase the expressive power of MLFP and MSO on the class of finite strings, one can allow formulas to also use the ternary relation + which is interpreted as the graph of the addition function. For a logic $\mathcal{L}$ we write $\mathcal{L}(+)$ to explicitly indicate that the addition predicate + may be used in $\mathcal{L}$-formulas. In [19] Lynch has shown that $\mathbf{NTIME}(n) \subseteq \mathbf{Mon}\Sigma^1_1(+)$ on the

class of finite strings with built-in addition. I.e., every string-language decidable by a nondeterministic multi-tape Turing machine with linear time bound is definable by a sentence in $\mathbf{Mon}\Sigma_1^1(+)$. Building upon this, one can show (cf., [21]) that MSO(+) = LINH, i.e., MSO(+) can define exactly those string-languages that belong to the *linear time hierarchy* (which is the linear time analogue of Stockmeyer's polynomial time hierarchy). Lynch's result was strengthened by Grandjean and Olive [11]: They showed that $\mathbf{Mon}\Sigma_1^1(+)$ can even define all string-languages that belong to the complexity class NLIN. The class NLIN and its deterministic version DLIN are based on linear time random access machines and were introduced by Grandjean in a series of papers [8,9, 10]. As argued in [10], DLIN and NLIN can be seen as "the" adequate mathematical formalizations of linear time complexity. For example, NLIN contains $\mathbf{NTIME}(n)$ and all 21 NP-complete problems listed by Karp in [17]. The class DLIN contains all problems in $\mathbf{DTIME}(n)$, i.e. all problems decidable in linear time by a deterministic multi-tape Turing machine. But DLIN also contains problems such as CHECKSORT (given two lists $\ell_1 = s_1, .., s_n$ and $\ell_2 = t_1, .., t_n$ of strings, decide whether ℓ_2 is the lexicographically sorted version of ℓ_1) which are conjectured not to belong to $\mathbf{DTIME}(n)$ (see [24]). In the present paper we show the following analogue of the result of [11]:

Theorem 1.2. *All string-languages that belong to the linear time complexity class* DLIN *are definable in* MLFP(+).

One area of research in Finite Model Theory considers extensions of logics which allow *invariant* uses of some auxiliary relations. For example, *order-invariant* formulas may use a linear ordering of a given structure's universe, but they must not depend on the particular choice of linear ordering. This corresponds to the "real world" situation where the physical representation of a graph or a database, stored in a computer, induces a linear order on the vertices of the graph or the tuples in the database. But this particular order is hidden to the user, because one wants the user's queries to be independent of the particular physical representation of the data. Therefore, for formulating queries, the user may be allowed to use the fact that *some* order is there, but he cannot make his queries depend on any *particular* order, because he does not know which order the data comes with. Similarly, *successor-* or *addition*-invariant formulas may use a successor-relation or an addition-relation on a structure's universe, but must be independent of the particular choice of successor- or addition-relation. Such kinds of invariance have been investigated with respect to first-order logic, e.g., in [15,22,4]. In the present paper we consider *addition-invariant* formulas on finite strings and show that both, the *equivalence* of addition-invariant MLFP and MSO, as well as a *separation* of addition-invariant MLFP from MSO would solve open problems in complexity theory: Let PH denote Stockmeyer's *polynomial time hierarchy,* and let LINH be the *linear time hierarchy,* i.e., the linear time analogue of PH.

Theorem 1.3. *(a) If addition-invariant* MLFP $\neq$ *addition-invariant* MSO *on the class of finite strings, then* DLIN $\neq$ LINH.

(b) If addition-invariant MLFP = *addition-invariant* MSO *on the class of finite strings, then* PH = PTIME.

In other words, it is most likely that addition-invariant **MLFP** $\neq$ **addition-invariant** MSO on strings, but actually *proving* this can be expected to be rather difficult, since it would imply the separation of the complexity class DLIN from the linear time hierarchy LINH.

The paper is structured as follows: Section 2 fixes the basic notations and gives an example of the present paper's use of MLFP-formulas. Theorem 1.1 is proved in Section 3. Section 4 concentrates on the proof of Theorem 1.2. Section 5 deals with the proof of Theorem 1.3. Some open questions are pointed out in Section 6. Due to space limitations, detailed proofs had to be defered to the full version of this paper [23].

2 Preliminaries

For an alphabet $\mathbb{A}$ we write $\mathbb{A}^+$ to denote the set of all finite non-empty strings over $\mathbb{A}$. For a set $\mathcal{U}$ we write $2^{\mathcal{U}}$ to denote the power set of $\mathcal{U}$, i.e., $2^{\mathcal{U}} := \{X : X \subseteq \mathcal{U}\}$. We use $\mathbb{N}$ to denote the set $\{0,1,2,..\}$ of natural numbers. For every $n \in \mathbb{N}$ we write $[n]$ for the set $\{0,..,n-1\}$. The logarithm of n with respect to base 2 is denoted $\lg n$.

A (relational) *signature* τ is a finite set of relation symbols. Each relation symbol $R \in \tau$ has a fixed arity *ar(R)*. A τ-**structure** $\mathcal{A}$ consists of a set $\mathcal{U}^{\mathcal{A}}$ called the *universe* of $\mathcal{A}$, and an interpretation $R^{\mathcal{A}} \subseteq (\mathcal{U}^{\mathcal{A}})^{ar(R)}$ of each relation symbol $R \in \tau$.
All structures considered in this paper are assumed to have a finite universe.

We assume that the reader is familiar with *first-order logic* FO, *monadic second-order logic* MSO, and *least fixed point logic* FO(LFP) (cf., e.g., the textbooks [5,16]). The k-**th** level, $\mathbf{Mon}\Sigma^1_k$, of the monadic second-order quantifier alternation hierarchy consists of all MSO-formulas that are in prenex normal form, having a prefix of k alternating blocks of set quantifiers, starting with an existential block, and followed by a first-order formula.

For a logic $\mathcal{L}$ we use $\mathcal{L}(\tau)$ to denote the class of all $\mathcal{L}$-**formulas** of signature τ. We write $\varphi(x_1,..,x_k,X_1,..,X_\ell)$ to indicate that the free first-order variables of the formula φ are $x_1,..,x_k$ and the free second-order variables are $X_1,..,X_\ell$. Sometimes we use $\overline{x}$ and $\overline{X}$ as abbreviations for sequences $x_1,..,x_k$ and $X_1,..,X_\ell$ of variables. A *sentence* φ of signature τ is a formula that has no free variable.

Let τ be a signature, let $\mathcal{C}$ be a class of τ-**structures**, and let $\mathcal{L}$ be a logic. We say that a set $L \subseteq \mathcal{C}$ is $\mathcal{L}$-*definable in* $\mathcal{C}$ if there is a sentence $\varphi \in \mathcal{L}(\tau)$ such that $L = \{\mathcal{A} \in \mathcal{C} : \mathcal{A} \models \varphi\}$. Similarly, we say that a set L of τ-**structures** is $\mathcal{L}$-**definable**, iff it is $\mathcal{L}$-**definable** in the class of all (finite) τ-**structures**.

We will mainly consider the *monadic least fixed point logic* MLFP, which is the restriction of least fixed point logic FO(LFP), where fixed point operators are required to be *unary*. For the precise definition of MLFP we refer the reader to the textbook [5] (MLFP is denoted FO(M-LFP) there). *Simultaneous monadic least fixed point logic* S-MLFP is the extension of MLFP by operators that allow to compute the simultaneous least fixed point of several unary operators. In other words: S-MLFP is obtained by restricting simultaneous least fixed point logic FO(S-LFP) to unary fixed point relations. For the formal definition of FO(S-LFP) we, again, refer to [5]. The following example illustrates the present paper's use of S-MLFP-formulas.

Example 2.1. Let $\tau_{<,+}$ be the signature that consists of a binary relation symbol $<$ and a ternary relation symbol $+$. For every $n \in \mathbb{N}$ let $\mathcal{A}_n$ be the $\tau_{<,+}$-**structure** with universe

$[n] = \{0, .., n-1\}$, where $<$ is interpreted by the natural linear ordering and $+$ is interpreted by the graph of the addition function, i.e., $+$ consists of all triples (a, b, c) over $[n]$ where $a+b = c$. Consider the formulas

$$\begin{aligned}\varphi_S(x, S, P) \;:=\; & \text{“}x{=}0\text{”} \vee \text{“}x{=}1\text{”} \vee \\ & \exists x_1 \exists x_2 \Big(x_1{<}x_2 \wedge x_2{<}x \wedge S(x_1) \wedge S(x_2) \wedge \\ & \qquad \forall z \big((x_1{<}z \wedge z{<}x_2) \rightarrow P(z) \big) \wedge \text{“}x{-}x_2 = x_2{-}x_1{+}2\text{”} \Big)\end{aligned}$$

$$\begin{aligned}\varphi_P(y, S, P) \;:=\; & \exists x_1 \exists x_2 \Big(x_1{<}x_2 \wedge x_2{<}y \wedge S(x_1) \wedge S(x_2) \wedge \\ & \qquad \forall z \big((x_1{<}z \wedge z{<}x_2) \rightarrow P(z) \big) \wedge \text{“}y{-}x_2 < x_2{-}x_1{+}2\text{”} \Big).\end{aligned}$$

Of course, the subformulas written in quotation marks “⋯” can easily be resolved by proper $\mathrm{FO}(\tau_{<,+})$-formulas. In the structure $\mathcal{A}_n$, the simultaneous least fixed point $(S^{(\infty)}_{\mathcal{A}_n}, P^{(\infty)}_{\mathcal{A}_n})$ of (φ_S, φ_P) is evaluated as follows: We start with the 0-th stage, where S and P are interpreted by the sets $S^{(0)}_{\mathcal{A}_n} = P^{(0)}_{\mathcal{A}_n} = \emptyset$. Inductively, for every $i \in \mathbb{N}$, the $(i{+}1)$-st stage is obtained via

$$\begin{aligned} S^{(i+1)}_{\mathcal{A}_n} &:= \{ a \in [n] : \mathcal{A}_n \models \varphi_S(a, S^{(i)}_{\mathcal{A}_n}, P^{(i)}_{\mathcal{A}_n}) \} \\ P^{(i+1)}_{\mathcal{A}_n} &:= \{ b \in [n] : \mathcal{A}_n \models \varphi_P(b, S^{(i)}_{\mathcal{A}_n}, P^{(i)}_{\mathcal{A}_n}) \}. \end{aligned}$$

In particular,

$$\begin{aligned} & S^{(1)}_{\mathcal{A}_n} = \{0,1\}, \quad S^{(2)}_{\mathcal{A}_n} = \{0,1,4\}, \quad S^{(3)}_{\mathcal{A}_n} = \{0,1,4,9\}, \quad S^{(4)}_{\mathcal{A}_n} = \{0,1,4,9,16\}, \\ & P^{(1)}_{\mathcal{A}_n} = \emptyset, \quad P^{(2)}_{\mathcal{A}_n} = \{2,3\}, \quad P^{(3)}_{\mathcal{A}_n} = \{2,3,5,6,7,8\} \quad \cdots \quad . \end{aligned}$$

At some stage i (with $i \leqslant n$), this process arrives at a fixed point, i.e., at a situation where $S^{(i)}_{\mathcal{A}_n} = S^{(i+1)}_{\mathcal{A}_n} = S^{(j)}_{\mathcal{A}_n}$ and $P^{(i)}_{\mathcal{A}_n} = P^{(i+1)}_{\mathcal{A}_n} = P^{(j)}_{\mathcal{A}_n}$, for every $j > i$. This particular tuple $(S^{(i)}_{\mathcal{A}_n}, P^{(i)}_{\mathcal{A}_n})$ is called the *simultaneous least fixed point* $(S^{(\infty)}_{\mathcal{A}_n}, P^{(\infty)}_{\mathcal{A}_n})$ of (φ_S, φ_P) in $\mathcal{A}_n$. It is not difficult to see that for our example formulas φ_S and φ_P we obtain that $S^{(\infty)}_{\mathcal{A}_n}$ is the set of all square numbers in $[n]$, whereas $P^{(\infty)}_{\mathcal{A}_n}$ is the set of all non-square numbers in $[n]$.

Now, $[\text{S-LFP}_{x,S,y,P}\varphi_S, \varphi_P]_S(u)$ is an S-MLFP-formula that is satisfied by exactly those elements u in $\mathcal{A}_n$'s universe that belong to $S^{(\infty)}_{\mathcal{A}_n}$, i.e., that are square numbers. Similarly, $[\text{S-LFP}_{x,S,y,P}\varphi_S, \varphi_P]_P(u)$ is an S-MLFP-formula that is satisfied by those elements u in $\mathcal{A}_n$'s universe that belong to $P^{(\infty)}_{\mathcal{A}_n}$, i.e., that are non-square numbers.

In the above example we have seen that, given the addition relation $+$, the set of square numbers is definable in S-MLFP. It is known (cf., e.g., [14, Corollary 5.3]) that MLFP has the same expressive power as S-MLFP. Since S-MLFP-definitions of certain properties or relations are sometimes easier to find and more convenient to read than equivalent MLFP-definitions, we will often present S-MLFP-definitions instead of MLFP-definitions.

3 MLFP and the MSO Quantifier Alternation Hierarchy

In this section we show that MLFP can define graph problems beyond any fixed level of the monadic second-order quantifier alternation hierarchy.

Let τ_{graph} be the signature that consists of a binary relation symbol E. We write $\mathcal{C}_{\text{graphs}}$ for the class of all finite directed graphs. A graph $G = \langle V^G, E^G \rangle$ is called *undirected* if the following is true: for every $v \in V^G$, $(v,v) \notin E^G$, and for every $(v,w) \in E^G$, also $(w,v) \in E^G$. We write $\mathcal{C}_{\text{ugraphs}}$ to denote the class of all finite undirected graphs. Let $\tau_{\text{grid}} := \{S_1, S_2\}$ be a signature consisting of two binary relation symbols. The *grid* of height m and width n is the τ_{grid}-structure

$$\underline{[m,n]} := \langle \{1,..,m\} \times \{1,..,n\}, S_1^{m,n}, S_2^{m,n} \rangle ,$$

where $S_1^{m,n}$ is the "vertical" successor relation consisting of all tuples $\big((i,j),(i+1,j)\big)$ in $\{1,..,m\} \times \{1,..,n\}$, and $S_2^{m,n}$ is the "horizontal" successor relation consisting of all tuples $\big((i,j),(i,j+1)\big)$. We define $\mathcal{C}_{\text{grids}} := \{\underline{[m,n]} : m,n \geqslant 1\}$ to be the class of all finite grids. It was shown in [20] that the monadic second-order quantifier alternation hierarchy is *strict* on the class of finite graphs and the class of finite grids. In the present paper we will use the following result:

Theorem 3.1 ([20]). *For every $k \geqslant 1$ there is a set L_k of finite grids such that L_k is definable in $\mathrm{Mon}\Sigma^1_k$ but not in $\mathrm{Mon}\Sigma^1_{k-1}$ (on the class of finite grids).*

Using the construction of [20] and the fact that MLFP is as expressive as S-MLFP, it is an easy (but tedious) exercise to show the following

Corollary 3.2. *For every $k \geqslant 1$ the set L_k is definable in* MLFP *and in $\mathrm{Mon}\Sigma^1_k$, but not in $\mathrm{Mon}\Sigma^1_{k-1}$ (on the class of finite grids).*

Note that the above corollary deals with structures over the signature τ_{grid} that consists of two binary relation symbols. In the remainder of this section we will transfer this to the classes $\mathcal{C}_{\text{graphs}}$ and $\mathcal{C}_{\text{ugraphs}}$. To this end, we need a further result of [20] which uses the notion of *strong first-order reductions*. The precise definition of this notion is of no particular importance for the present paper. What *is* important is that a strong first-order reduction from a class $\mathcal{C}$ of τ-structures to a class $\mathcal{C}'$ of τ'-structures is an injective mapping $\Phi : \mathcal{C} \to \mathcal{C}'$ such that every structure $\mathcal{A} \in \mathcal{C}$ can be interpreted in the structure $\Phi(\mathcal{A})$ and, vice versa, $\Phi(\mathcal{A})$ can be interpreted in $\mathcal{A}$. The fundamental use of strong first-order reductions comes from the following result:

Theorem 3.3 ([20, Theorem 33]). *Let $\mathcal{C}$ and $\mathcal{C}'$ be classes of structures over the relational signatures τ and τ', respectively. Let Φ be a strong first-order reduction from $\mathcal{C}$ to $\mathcal{C}'$. Let $\mathcal{L}$ be one of the logics $\mathrm{Mon}\Sigma^1_k$, for some $k \geqslant 0$. Let the image $\Phi(\mathcal{C}) := \{\Phi(\mathcal{A}) : \mathcal{A} \in \mathcal{C}\}$ of Φ be $\mathcal{L}$-definable in $\mathcal{C}'$. Then, the following is true for every $L \subseteq \mathcal{C}$:*

$$L \text{ is } \mathcal{L}\text{-definable in } \mathcal{C} \iff \Phi(L) \text{ is } \mathcal{L}\text{-definable in } \mathcal{C}'.$$

In the present paper, the following strong first-order reductions will be used:

Proposition 3.4 ([20, Proposition 38]).

(a) There exists a strong first-order reduction Φ_1 *from* $\mathcal{C}_{grids}$ *to* $\mathcal{C}_{graphs}$, *and the image* $\Phi_1(\mathcal{C}_{grids})$ *of* Φ_1 *is* $\mathrm{Mon}\Sigma^1_2$*-definable and* MLFP*-definable in* $\mathcal{C}_{graphs}$.

(b) There exists a strong first-order reduction Φ_2 *from* $\mathcal{C}_{graphs}$ *to* $\mathcal{C}_{ugraphs}$, *and the image* $\Phi_2(\mathcal{C}_{graphs})$ *of* Φ_2 *is* FO*-definable in* $\mathcal{C}_{ugraphs}$.

This directly allows to transfer Theorem 3.1 from finite grids to finite graphs and finite undirected graphs, respectively. To also transfer Corollary 3.2 from $\mathcal{C}_{\mathrm{grids}}$ to $\mathcal{C}_{\mathrm{graphs}}$ and $\mathcal{C}_{\mathrm{ugraphs}}$, we need the following easy lemma:

Lemma 3.5. *Let* $\mathcal{C}$ *and* $\mathcal{C}'$ *be classes of structures over the relational signatures* τ *and* τ', *respectively. Let* Φ *be a strong first-order reduction from* $\mathcal{C}$ *to* $\mathcal{C}'$. *Every* $\mathrm{MLFP}(\tau)$*-sentence* ψ *can be translated into an* $\mathrm{MLFP}(\tau')$*-sentence* ψ' *such that, for every* $\mathcal{A} \in \mathcal{C}$, $\mathcal{A} \models \psi \iff \Phi(\mathcal{A}) \models \psi'$.

Using this, it is not difficult to prove this section's main result:

Theorem 3.6. *For every* $k \geqslant 2$ *there is a set* D_k *of finite directed graphs (respectively, a set* U_k *of finite undirected graphs) such that* D_k *(respectively,* U_k*) is definable in* MLFP *and* $\mathrm{Mon}\Sigma^1_k$, *but not in* $\mathrm{Mon}\Sigma^1_{k-1}$.

4 MLFP and Linear Time Complexity

We identify a string $w = w_0 \cdots w_{n-1}$ of length $|w| = n \geqslant 1$ over an alphabet $\mathbb{A}$ with a structure $\underline{w}$ in the usual way: We choose $\tau_{\mathbb{A}}$ to consist of the binary relation symbol $<$ and a unary relation symbol P_a, for each letter $a \in \mathbb{A}$. We choose $\underline{w}$ to be the $\tau_{\mathbb{A}}$-structure $\langle\{0,..,n-1\}, <, (P^w_a)_{a\in\mathbb{A}}\rangle$, where $<$ denotes the natural linear ordering of $[n] = \{0,..,n-1\}$ and P^w_a consists of all positions of w that carry the letter a.
In this section we equip the structure $\underline{w}$ with an additional ternary addition relation $+$. I.e., we identify the string w with the structure $\langle \underline{w}, + \rangle := \langle [n], <, +, (P^w_a)_{a\in\mathbb{A}}\rangle$, where $+$ consists of all triples $(a,b,c) \in [n]^3$ with $a + b = c$. We identify the set $\mathbb{A}^+$ of all non-empty strings over alphabet $\mathbb{A}$ with the set $\mathcal{C}_{\mathbb{A}} := \{\underline{w} : w \in \mathbb{A}^+\}$, respectively, with the set $\mathcal{C}_{\mathbb{A},+} := \{\langle \underline{w}, + \rangle : w \in \mathbb{A}^+\}$.

To give the precise definition of Grandjean's linear time complexity class DLIN, we need the following notion of *random access machines,* basically taken from [12].
A DLIN-RAM $\mathcal{R}$ is a random access machine that consists of two *accumulators* A and B, a *special register* M, *registers* R_i, for every $i \in \mathbb{N}$, and a *program* that is a finite sequence $\mathcal{I}(1), \ldots, \mathcal{I}(r)$ of *instructions,* each of which is of one of the following forms:

- $A := 0$,
- $A := 1$,
- $A := M$,
- $A := A + B$,
- $A := R_A$,
- $R_A := B$,
- $M := A$,
- $B := A$,
- IF $A{=}B$ THEN $\mathcal{I}(i_0)$ ELSE $\mathcal{I}(i_1)$,
- HALT.

The meaning of most of these instructions is straightforward. If A contains a number i, then the execution of the instruction $A := R_A$ copies the content of register R_i into the accumulator A. Similarly, the execution of the instruction $R_A := B$ copies the content

of accumulator B into register R_i. We stipulate that the last instruction, $\mathcal{I}(r)$, is the instruction HALT.

The input to $\mathcal{R}$ is assumed to be present in the first registers of $\mathcal{R}$ at the beginning of the computation. Precisely, an *input to* $\mathcal{R}$ is a function $f : [m] \to [m]$, for an arbitrary $m \in \mathbb{N}$. The initial content of the special register M is the number m, and for every $i \in \mathbb{N}$, the initial content of register R_i is $f(i)$ if $i \in [m]$, and 0 otherwise. The accumulators A and B are initialized by 0. The computation of $\mathcal{R}$ starts with instruction $\mathcal{I}(1)$ and finishes when it encounters a HALT statement. We say that $\mathcal{R}$ *accepts* an input f, if the content of register R_0 is non-zero when $\mathcal{R}$ reaches a HALT statement.
$\mathcal{R}$ *recognizes* a set $\mathcal{F} \subseteq \{f : [m] \to [m] : m \in \mathbb{N}\}$ *in time* $\mathcal{O}(m)$, if

1. $\mathcal{R}$ accepts an input f if, and only if, $f \in \mathcal{F}$, and
2. there is a number $d \in \mathbb{N}$ such that $\mathcal{R}$ is ***d-bounded***, i.e., for every $m \in \mathbb{N}$ and every $f : [m] \to [m]$ the following is true: when started with input f, $\mathcal{R}$ performs less than $d \cdot m$ computation steps before reaching a HALT statement, and throughout the computation, each register and each accumulator contains numbers of size $< d \cdot m$.

To use DLIN-RAMs for recognizing *string-languages,* one represents strings w by functions f_w as follows (cf., [11]). W.l.o.g. we restrict attention to strings over the alphabet $\mathbb{A} := \{1, 2\}$. For every $n \geqslant 1$ we define $\ell(n) := \lceil \frac{1}{2} \lg(n+1) \rceil$ and $m(n) := \lceil \frac{n}{\ell(n)} \rceil$. A string w over $\mathbb{A} = \{1, 2\}$ of length n can (uniquely) be decomposed into substrings $w_0, w_1, \ldots, w_{m(n)-1}$ such that

- w is the concatenation of the strings $w_0, \ldots, w_{m(n)-1}$,
- w_i has length $\ell(n)$, for every $i < m(n)-1$, and
- $w_{m(n)-1}$ has length at most $\ell(n)$.

For each $i \in [m(n)]$ let w_i^{dy} be the integer whose dyadic representation is w_i. I.e., if $w_i = d_0 \cdots d_{\ell(n)-1}$ with $d_j \in \{1, 2\}$, then $w_i^{dy} = \sum_{j<\ell(n)} d_j \cdot 2^j$. It is straightforward to see that $w_i^{dy} < m(n)$. Now, w is represented by the function $f_w : [m(n)] \to [m(n)]$ with $f_w(i) := w_i^{dy}$, for every $i \in [m(n)]$.

Definition 4.1 (DLIN, [10]). *A string-language L over alphabet $\mathbb{A} = \{1, 2\}$ belongs to the complexity class* DLIN *if, and only if, the set of its associated functions $\{f_w : w \in L\}$ is recognized by a DLIN-RAM in time $\mathcal{O}(m)$.*

At first sight, the class DLIN may seem a bit artificial: a string w of length n is represented by a function f_w of domain $[m(n)]$ where $m(n)$ is of size $\Theta(\frac{n}{\lg n})$. A DLIN-RAM with input f_w is allowed to perform only $\mathcal{O}(\frac{n}{\lg n})$ computation steps, with register contents of size $\mathcal{O}(\frac{n}{\lg n})$. However, as argued in [8,9,10,12], DLIN is a very reasonable formalization of the intuitive notion of "linear time complexity". In particular, DLIN contains all string-languages recognizable by a deterministic Turing machine in $\mathcal{O}(n)$ steps, and, in addition, also some problems (such as CHECKSORT, cf., Section 1) that are conjectured not to be solvable by Turing machines with time bound $\mathcal{O}(n)$.

Grandjean and Olive [11] showed that $\mathbf{Mon}\Sigma_1^1(+)$ can define (at least) all string-languages that belong to the nondeterministic version NLIN of DLIN. In the remainder of this section we show the following analogue of the result of [11]:

Theorem 4.2 (DLIN $\subseteq$ MLFP(+) on finite strings with built-in addition). *For every finite alphabet $\mathbb{A}$ and every string-language $L \subseteq \mathbb{A}^+$ in* DLIN *there is an* MLFP($\tau_{\mathbb{A}} \cup \{+\}$)-*sentence* φ_L *such that, for every* $w \in \mathbb{A}^+$, $w \in L$ *iff* $\langle \underline{w}, + \rangle \models \varphi_L$.

The proof of [11]'s result on NLIN and $\mathbf{Mon}\Sigma_1^1(+)$ uses, as an intermediate step, a characterization of the class NLIN by a logic that existentially quantifies unary functions. There also exists an algebraic characterization of the class DLIN via unary functions [12]. Unfortunately, this characterization is not suitable for being used as an intermediate step in the proof of Theorem 4.2. What *can* be used for the proof of Theorem 4.2, however, is the following representation, basically taken from [11], of a *run* of a *d*-**bounded** DLIN-RAM $\mathcal{R}$. A run of $\mathcal{R}$ with input $f : [m] \to [m]$ is fully described by 6 functions $I, A, B, M, R_A, R'_A : [d{\cdot}m] \to [d{\cdot}m]$:

$I(t)$ = the number of the instruction performed in computation step $t{+}1$
$A(t)$ = content of the accumulator A directly *before* performing step $t{+}1$
$B(t)$ = content of the accumulator B directly *before* performing step $t{+}1$
$M(t)$ = content of the special register M directly *before* performing step $t{+}1$
$R_A(t)$ = content of register $R_{A(t)}$ directly *before* performing step $t{+}1$
$R'_A(t)$ = content of register $R_{A(t)}$ directly *after* performing step $t{+}1$.

It is not difficult to give inductive definitions of these functions

The flattening $\widetilde{G}$ of a function $G : [d{\cdot}m] \to [d{\cdot}m]$ is the concatenation of the $\{0,1\}$-strings $\widetilde{G}_0, \widetilde{G}_1, \ldots, \widetilde{G}_{dm-1}$, where $\widetilde{G}_i$ is the reverse binary representation of length $l := \lfloor \lg(d{\cdot}m) + 1 \rfloor$ of the number $G(i)$. I.e., $\widetilde{G}_i = b_0 b_1 \cdots b_{l-1}$ with $b_j \in \{0,1\}$ and $G(i) = \sum_{j<l} b_j \cdot 2^j$. It is straightforward to see that for every $d \in \mathbb{N}$ there is a $c \in \mathbb{N}$ such that the following is true for every $n \in \mathbb{N}$ and every function $G : [d{\cdot}m(n)] \to [d{\cdot}m(n)]$: The flattening $\widetilde{G}$ of G is a $\{0,1\}$-string of length $\leqslant c \cdot n$. Consequently, $\widetilde{G}$ can be represented by c subsets $\widetilde{G}^{(0)}, \ldots, \widetilde{G}^{(c-1)}$ of $[n]$ as follows: For every $p \in [n]$ and $\gamma \in [c]$, the $(\gamma{\cdot}n + p)$-th position of $\widetilde{G}$ carries the letter 1 if, and only if, $p \in \widetilde{G}^{(\gamma)}$.

We write $\widetilde{G}^{\bullet}$ for the *complement* of $\widetilde{G}$, i.e., the $\{0,1\}$-string obtained from $\widetilde{G}$ by replacing every 0 by 1 and every 1 by 0. Similarly, for $\gamma \in [c]$, $\widetilde{G}^{\bullet(\gamma)}$ denotes the complement of the set $\widetilde{G}^{(\gamma)}$.

Clearly, given a string w of length n and its functional representation $f_w : [m(n)] \to [m(n)]$, the flattenings (and their complements) of the functions $I, A, B, M, R_A, R'_A : [d{\cdot}m(n)] \to [d{\cdot}m(n)]$ that describe the computation of $\mathcal{R}$ on input f_w, can be represented by a fixed number of subsets of $[n]$. Using the inductive definitions of the functions I, A, B, M, R_A, R'_A mentioned above, we can show the following:

Lemma 4.3. *Let* $\mathbb{A} := \{1,2\}$, *let* $L \subseteq \mathbb{A}^+$, *let* $d \in \mathbb{N}$, *let* $\mathcal{R}$ *be a* ***d*-bounded** *DLIN-RAM that recognizes the set* $\{f_w : w \in L\}$, *and let* $c \in \mathbb{N}$ *be such that, for every* $n \geqslant 1$, *the flattening* $\widetilde{G}$ *of every function* $G : [d{\cdot}m(n)] \to [d{\cdot}m(n)]$ *is a* $\{0,1\}$-*string of length* $\leqslant c{\cdot}n$. *For every symbol* $S \in \mathbf{S} := \{I, A, B, M, R_A, R'_A, I^{\bullet}, A^{\bullet}, B^{\bullet}, M^{\bullet}, R_A{}^{\bullet}, R'_A{}^{\bullet}\}$ *and every* $\gamma \in [c]$ *let* $X_{S,\gamma}$ *be a set variable. Let* $\overline{X_{\mathbf{S},c}}$ *be the list of the set variables* $X_{S,\gamma}$ *for all* $S \in \mathbf{S}$ *and all* $\gamma \in [c]$.

For every $S \in \mathbf{S}$ *and every* $\gamma \in [c]$ *there is an* MLFP($\tau_{\mathbb{A}} \cup \{+\}$)-***formula*** $\varphi_{S,\gamma}(x, \overline{X_{\mathbf{S},c}})$ *such that the following is true for every string* $w \in \mathbb{A}^+$:

Let n be the length of w, let $f_w : [m(n)] \to [m(n)]$ be the functional representation of w, and let I, A, B,M, $R_A, R'_A : [d{\cdot}m(n)] \to [d{\cdot}m(n)]$ be the functions that describe the computation of $\mathcal{R}$ on input f_w. In the structure $\langle \underline{w}, + \rangle$, the simultaneous least fixed *point of all the formulas $\varphi_{S,\gamma}$ (for all $S \in \mathrm{S}$ and $\gamma \in [c]$) consists exactly of the sets $\big(\widetilde{S}^{(\gamma)}\big)_{S \in \mathrm{S}, \gamma \in [c]}$ that represent the flattenings, and their complements, of the functions I, A, B, M, R_A, R'_A.*

An important tool for the proof of Lemma 4.3 is the following

Lemma 4.4 (full arithmetic and counting in MLFP(+)).

(a) There are MLFP(+)*-formulas*

$$\varphi_<(x,y), \quad \varphi_\times(x,y,z), \quad \varphi_{Exp}(x,y,z), \quad \varphi_{Bit}(x,y), \quad \varphi_{Dy_1}(x,y), \quad \varphi_{Dy_2}(x,y),$$

such that for all $n \in \mathbb{N}$ and all $a, b, c \in [n]$, $\langle [n], + \rangle \models \varphi_<(a,b)$ (respectively, $\varphi_\times(a,b,c)$, $\varphi_{Exp}(a,b,c)$, $\varphi_{Bit}(a,b)$, $\varphi_{Dy_1}(a,b)$, $\varphi_{Dy_2}(a,b)$) if, and only if, $a < b$ (resp., $a \times b = c$, $a^b = c$, the b-th bit in the binary representation of a is 1, the b-th bit in the dyadic representation of a is 1, resp., 2).

(b) Let Y be a unary relation symbol. There is an MLFP(+)*-formula $\varphi_\#(x, Y)$ such that for all $n \in \mathbb{N}$, all $a \in [n]$, and all $B \subseteq [n]$ we have that*

$$\langle [n], + \rangle \models \varphi_\#(a, B) \iff a = |B| \,.$$

Using Lemma 4.3, Lemma 4.4, and the fact that MLFP has the same expressive power as S-MLFP (cf., Section 2), it is rather straightforward to find an **MLFP($\tau_{\mathbb{A}} \cup \{+\}$)-sentence** φ_L which, for every string $w \in \mathbb{A}^+$, is satisfied by $\langle \underline{w}, + \rangle$ if, and only if, $\mathcal{R}$ *accepts* input f_w. This, finally, will complete the proof of Theorem 4.2.

5 Addition-Invariant MLFP

In this section we concentrate on *addition invariant* formulas, i.e., on formulas that may use an addition relation on the underlying universe but that are independent of the particular choice of the addition relation.

The notion of "addition relation" is defined as follows: Let $\mathcal{U}$ be a finite set, let $n := |\mathcal{U}|$, and let $\oplus$ be a ternary relation on $\mathcal{U}$. $\oplus$ is called *an addition relation on $\mathcal{U}$* if there is a linear ordering $\leqslant$ of $\mathcal{U}$ such that $\mathcal{U} = \{u_0, .., u_{n-1}\}$ with $u_0 \leqslant \cdots \leqslant u_{n-1}$ and $\oplus = \big\{ (u_i, u_j, u_k) : i + j = k \text{ and } i, j, k \in \{0, .., n-1\} \big\}$.
We say that $\oplus$ is the particular addition relation that fits to the linear ordering $\leqslant$.

Definition 5.1 (addition-invariance). *Let $\mathcal{L}$ be a logic, let τ be a signature, and let $\oplus$ be a ternary relation symbol that does not occur in τ. An $\mathcal{L}(\tau \cup \{\oplus\})$-formula $\varphi(x_1, .., x_k)$ is called* addition-invariant *if the following is true for all finite τ-structures $\mathcal{A}$: For any two addition relations $\oplus_1$ and $\oplus_2$ on $\mathcal{U}^{\mathcal{A}}$ and all $a_1, .., a_k \in \mathcal{U}^{\mathcal{A}}$ we have $\langle \mathcal{A}, \oplus_1 \rangle \models \varphi(a_1, .., a_k) \iff \langle \mathcal{A}, \oplus_2 \rangle \models \varphi(a_1, .., a_k)$.*

Using Lemma 4.4, we can show

Lemma 5.2. *On linearly ordered structures, addition-invariant* MLFP *can define the particular addition relation that fits to the given linear ordering of the underlying structure.*

From Theorem 4.2 and Lemma 5.2 one directly obtains

Corollary 5.3 (DLIN $\subseteq$ addition-invariant MLFP on the class of finite strings). *For every finite alphabet $\mathbb{A}$ and every string-language $L \subseteq \mathbb{A}^+$ in* DLIN *there is an addition-invariant MLFP-sentence φ of signature $\tau_{\mathbb{A}} \cup \{\oplus\}$ such that, for every string $w \in \mathbb{A}^+$ and every addition relation $\oplus$ on $\underline{w}$'s universe, $w \in L$ iff $\langle \underline{w}, \oplus \rangle \models \varphi$.*

Using this and the well-known result that the satisfiability problem for *quantified Boolean formulas with k alternations of quantifiers* is complete for the k-th level of the polynomial time hierarchy, we can show that both, the equivalence of addition-invariant MLFP and MSO, as well as a separation of addition-invariant MLFP from MSO would solve open problems in complexity theory:

Theorem 5.4. *(a) If addition-invariant* MLFP $\neq$ *addition-invariant* MSO *on the class of finite strings, then* DLIN $\neq$ LINH.

(b) If addition-invariant MLFP = *addition-invariant* MSO *on the class of finite strings, then* PH = PTIME.

6 Conclusion

The main results of the present paper are (1) that MLFP can express graph properties beyond any fixed level of the monadic second-order quantifier alternation hierarchy, (2) that *addition-invariant* MLFP can express at least all string-problems that belong to the linear time complexity class DLIN, and (3) that settling the question whether addition-invariant MLFP has the same expressive power as addition-invariant MSO on finite strings would solve open problems in complexity theory.
Many interesting aspects of MLFP remain to be further investigated, for example:

- Is there a natural complexity class that is *exactly* captured by MLFP(+) on strings with built-in addition (analogous to the known result that MSO(+) exactly captures the linear time hierarchy LINH)? A promising candidate might be the time-space complexity class PTIME&LINSPACE of problems solvable by deterministic polynomial time, linear space bounded Turing machines.
- Is there a hierarchy within MLFP with respect to the alternation of *least* and *greatest* fixed point quantifiers? I.e., does MLFP have a hierarchy analogous to Bradfield's modal μ-calculus alternation hierarchy [2] ? Note that every level of this MLFP alternation hierarchy is closed under first-order quantification. Therefore, the alternation hierarchy of MLFP might be viewed as a "deterministic" analogue of the *closed monadic hierarchy* of [1] rather than as an analogue of the monadic second-order quantifier alternation hierarchy of [20].
- Investigate the parameterized complexity of the model checking problem for MLFP on various classes of finite structures. E.g., is the model checking problem for MLFP fixed parameter tractable on the class of planar graphs? Partial answers to this question have been obtained by Lindell [18].
- Does Theorem 3.6 still hold when replacing MLFP with the modal μ-calculus?

Acknowledgments. I want to thank Martin Grohe for valuable discussions on the subject of this paper.

References

1. M. Ajtai, R. Fagin, and L. Stockmeyer. The closure of Monadic NP. *Journal of Computer and System Sciences,* 60(3):660–716,2000. Journal version of *STOC'98* paper.
2. J. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theoretical Computer Science,* 195(2): 133–153,1998. Journal version of *CONCUR'96* paper.
3. A. Dawar. A restricted second order logic for finite structures. *Information and Computation,* 143:154–174,1998. Journal version of *LCC'94* paper.
4. A. Durand, C. Lautemann, and M. More. Counting results in weak formalisms. Technical Report 1998-14, Université de Caen, France, 1998.
5. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 2nd edition, 1999.
6. J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *Journal of the ACM,* 49(6):716–752,2002. Journal version of *ICDT'01* paper.
7. G. Gottlob and C. Koch. Monadic datalog and the expressive power of web information extraction languages. *Journal of the ACM,* 51(1):74–113,2004. Journal version of *PODS'02* paper.
8. E. Grandjean. Invariance properties of RAMs and linear time. *Computational Complexity,* 4:62–106,1994.
9. E. Grandjean. Linear time algorithms and NP-complete problems. *SIAM Journal on Computing,* 23(3):573–597,1994.
10. E. Grandjean. Sorting, linear time, and the satisfiability problem. *Annals of Mathematics and Artificial Intelligence,* 16:183–236,1996.
11. E. Grandjean and F. Olive. Monadic logical definability of nondeterministic linear time. *Computational Complexity,* 7(1):54–97,1998. Journal version of *CSL'94* paper.
12. E. Grandjean and T. Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. *SIAM Journal on Computing,* 32(1): 196–230,2002.
13. M. Grohe. *The structure of fixed-point logics.* PhD thesis, Albert-Ludwigs Universität Freiburg, Germany, 1994.
14. M. Grohe and N. Schweikardt. Comparing the succinctness of monadic query languages over finite trees. Technical Report EDI-INF-RR-0168, School of Informatics, University of Edinburgh, Scotland, U.K., 2003. Full version of *CSL'03* paper.
15. M. Grohe and T. Schwentick. Locality of order-invariant first-order formulas. *ACM Transactions on Computational Logic,* 1:112–130,2000. Journal version of *MFCS'98* paper.
16. N. Immerman. *Descriptive Complexity.* Springer, 1999.
17. R.M. Karp. Reducibility among combinatorial problems. In *IBM Symposium 1972, Complexity of Computers Computations.* Plenum Press, New York, 1972.
18. S. Lindell. Linear-time algorithms for monadic logic. Short presentation at the 18th IEEE Symposium on Logic in Computer Science (LICS'03), 2003.
19. J. F. Lynch. Complexity classes and theories of finite models. *Mathematical Systems Theory,* 15:127–144,1982.
20. O. Matz, N. Schweikardt, and W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs. *Information and Computation,* 179(2):356–383, 2002.
21. M. More and F. Olive. Rudimentary languages and second-order logic. Technical Report 1996-1, Université de Caen, France, 1996.
22. B. Rossman. Successor-invariance in the finite. In *Proceedings of the 18th IEEE Symposium on Logic in Computer Science (LICS'03),* 2003.

23. N. Schweikardt. On the expressive power of monadic least fixed point logic. Full version of *ICALP'04* paper. Available at `http://www.informatik.hu-berlin.de/~schweika/publications.html`.
24. T. Schwentick. Descriptive complexity, lower bounds and linear time. In *Proceedings of the 12th International Workshop on Computer Science Logic (CSL'98),* volume 1584 of *Lecture Notes in Computer Science,* pages 9–28. Springer, 1998. Invited paper.
25. W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages,* volume 3. Springer, 1996.

Counting in Trees for Free

Helmut Seidl[1], Thomas Schwentick[2]*, Anca Muscholl[3], and Peter Habermehl[3]

[1] TU München, I2,
seidl@in.tum.de
[2] Philipps-Universität Marburg, FB Mathematik und Informatik
tick@informatik.uni-marburg.de
[3] LIAFA, Université Paris 7, 2, pl. Jussieu, F-75251 Paris

Abstract. It is known that MSO logic for ordered unranked trees is undecidable if Presburger constraints are allowed at children of nodes. We show here that a decidable logic is obtained if we use a modal flxpoint logic instead. We present a characterization of this logic by means of *deterministic* Presburger tree automata and show how it can be used to express numerical document queries. Surprisingly, the complexity of satisfiability for the extended logic is asymptotically the same as for the original fixpoint logic. The non-emptiness for Presburger tree automata (PTA) is PSPACE-complete, which is moderate given that it is already PSPACE-hard to test whether the complement of a regular expression is non-empty. We also identify a subclass of PTAs with a tractable non-emptiness problem. Further, to decide whether a tree t satisfies a formula φ is polynomial in the size of φ and linear in the size of t.
A technical construction of independent interest is a *linear* time construction of a Presburger formula for the Parikh image of a regular language.

1 Introduction

In XML schema description languages as DTDs and XML Schema, the content of elements, i.e., the possible sequences of children elements of a node, is described mainly by regular expressions.[1] This is sufficient in very many cases. But often one is interested in expressing conditions on the frequency of occurrences of elements in the children sequence. When the order of elements is very constrained regular expressions still do the job, e.g. by $\texttt{(title author author}^+\texttt{)}$ one might express that there have to be at least two authors in a paper. If the order is not fixed, even simple conditions require complicated regular expressions. E.g., saying that there is exactly one title and there are at least two authors would require an expression like $\texttt{(title author author}^+\texttt{)} \mid \texttt{(author}^+ \texttt{ title author}^+\texttt{)} \mid \texttt{(author author}^+ \texttt{ title)}$. It would be desirable to describe this condition simply by an expression like $|\texttt{title}| = 1 \wedge |\texttt{author}| \geq 2$. While these conditions do not go beyond the scope of regular expressions, others do. A simple example is $|\texttt{author}| \leq 2 \cdot |\texttt{title}|$.

* Contact author. The support of this work by the DAAD and Egide under PROCOPE grant D/0205766 is kindly acknowledged.

[1] We view an XML document here and in the rest of the paper as a labeled, unranked, ordered tree.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1136–1149, 2004.

Most of the existing theoretical work on XML schema languages has concentrated on regular tree languages. These languages can be described by tree automata [14,15] and a variety of other formalisms [16,8] including fixpoint formulas [13]. Typically, the interaction between the children of a node and the node itself are usually expressed in terms of regular expressions. Other work extended these formalisms to let them formulate (at least unary) queries. The resulting query facilities usually have the expressive power of monadic second-order (MSO) logic. Here, we study extensions of such formalisms by numerical conditions as above. In particular, we are interested in the main complexity questions.

The conditions we allow are Boolean combinations of regular expressions and Presburger formulas. Presburger formulas basically allow linear (in)equalities and expressions of the form $t \equiv c \pmod{n}$. A more detailed definition can be found in Section 2. Counting conditions in schema languages have been used, e.g., in [12].

In a previous paper [22] we considered *non-deterministic* tree automata with such extended conditions (PTAs). This kind of automata is not closed under complementation. Moreover, whereas their non-emptiness problem (whether an automaton accepts some tree) is decidable, the universality problem (whether it accepts all trees) is not. Consequently, MSO logic extended by such conditions has an undecidable satisfiability problem. In the present paper, we study two weaker formalisms. We consider a *fixpoint* logic instead of MSO logic and *deterministic* instead of non-deterministic PTAs. It turns out that these two formalisms define the same class of tree languages. Furthermore, their non-emptiness (resp., satisfiability) problem is decidable. Actually, it came as a surprise that the complexities of these problems are as low as one could hope for[2]:

- It is already PSPACE-hard to check whether the intersection of several regular expressions is empty. Therefore, the non-emptiness problem for PTAs is trivially PSPACE-hard. We prove that it is also in PSPACE. Additionally, we show that it becomes tractable when each precondition of the automaton is a disjunction of formuls $r \wedge f$, where r is a regular expression and f is an equation (with existentially quantified variables).
- Satisfiability for fixpoint formulas (without numerical conditions) is EXPTIME-complete. We show that the complexity does not increase when we add numerical conditions.
- The same complexities can be easily derived for the containment problem.
- Checking whether a tree t is accepted by a Presburger tree automaton A or a fixpoint formula ϕ can be decided in time $O(|t||A|)$ and $O(|t||\phi|^2)$, respectively.

Furthermore, we show how Presburger fixpoint formulas can be used to formulate unary queries. These queries can be evaluated in time which is linear in the size of the tree and polynomial in the size the formula. During our investigation we also studied the relationship between regular expressions and Presburger formulas. It is well-known that the Parikh image of each regular language (i.e., basically the set of symbol frequency vectors of words) can be expressed by a Presburger formula. We show that such a formula can be constructed very efficiently, in linear time, even from a non-deterministic finite automaton.

[2] Actually these complexities hold only with quantifier-free Presburger formulas. However, this does not restrict the expressivity of the logic.

The paper is organized as follows. In Section 2 we give the basic definitions for Presburger logic. Section 3 explains how to compute efficiently a Presburger formula from a regular string language. Section 4 introduces Presburger fixpoint formulas. In Section 5 we define Presburger automata and prove the equivalence with Presburger fixpoint formulas. Section 6 contains the complexity results. In Section 7 we define a query extension of Presburger fixpoint formulas and consider its evaluation complexity. We end with a conclusion.

Related work. Unordered document trees are closely related to the generalization of feature trees considered by Niehren and Podelski in [17] where they study the (classical) notion of recognizability and give a characterization of this notion by means of feature automata. No counting constraints are considered. Query languages for unordered trees have been proposed by Cardelli and Ghelli [2,1,3,4]. Their approach is based on first-order logic and fixpoint operators. An extension to numerical constraints has recently been proposed by Dal Zilio et al. [5]. Kupferman et al. study a μ-**calculus** with *graded* modalities where one can express, e.g., that a node has at least n successors satisfying a given property [10]. The numbers n, however, are hard-coded into the formula. Ordered successors are not considered. Klaedtke and Ruess consider automata on the unlabeled infinite binary tree, that have an accepting condition depending on a global Presburger constraint [9].

Our notion of Presburger Tree Automata, which combines both regular constraints on the children of nodes as well as numerical constraints given by Presburger formulas, has independently been introduced by Lugiez and Dal Zilio [11] and Seidl et al. [22]. In their paper, Lugiez and Dal Zilio indeed propose a modal logic for XML documents which they call *Sheaves logic*. This logic allows to reason about numerical properties of the contents of elements but still lacks recursion, i.e., fixpoint operators. Lugiez and Dal Zilio consider the satisfiability and the membership problem and they show that Sheaves logic formulas can be translated into deterministic PTAs. Seidl et al. in [22] on the other hand, prove that nondeterministic PTAs precisely correspond to the existential fragment of MSO logic on ordered trees enhanced with Presburger constraints on the children of nodes. As a technical result, they also show that *first-order* formulas can be translated into deterministic PTAs.

2 Preliminaries

Presburger Logic is the first-order logic over the structure $(\mathbb{N}, \leq, +)$. Given a formula f and an *assignment* σ mapping the variables of f to numbers, we write $\sigma \models f$ if f holds for σ (in the obvious sense) and call σ a solution of f. For convenience, we use an extended language. Thus, we write cx for $x + \cdots + x$ (c **times**). Also, we allow terms with negative coefficients as in $2y - 3x$. A typical Presburger formula is $\exists y\ (2y = x)$ stating that x is even. It is well-known that the extension of Presburger logic by 0, 1 and the binary predicates $x \equiv y\ (\mathbf{mod}\ n)$, for each constant n, has quantifier elimination, i.e., for each formula there is an equivalent quantifier-free formula [20]. E.g., the above formula can be written as $x \equiv 0\ (\mathbf{mod}\ 2)$. Here, we call quantifier-free formulas in the extended language with modulo predicates and equality over terms with integer coefficients *quantifier-free Presburger formulas.* We say that formulas of the form $\exists x_1, \ldots, x_k \bigvee_{i=1}^{m} f_i$, where

each f_i is a conjunction of equations $t = c$ with a term t and an integer constant c are in *equation normal form.* Note that formulas in equation normal form do not contain any negations.

Lemma 1. *Every Presburger formula has an equivalent formula in equation normal form.*

It is well-known that sets of assignments which fulfill a given Presburger formula f are equivalent to *semi-linear sets* [7]. A semi-linear set is a finite union of *linear sets* of the form $\{\sigma + \sum_{i=1}^{m} \sigma_i z_i \mid z_i \in \mathbb{N}\}$, where σ and the σ_i are assignments to a finite set of variables (using a fixed enumeration of the variables) or vectors from $\mathbb{N}^k$ for a given k.

The *Parikh image* of a word $w = a_1 \cdots a_k, a_j \in \Sigma$ is the assignment $\sigma \in \mathbb{N}^{\Sigma}$ which maps the variables $|a|, a \in \Sigma$, to the number of occurrences of the letter a in w, i.e., $\sigma(|a|) = \#\{j \mid a = a_j\}$. Accordingly, the Parikh image of a set $L \subseteq \Sigma^*$ is the set of Parikh images of $w \in L$.

3 Regular String Languages and Presburger Formulas

The fixpoint formulas as well as the tree automata studied here use Boolean combinations of regular expressions and Presburger formulas as conditions on the children of nodes. Whereas it is well-known that the Parikh image of a regular (even context-free) language is semilinear [19] and thus can be described by a Presburger formula with free variables $|a|, a \in \Sigma$, it seems to be not quite as well-known how large the corresponding formula must be. In this section, we show that a Presburger formula for the Parikh image of the language of an NFA A can be computed in linear time. In particular, the size of the formula is *linear* in the size $|A|$ of A (which equals the number of states plus the number of transitions). For regular expressions we have another, direct linear-time construction.

Theorem 1. *For any NFA A, an existential Presburger formula φ_A for the Parikh image of the language $\mathcal{L}(A)$ of A can be constructed in time $\mathcal{O}(|A|)$.*

Sketch of proof. With an accepting run of an NFA A on a string w we associate a *flow* f as follows: each transition (p, a, q) of A is labeled by the number of times it is taken in the computation. We construct a Presburger formula which checks two properties. First, the flow is locally consistent, e.g., for each inner node the incoming equals the outgoing flow. Secondly, the subgraph induced by the states with non-zero flow is connected. Here for each node, the distance is guessed from s w.r.t. non-zero flow edges. □

4 Presburger Fixpoint Formulas

In many applications, e.g., where documents are automatically generated from databases as textual representations of querying results, the element ordering on the children does not matter (or it is not known in advance). In other applications, though, which are more related to classical document processing the ordering matters. Since we cannot tell just from looking at a linearized textual representation of the document whether the ordering of children is irrelevant, we prefer to work with ordered trees only but allow the logic

to express properties of unordered documents. Thus, given an alphabet Σ of element or node names, the set of all (ordered but unranked) trees t is given by:

$$t \ ::= \ a\langle t_1, \ldots, t_k \rangle \ , \quad a \in \Sigma, k \geq 0$$

We write $\mathcal{T}_\Sigma$ for the set of all such trees. We consider a calculus of fixpoint formulas which allows to express both regular and Presburger constraints on children of nodes. Presburger fixpoint formulas ϕ are constructed according to the following grammar:

$$\begin{array}{l} \phi ::= \top \ \mid \ x \ \mid \ \mu x.\phi \ \mid \ \phi_1 \vee \phi_2 \ \mid \ \phi_1 \wedge \phi_2 \ \mid \ a\langle F \rangle \ \mid \ *\langle F \rangle \\ F ::= r \ \mid \ \neg r \ \mid \ f \end{array}$$

Here, "$*$" denotes an arbitrary node label from Σ, and F denotes a generic pre-condition on the children of a node. Such a pre-condition is either a regular expression r over letters ϕ, ϕ a fixpoint formula, or a Presburger formula f with free variables $|\phi|$ denoting the number of children satisfying ϕ. Essentially the same calculus is obtained if we enhance the *Sheaves logic* of Dal Zilio and Lugiez [11] with recursion.

In the sequel, we assume that ϕ is a formula where all bound variables are distinct. Let Φ denote the set of all subformulas of ϕ plus $\top$ (the constant true).[3] We consider assertions $t : \psi, t \in \mathcal{T}_\Sigma, \psi \in \Phi$. We write $\vdash t : \psi$ either if $\psi \equiv \top$ (every tree satisfies $\top$) or if the assertion $t : \psi$ can be derived from valid assertions by means of the following rules:

$$\frac{t : \psi \qquad \mu x.\psi \in \Phi}{t : x} \qquad \frac{t : \psi \qquad \mu x.\psi \in \Phi}{t : \mu x.\psi}$$

$$\frac{t : \psi_1 \qquad t : \psi_2}{t : \psi_1 \wedge \psi_2} \qquad \frac{t : \psi_i}{t : \psi_1 \vee \psi_2}$$

$$\frac{u : F}{a\langle u \rangle : a\langle F \rangle} \qquad \frac{u : F}{a\langle u \rangle : *\langle F \rangle}$$

Thus, besides assertions $t : \psi, t \in \mathcal{T}_\Sigma$, we additionally need auxiliary assertions $u : F$ where u is a sequence of trees and F is either a regular expression or a Presburger formula. A sequence $u = t_1 \ldots t_k$ satisfies a regular pre-condition r (or $\neg r$) iff there are formulas $\psi_1, \ldots, \psi_k$ such that $t_i : \psi_i$ and the sequence $\psi_1 \ldots \psi_k$ is (not) contained in the language $\mathcal{L}(r)$ of r. In case of a Presburger formula f, we collect for every formula ψ the number of children t_i satisfying ψ into an assignment σ. Then u satisfies f iff $\sigma \models f$. Thus we have:

$$\frac{t_i : \psi_i \quad (i = 1, \ldots, k) \qquad \psi_1 \ldots \psi_k \in \mathcal{L}(r)}{t_1 \ldots t_k : r} \qquad \frac{t_i : \psi_i \quad (i = 1, \ldots, k) \qquad \psi_1 \ldots \psi_k \notin \mathcal{L}(r)}{t_1 \ldots t_k : \neg r}$$

$$\frac{\sigma \models f \quad \text{where} \quad \sigma(|\psi|) = \#\{i \mid t_i : \psi\}}{t_1 \ldots t_k : f}$$

Note that according to this rule for Presburger formulas, the same tree t_i may be counted several times, once for every ψ such that $t_i : \psi$. A *proof* of an assertion $t : \psi$ consists of all rule applications to derive this assertion. In particular this means for $t = a\langle t_1 \ldots t_k \rangle$

[3] Φ also contains the subformulas of ψ if $|\psi|$ occurs in ϕ and so on.

and $\psi = a\langle f \rangle$, f a Presburger formula, that a proof of $t : \psi$ contains for every $i = 1, \ldots, k$, and every ψ' a subproof of $\vdash t_i : \psi'$ – whenever it exists. Moreover, we assume that a proof always has tree-like structure. Thus, we may have several copies of a subproof for distinct occurrences of the same subtree within t. Finally, the language denoted by the formula ϕ is given by: $\mathcal{L}(\phi) = \{t \in \mathcal{T}_\Sigma \mid \vdash t : \phi\}$. In particular, $\mathcal{L}(\top) = \mathcal{T}_\Sigma$ and $\mathcal{L}(\mu x.\, x) = \emptyset$. Using the convenient abbreviation "_" for $\top^*$, we may write $\mu x.\, (a\langle \_ \rangle \vee *\langle \_\ x\ \_ \rangle)$ for the set of all trees with at least one inner node labeled a. Note that our fixpoint expressions do not provide an explicit notion of negation. However, we always can construct an equivalent expression with *guarded* fixpoints for which complementation is easy [23].

5 Presburger Automata

We recall the notion of a Presburger tree automaton (PTA) for ordered trees from [22, 11]. A *Presburger tree automaton* A is a tuple (Q, Σ, δ, T) where, as usual, Q, Σ, δ and $T \subseteq Q$ are the finite set of states, the input alphabet, the transition relation and the set of accepting states of A, respectively. Here the transition relation δ is given by a mapping from $Q \times \Sigma$ to a pre-condition on the children of a node with label a to reach q in a bottom-up run over an input tree. For PTAs, pre-conditions are Boolean combinations of regular expressions r over the state set Q and Presburger formulas f with free variables $|q|, q \in Q$. We define satisfaction relations $u \models p$ for $u \in Q^*$ and pre-conditions p and $t \models_A q$ for $t \in \mathcal{T}_\Sigma, q \in Q$:

$$
\begin{array}{rl}
q_1 \ldots q_k \models r & \text{iff } q_1 \ldots q_k \in \mathcal{L}(r) \\
q_1 \ldots q_k \models f & \text{iff } \sigma \models f \text{ where } \sigma(|q|) = \#\{i \mid q_i = q\} \\
q_1 \ldots q_k \models p_1 \vee p_2 & \text{iff } q_1 \ldots q_k \models p_1 \text{ or } q_1 \ldots q_k \models p_2 \\
q_1 \ldots q_k \models p_1 \wedge p_2 & \text{iff } q_1 \ldots q_k \models p_1 \text{ and } q_1 \ldots q_k \models p_2 \\
q_1 \ldots q_k \models \neg p & \text{iff } q_1 \ldots q_k \not\models p \\
a\langle t_1 \ldots t_k \rangle \models_A q & \text{iff } t_i \models_A q_i \text{ for all } i \text{ and } q_1 \ldots q_k \models \delta(q, a),
\end{array}
$$

Note here that satisfaction of a Presburger pre-condition f takes a different flavor than the corresponding definition for fixpoint formulas: In an automaton each subtree of a node takes only one state and thus contributes exactly once to the value of some $\sigma(|q|)$. Opposed to this, the variables $|\psi|$ in fixpoint formulas count every subtree on which ψ holds, hence a subtree might contribute to the value of several (or no) variables.

The automaton A is called *deterministic* iff for all $a \in \Sigma$ and all $u \in Q^*$, $u \models \delta(q, a)$ for exactly one $q \in Q$. In the proof that deterministic PTA and Presburger fixpoint formulas are equivalent we use the following notion. For a subset $B \subseteq \Phi$ of subformulas of ϕ, define the *closure* $\mathsf{cl}(B)$ as the least superset B' of B such that:

- $\top \in B'$;
- If $\phi_1 \in B'$ and $\phi_2 \in B'$ then also $\phi_1 \wedge \phi_2 \in B'$, whenever $\phi_1 \wedge \phi_2 \in \Phi$;
- If $\phi_1 \in B'$ or $\phi_2 \in B'$ then also $\phi_1 \vee \phi_2 \in B'$, whenever $\phi_1 \vee \phi_2 \in \Phi$;
- If $\phi' \in B'$ then $\mu x.\phi' \in B'$ and $x \in B'$, whenever $\mu x.\phi' \in \Phi$.

Intuitively, the closure of a set B of subformulas contains all subformulas which are implied by the formulas in B and reachable by a (virtual) bottom-up traversal over an input tree constructing a proof for the fixpoint formula ϕ.

Theorem 2. *For a tree language $L \subseteq \mathcal{T}_\Sigma$ the following statements are equivalent:*

(1) $L = \mathcal{L}(\phi)$ for some fixpoint formula ϕ;
(2) $L = \mathcal{L}(A)$ for some deterministic PTA A.

Proof. (1) $\Rightarrow$ (2): Let ϕ be a Presburger fixpoint formula. We construct a PTA A as follows. Let Ψ denote the set of all subformulas of ϕ of the form $a\langle F\rangle$ or $*\langle F\rangle$. The set Q of states of A is given as the set of all subsets $B \subseteq \Psi$. The set T of accepting states consists of all subsets B such that $\phi \in \mathsf{cl}(B)$, i.e., whose closure contains the whole formula ϕ. Given a state $B \in Q$ and $a \in \Sigma$, we determine the pre-condition $\delta(B, a)$ as

$$\delta(B,a) = \bigwedge_{\psi\in B} \delta_0(\psi,a) \ \wedge \ \bigwedge_{\psi\in\Psi\setminus B} \neg\delta_0(\psi,a)$$

where:

$$\begin{aligned}
&\delta_0(a\langle F\rangle, a) = \bar{F}\\
&\delta_0(*\langle F\rangle, a) = \bar{F}\\
&\delta_0(b\langle F\rangle, a) = \mathsf{false} \qquad \text{if } a \neq b
\end{aligned}$$

and $\bar{F}$ is constructed as follows. For a possibly negated regular expression r, we obtain $\bar{r}$ from r by substituting $(B_1 \mid \ldots \mid B_m)$ for every occurrence of a formula ψ if $\{B_1, \ldots, B_m\}$ is the set of all states B such that $\psi \in \mathsf{cl}(B)$. For a Presburger formula f, let $\bar{f}$ be obtained from f by substituting $\sum_{\psi\in\mathsf{cl}(B_i)} |B_i|$ for every occurrence of the free variable $|\psi|$. By construction, the resulting automaton is deterministic. We claim:

1. For every $\psi \in \Phi$, $\vdash t : \psi$ iff $t \models_A B$ for some $B \in Q$ with $\psi \in \mathsf{cl}(B)$;
2. $\vdash t_1 \ldots t_k : r$ iff $t_i \models_A B_i$ for some states B_i such that $B_1 \ldots B_k \in \mathcal{L}(\bar{r})$;
3. $\vdash t_1 \ldots t_k : f$ iff $t_i \models_A B_i$ for some states B_i such that $\sigma \models \bar{f}$ where σ is the Parikh image of $B_1 \ldots B_k$.

In particular, the first item of the claim implies that $\mathcal{L}(\phi) = \mathcal{L}(A)$.
(2) $\Rightarrow$ (1): For the reverse implication, consider a deterministic PTA $A = (Q, \Sigma, \delta, F)$. W.l.o.g. we may assume that no negation occurs in preconditions. We introduce one variable x_q for every state $q \in Q$. For these variables, we construct an equation system S_A:

$$x_q \ = \ \psi_q \ , \quad q \in Q$$

where the right-hand sides are fixpoint formulas. The semantics of such equation systems is an extension of the semantics for fixpoint formulas. The only addition is a rule:

$$\frac{t : \psi}{t : x}$$

for every equation $x = \psi$. Thus, whenever a tree satisfies the right-hand side of an equation, then it also satisfies the variable to the left. The right-hand sides ϕ_q of the equation system S_A are constructed from the right-hand sides $\delta(q, a), a \in \Sigma$, as follows:

$$\phi_q \ = \ \bigvee_{a\in\Sigma} [\delta(q,a)]_a$$

where $[.]_a$ takes a pre-condition and returns a fixpoint formula (without fixpoints):

$$\begin{aligned}
[r]_a &= a\langle r\{q \mapsto x_q \mid q \in Q\}\rangle \\
[f]_a &= a\langle f\{|q| \mapsto |x_q| \mid q \in Q\}\rangle \\
[p_1 \vee p_2]_a &= [p_1]_a \vee [p_2]_a \\
[p_1 \wedge p_2]_a &= [p_1]_a \wedge [p_2]_a
\end{aligned}$$

Thus, a regular expression r over states q is transformed by first substituting the states by the corresponding variables and then putting a node a on top. A Presburger formula is transformed by first replacing the free $|q|$ with $|x_q|$, $q \in Q$, and again putting a node a on top, whereas conjunctions and disjunctions are transformed by recursively proceeding to the involved conjuncts and disjuncts, respectively. By induction on the depth of terms $t, t_1, \ldots, t_m$ and pre-conditions p, we prove for every $q \in Q$ and $a \in \Sigma$:

(1) $t \models_A q$ iff $t : x_q$;
(2) $t_i \models_A q_i$ for $i = 1, \ldots, m$, with $q_1 \ldots q_m \models p$ iff $a\langle t_1 \ldots t_m\rangle : [p]_a$

The first claim then proves the correctness of the construction. The only non-trivial point in the proof of the claim is the inductive step for assertion (2). □

6 Complexity

In this section we study the complexity of decision problems related to Presburger automata and Presburger fixpoint formulas. The complexity of testing satisfiability of arbitrary Presburger formulas is prohibitively high, since the problem is hard for non-deterministic double exponential time [6]. As we are interested to study the interplay between regular expressions, Presburger formulas and tree automata, we assume for our complexity considerations that all Presburger formulas are given as *quantifier-free formulas* (possibly with modulo predicates and subtraction). Coefficients, like c in cx, are in binary notation, except in theorem 4, which is a special case that can be dealt with efficiently. Recall also from Lemma 1 that any quantifier-free Presburger formula can be transformed into equation normal form. The DNF transformation might yield an exponential number of disjuncts. However, each disjunct can only contain a linear number of atoms. Further, the normalization does not increase the size of the occurring numbers.

First, we consider the non-emptiness problem for Presburger automata, i.e., given a PTA A it has to be checked whether A accepts at least one tree. It is already PSPACE-hard to decide whether a given set of regular expressions has a non-empty intersection or whether the complement of a single regular expression is non-empty[24]. Hence, the non-emptiness problem for PTA is PSPACE-hard. Surprisingly, it can also be solved in PSPACE. For that, the following observation about the representation of Parikh images of finite word automata turns out to be useful. It follows with a pumping argument, by observing that every path in the automaton can be decomposed into a union of simple cycles and one simple path.

Lemma 2. *Assume A is a (non-deterministic) finite word automaton with n states and input alphabet of size k. Then the Parikh image of $\mathcal{L}(A)$ is a union of linear sets $\{\sigma_0 + \sum_{i=1}^{m} x_i \cdot \sigma_i \mid x_i \geq 0\}$ where each component of each vector $\sigma_j \in \mathbb{N}^k$ is at most n.*

In particular, the number m of occurring vectors is at most n^k.

Theorem 3. *The non-emptiness problem for (non-deterministic) PTAs is* PSPACE-*complete.*

Proof. It remains to prove the upper bound. For that, let $A = (Q, \Sigma, \delta, T)$ be a PTA of size n. We call a state q of A *reachable,* if there is a tree t such that $t \models_A q$. We have to check whether there is a reachable state in T. The set R of reachable states can be computed in a standard fashion as follows. First, the set R consists of all states q such that for some single-node tree t, we have $t \models q$. Then, given a set R of reachable states, we obtain a (possibly larger) set R' of reachable states q by checking whether there is a string w over R and a symbol a, such that $w \models \delta(q, a)$. This process stops after at most $|Q|$ iterations. Hence, to get the desired upper bound, it suffices to show the following claim.

Claim. Given a PTA $A = (Q, \Sigma, \delta, T)$, $R \subseteq Q$, $q \in Q$, $a \in \Sigma$, it can be checked in space polynomial in $|A|$, whether there is a string $w \in R^*$ such that $w \models \delta(q, a)$.
The proof of the claim proceeds in two steps. First, we show that the length of the shortest word satisfying $\delta(q, a)$ has length at most $2^{p(n)}$, p a suitably defined polynomial not depending on A. Second, we show that checking whether there exists some $w \in R^*$ of length at most $2^{p(n)}$ with $w \models \delta(q, a)$ can be checked in polynomial space.

The precondition $\delta(q, a)$ can be written in disjunctive normal form. Each disjunct is a conjunction of regular expressions $r_1, \ldots, r_k$, negated regular expressions $\neg r'_1, \ldots, \neg r'_l$, and $m \leq n$ Presburger equations of the form $t_i = c_i$ over variables $|q|, q \in Q$, and possibly other, free variables (at most $5n$). The formula $\delta(q, a)$ has a model if and only if one of these disjuncts has a model. Since the regular expressions all occur in A, the sum of their sizes is less than n. Let $A_1, \ldots, A_k$ and $A'_1, \ldots, A'_l$ be the corresponding non-deterministic automata. Then the natural deterministic automaton A' for their product has at most 2^n states. By Lemma 2, the Parikh image of $\mathcal{L}(A')$ is a union of linear sets $\{\sigma_0 + \sum_{i=1}^{h} x_i \sigma_i \mid x_i \in \mathbb{N}\}$, where $h < 2^{n \cdot |Q|} \leq 2^{n^2}$ and the entries of the vectors σ_0, σ_i are smaller than 2^n. Hence, a word fulfills the disjunct iff its Parikh image τ fulfills the Presburger equations and is in one of these linear sets. The latter can be expressed by:

$$|q| = \sigma_0(q) + \textstyle\sum_{i=1}^{h} x_i \cdot \sigma_i(q) \quad , \; q \in Q \; .$$

Together we have $M = m + |Q| \leq 2n$ equations with at most $N = 6n + 2^{n^2}$ variables and coefficients of values bounded by $a = 2^n$. By a result of Papadimitriou [18] such a system has a solution with numbers bounded by

$$N \cdot (M \cdot a + 1)^{2M+4} = (6n + 2^{n^2}) \cdot (2n2^n + 1)^{4n+4} = 2^{O(n^2)}$$

This proves the first step, for some polynomial $p(n) = O(n^2)$.

It remains to describe the algorithm which checks whether a string w of size $2^{p(n)}$ over Q exists such that $w \models \delta(q, a)$. The algorithm is non-deterministic. It simply guesses w symbol by symbol. For each regular expression r in $\delta(q, a)$, it computes the set of states that can be reached by the corresponding automaton A_r when reading w. Further, for each $q' \in Q$ it counts how often q' occurs in w. All this can be done in polynomial space without actually storing w. A counter keeps track of the length of w. In the end, it can be checked whether $w \models \delta(q, a)$. By Savitch's theorem this non-

deterministic polynomial space algorithm can be turned into a deterministic one still using polynomial space. □

Since our PTAs are deterministic and thus complementable by exchanging the sets of accepting and non-accepting states, we obtain as an immediate consequence:

Corollary 1. *The containment problem for deterministic PTAs is* PSPACE*-complete.* □

For certain PTAs non-emptiness can be tested in polynomial time — actually with the same complexity as for tree automata with single regular expressions as preconditions.

Theorem 4. *Non-emptiness can be decided in polynomial time for PTAs where every precondition is of the form* $\bigvee_{i=1}^{k}(r_i \wedge f_i)$*, with regular expressions* r_i *and Presburger formulas* f_i *in equation normal form with only one equation and coefficients represented in unary.*

Allowing coefficients in binary notation makes this problem less tractable.

Theorem 5. *The non-emptiness problem for PTAs as in Theorem 4 but with coefficients represented in binary is* NP*-complete.*

Now we turn to the related problem of deciding satisfiability for Presburger fixpoint formulas. Here, an EXPTIME lower bound is given by the same problem for fixpoint formulas without Presburger subformulas. The lower bound is achieved already by formulas with only one occurrence of μ (a similar result holds for model-checking μ-calculus against pushdown graphs, [25]). Testing whether such formulas are satisfiable by some tree is EXPTIME-complete. Again, it turns out that adding Presburger formulas does not increase the complexity, i.e., we get the following result.

Theorem 6. *Satisfiability for Presburger fixpoint formulas is* EXPTIME*-complete.*

The proof follows a similar line as the one for Theorem 3. Next we show that membership for deterministic PTA as well as for fixpoint formulas can be solved efficiently. This means that properties expressed by deterministic PTA are indeed of practical use:

Theorem 7. *Given a tree* t *and a deterministic PTA* A*, it can be checked in time* $\mathcal{O}(|t| \cdot |A|)$ *whether* $t \in \mathcal{L}(A)$.

Proof. Since the PTA is deterministic, it suffices to compute bottom-up the state reached by each node of t. Since every Presburger formula and thus, every precondition on a node with k children can be evaluated in time $\mathcal{O}(k \cdot |A|)$, the claim follows. □

Theorem 8. *Given a tree* t *and a fixpoint formula* ϕ*, it can be checked in time* $\mathcal{O}(|t| \cdot |\phi|^2)$ *whether* $t \models \phi$.

Proof. We compute bottom-up the set of formulas satisfied by each subtree. For each node we have to simulate the NFA corresponding to regular expressions r, by keeping the set of reachable states of the NFA. Since each NFA is of size at most $|\phi|$, each such simulation costs at most $\mathcal{O}(|\phi|^2)$. For Presburger constraints we just need to count how many children satisfy a given subformula, which can be done in $\mathcal{O}(|\phi|)$. □

7 The Query Language

Fixpoint formulas allow to express properties of (document) trees. We now construct an expressive querying language which still allows for efficient algorithms to collect all matches in a tree. In the example shown in Figure 7 we might ask for all items containing "Bartoli". A second query could ask for items containing "Bartoli" and having at least three reviews. with at least three reviews, Presburger fixpoint formulas easily can express that a tree contains a node (at unkown depth) satisfying a given property. E.g., the formula $\phi_1 = *\langle \_ \texttt{"Bartoli"} \_ \rangle$ describes all elements containing "Bartoli". Note that text contents can be taken into account by (conceptually) considering each text character as a separate element node. We are interested in the class of all documents containing sub-documents satisfying the specific property ϕ_1. These are described by: $\mu\, x.(*\langle \_\ x\ \_ \rangle \vee \phi_1)$.

```
<music> ...
   <classical> ...
      <opera>
         <title> The Salieri Album </title>
         <composer> Bartoli </composer>
         <review> ... </review>
         <review> ... </review>
         <review> ... </review></opera>
      <opera>
         <title> The No. 1 Opera Album </title>
         <composer> Puccini ; Verdi </composer>
         <performer> Bartoli ; Pavarotti </performer>
         <review> ... </review></opera>
   </classical> ...
</music>
<dvd> ...
   <music dvd>
      <opera>
         <title> Rossini - La Cenerentola </title>
         <performer> Bartoli </performer>
         <review> ... </review>
         <review> ... </review></opera> ...
   </music dvd>
</dvd>
```

Fig. 1. Part of an example document containing information about items sold by a store.

In order to indicate the sub-formula corresponding to requested sub-documents, we introduce an extra marker "•". Thus, we specify the query as $\psi_1 = \mu x.(*\langle \_\ x\ \_ \rangle \vee (\bullet \wedge \phi_1))$. Accordingly for the second query, we describe the set of all elements containing at least three reviews by: $\phi_2 = *\langle |\texttt{review}| \geq 3 \rangle$. The query formula then can be formulated as:

$$\psi_2 = \mu x.(*\langle \_\ x\ \_ \rangle \vee (\bullet \wedge \phi_1 \wedge \phi_2))$$

Thus, a query language is obtained by extending Presburger fixpoint formulas by one case:

$$\phi \quad ::= \quad \ldots \quad | \quad \bullet \quad | \quad \ldots$$

Accordingly, we add new axioms $\vdash t : \bullet$ for all trees t. A *match* s of a formula ϕ containing a subformula $\bullet$ is a proof for $t : \phi$ containing the fact $s : \bullet$. We want to construct an algorithm to determine for a fixed query formula ϕ, all matches inside a document tree t. We first observe that we can determine in linear time for every subtree s of t the set of all subformulas ψ' of ϕ such that $\vdash s : \psi'$. For that, we could construct, e.g., the deterministic PTA A for ϕ as considered in the last section. In order to deal with the special symbol $\bullet$ occurring in ϕ, we extend the notion of closure of states by adding the formula $\bullet$. The rest of the construction we leave unchanged. Let then $S(s)$ denote the unique state of A with $s \models_A S(s)$. By construction, $\psi' \in \mathsf{cl}(S(s))$ iff $\vdash s : \psi'$. Moreover, all these sets can be determined by a single run of A over the tree t, i.e., in linear time.

In a second topdown pass over the tree t, we determine for every subtree (occurrence) s the subset $R(s) \subseteq \mathsf{cl}(S(s))$ containing all those ψ' which may occur in some proof of $t : \phi$. Then s is a match iff $\bullet \in R(s)$. For a closed set of subformulas B, we introduce the auxiliary function core_B which takes a subformula ψ' of ϕ and returns the set of subformulas in B which potentially contribute to proofs of ψ'. So, $\mathsf{core}_B(\psi') = \{\psi'\} \cup \mathsf{core}'_B(\psi')$ where $\mathsf{core}'_B(\bullet) = \mathsf{core}'_B(\top) = \mathsf{core}'_B(a\langle F\rangle) = \mathsf{core}'_B(*\langle F\rangle) = \emptyset$ and:

$$\begin{array}{lll}
\mathsf{core}'_B(\mu x.\psi') & = \mathsf{core}_B(\psi') & \\
\mathsf{core}'_B(x) & = \mathsf{core}_B(\psi') & \text{if } \mu x.\psi' \in B \\
\mathsf{core}'_B(\psi_1 \wedge \psi_2) & = \mathsf{core}(\psi_1)_B \cup \mathsf{core}_B(\psi_2) & \\
\mathsf{core}'_B(\psi_1 \vee \psi_2) & = \begin{cases} \mathsf{core}(\psi_i) & \text{if } \psi_{3-i} \notin B \\ \mathsf{core}(\psi_1) \cup \mathsf{core}(\psi_2) & \text{otherwise} \end{cases} &
\end{array}$$

Moreover, we set: $\mathsf{core}_B(R) = \bigcup_{\psi \in R} \mathsf{core}_B(\psi)$ for $R \subseteq B$.

The second pass over t starts at the root of t. There, we have: $R(t) = \mathsf{core}_B(\phi)$ for $B = \mathsf{cl}(S(t))$. Now assume we have already computed the set $R(s)$ for the occurrence s of a subtree $a\langle s_1 \ldots s_k\rangle$. Let $R' = R(s) \cap \Psi$ denote the set of subformulas in $R(s)$ of the form $a\langle F\rangle$ or $*\langle F\rangle$. Then $R(s_i) = \bigcup_{\psi' \in R'} R_{\psi'}(i)$ where $R_{\psi'}(i)$ equals the set of subformulas for the i-th child of s which may occur at s_i in a proof of $s : \psi'$. If $\psi' = a\langle f\rangle$ or $\psi' = *\langle f\rangle$ for a Presburger formula f, then we must compute the assignment to the global variables of f. In fact, *all* valid sub-formulas at child trees s_i contribute to this assignment. Therefore, $R_{\psi'}(s_i) = S(s_i)$ for all i. On the other hand, if $\psi' = a\langle r\rangle$ or $\psi' = *\langle r\rangle$ for a regular expression r, then $R_{\psi'}(s_i) = \mathsf{core}_{B_i}(R_i)$ where $B_i = \mathsf{cl}(S(s_i))$ and

$$R_i = \{\psi_i \mid \exists\, \psi_1 \ldots \psi_k \in \mathcal{L}(r) : \forall j : \psi_j \in S(s_j)\}$$

The set R_i denotes all subformulas provable for s_i which may contribute to the validation of r. From these, we take all the formulas $a\langle F\rangle$ or $*\langle F\rangle$ in $S(s_i)$ which may contribute to a proof of these. According to this definition, the sets $R_{\psi'}(s_i)$, $i = 1, \ldots, k$ can jointly be computed by a left-to-right followed by a right-to-left pass of a finite (string) automaton for r over the children of s. The case of negated regular expressions is treated analogously. Summarizing we conclude:

Theorem 9. *The set of matches of a fixpoint query ϕ in an input tree t can be computed in time linear in $|t|$. If ϕ is part of the input, the joint query complexity is $\mathcal{O}(|\phi|^2 \cdot |t|)$.* □

8 Conclusion

We have enhanced a simple fixpoint logic for unranked trees with Presburger constraints. For the basic decision problems such as satisfiability, membership and containment the resulting logic turned out to have comparable complexities to the fixpoint logic without Presburger constraints. Therefore, our logic is a promising candidate for a smooth enhancement of classical Schema and querying languages for XML documents.

It remains a challenging engineering problem to obtain an implementation of the new logic which behaves well on practical examples. Also, we would like to know more about the complexity of the satisfiability problem for other restrictions on the transition function of a PTA or the fixpoint formula to obtain further useful classes with efficient algorithms.

Since the class of tree languages defined by deterministic PTA is a strict superclass of the regular tree languages, we would also like to see other characterizations of this class.

References

1. L. Cardelli and G. Ghelli. A Query Language Based on the Ambient Logic. In *10th European Symposium on Programming (ESOP),* 1–22. LNCS 2028, 2001.
2. L. Cardelli and A. Gordon. Anytime, Anywhere: Modal Logics for Mobile Ambients. In *27th ACM Conf. on Principles of Programming Languages (POPL),* 365–377, 2000.
3. G. Conforti, O. Ferrara, and G. Ghelli. TQL Algebra and its Implementation (Extended Abstract). In *IFIP Int. Conf. on Theoretical Computer Science (IFIP TCS),* 422–434, 2002.
4. G. Conforti, G. Ghelli, A. Albano, D. Colazzo, P. Manghi, and C. Sartiani. The Query Language TQL. In *5th Int. Workshop on the Web and Databases (WebDB),* 2002.
5. S. Dal-Zilio, D. Lugiez, and C. Meyssonnier. A Logic you can Count on. In *31st ACM Symp. on Principles of Programming Languages (POPL),* 135–146, 2004.
6. M.J. Fischer and M.O. Rabin. Superexponential Complexity of Presburger Arithmetic. In *AMS Symp. on the Complexity of Computational Computational Processes. Vol. 7,* 27–41, 1974.
7. S. Ginsburg and E.H. Spanier. Semigroups, Presburger Formulas and Languages. *Pacific Journal of Mathematics,* 16(2):285–296, 1966.
8. G. Gottlob and C. Koch. Monadic Datalog and the Expressive Power of Languages for Web Information Extraction. In *PODS,* 17–28, 2001.
9. F. Klaedtke and H. Ruess. Parikh Automata and Monadic Second-Order Logics with Linear Cardinality Constraints. Technical Report 177, Institute of CS at Freiburg University, 2002.
10. O. Kupferman, U. Sattler, and M.Y. Vardi. The Complexity of the Graded μ-Calculus. In *18th Int. Conf. on Automated Deduction (CADE),* 423–437. LNCS 2392, 2002.
11. D. Lugiez and S. Dal Zilio. XML Schema, Tree Logic and Sheaves Automata. In *14th Int. Conf. on Rewriting Techniques and Applications (RTA),* 246–263. LNCS 2706, 2003.
12. W. Martens and F. Neven. Typechecking Top-down Uniform Unranked Tree Transducers. In *ICDT,* 64–78, 2003.

13. A. Neumann and H. Seidl. Locating Matches of Tree Patterns in Forests. TR 98-08, Trier, 1998.
14. F. Neven. Automata, Logic, and XML. In *16th Int. Workshop CSL,* 2–26. LNCS 2471, 2002.
15. F. Neven and T. Schwentick. Query Automata over Finite Trees. *TCS,* 275(1-2):633–674, 2002.
16. F. Neven and J. Van den Bussche. Expressiveness of Structured Document Query Languages Based on Attribute Grammars. *Journal of the ACM,* 49(1):56–100,2002.
17. J. Niehren and A. Podelski. Feature Automata and Recognizable Sets of Feature Trees. In *4th TAPSOFT,* 356–375. LNCS 668,1993.
18. C.H. Papadimitriou. On the Complexity of Integer Programming. *J. ACM,* 28(4):765–768, 1981.
19. R. J. Parikh. On Context-free Languages. *J. of the ACM,* 13(4):570–581, 1966.
20. M. Presburger. On the Completeness of a Certain System of Arithmetic of Whole Numbers in which Addition Occurs as the only Operation. *Hist. Philos. Logic,* 12:225–233, 1991. English translation of the original paper from 1929.
21. H. Seidl. Haskell Overloading is DEXPTME Complete. *Inf. Proc. Lett. (IPL),* 54:57–60, 1994.
22. H. Seidl, T. Schwentick, A. Muscholl. Numerical Document Queries. In *PODS,* 155–166, 2003.
23. H. Seidl and A. Neumann. On Guarding Nested Fixpoints. In *Ann. Conf. of the European Association of Logic in Computer Science (CSL),* 484–498. LNCS 1683, 1999.
24. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC,* 1–9, 1973.
25. Igor Walukiewicz. Pushdown Processes: Games and Model-Checking. *Information and Computation,* 164(2):234–263, 2001.

Games with Winning Conditions of High Borel Complexity*

Olivier Serre

LIAFA, Université Paris VII, 2, place Jussieu, case 7014, F-75251 Paris Cedex 05

Abstract. We first consider infinite two-player games on pushdown graphs. In previous work, Cachat, Duparc and Thomas [4] have presented a winning decidable condition that is Σ_3-complete in the Borel hierarchy. This was the first example of a decidable winning condition of such Borel complexity. We extend this result by giving a family of decidable winning conditions of arbitrary high finite Borel complexity. From this family, we deduce a family of decidable winning conditions of arbitrary finite Borel complexity for games played on finite graphs. The problem of deciding the winner for these winning conditions is shown to be non-elementary complete.

Keywords: Pushdown Automata, Two-player Games, Borel Complexity.

1 Introduction

Infinite two-player games have been intensively studied in the last few years. One of the main motivations is the strong relation that exists with verification questions and controller synthesis. For instance, μ-calculus model checking for finite graphs (respectively for pushdown graphs) is polynomially equivalent to the problem of deciding the winner in a game played on a finite graph [6] (respectively on a pushdown graph [14]). In addition, constructing a winning strategy is the same as synthesizing a discrete controller [1].

One important branch of game theory is developed in the framework of descriptive set theory in which the central question is determinacy, that is the existence of a winning strategy. One of the deepest results is due to Martin [9] and states that, for Borel winning conditions, games are determined. In computer science, the games considered are in general equipped with winning conditions of low Borel complexity and therefore trivially determined. Nevertheless, deciding the winner is, in many cases, a difficult problem. Since we are mostly interested in decidable games, it is natural to ask whether there exist decidable games of arbitrary high finite Borel complexity.

For finite graphs and for the natural conditions appearing in verification and model-checking, efficient algorithms are known to decide the winner and to

* This research has been partially supported by the European Community Research Training Network "Games and Automata for Synthesis and Validation" (GAMES), (contract HPRN-CT-2002-00283), see www.games.rwth-aachen.de.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1150–1162, 2004.

compute the associated winning strategies [11,15,7,13]. These winning conditions all belong to a low level of Borel hierarchy, namely to the boolean closure of the Borel class Σ_2.

In [12], Thomas proposes to study games with winning conditions of Borel level larger than 2. In the paper we answer this question by exhibiting a family of winning conditions on pushdown games that have an arbitrary high Borel complexity while remaining decidable. As a corollary, one obtains a similar result for games on finite graphs. In addition, these games have effective winning strategies.

The first results concerning high level Borel conditions come from pushdown games. In this model, the game graph is the infinite graph of the configurations of a pushdown process. Walukiewicz has shown that parity games on such graphs can be effectively solved [14]. For this model, higher level winning conditions exploiting the infinity of the stack become natural. In [4], Cachat, Duparc and Thomas have considered the following condition: Eve wins a play if and only if some configuration is infinitely often visited. They have shown that it is a decidable winning condition belonging to Σ_3. More recently, Bouquet, Serre and Walukiewicz have considered in [2] winning conditions that are boolean combinations of a Büchi condition with a condition called *unboundedness* that requires the stack to be unbounded. These winning conditions are closely related with the one of [4] and remain decidable [2]. A natural question was therefore to consider higher level winning conditions.

In this paper, we give a uniform answer to the question of [12] by providing a family of winning conditions of increasing finite Borel complexity. The main idea is to require the stack to converge to some limit and then to have additional conditions on the limit. To solve classical conditions on pushdown games one method consists in reducing the problem to a game on a finite graph [14, 2]. We will adapt this method and reduce the problem of deciding the winner in a pushdown game to the problem of deciding the winner in another pushdown game, equipped with a lower winning condition. Then the proof goes by induction.

From the proofs we also infer the effectiveness of the winning strategies. Whereas for previously studied winning conditions on pushdown games, the set of winning positions was regular, it is no longer the case here. The exact nature of these sets remains open. We further show that determining the winner for these high level Borel winning conditions is non-elementary complete. We also show that Eve has, from a winning position, a persistent strategy, that is a strategy using memory but such that the move given from some node is always the same for a given play.

The paper is organized as follows. In Section 2, we start with basic definitions on games and introduce the family of winning conditions that we will consider in the rest of the paper. In Section 3, we precise the Borel complexity of these winning conditions. In Section 4, we give the decidability results and constructions of these games. In Section 5 we show that deciding the winner for such

winning conditions is non-elementary complete. Finally, in Section 6 we discuss several points. Due to the page limit, the paper contains only proof sketches.

2 Definitions

Definition 1 *An alphabet A is a finite or infinite set of letters. A^* denotes the set of finite words on A, A^ω the set of infinite words on A and A^∞ the set $A^* \cup A^\omega$. The empty word is denoted by ε. For a word u, we denote its (possibly infinite) length by $|u|$.*

Definition 2 (Prefix) *Let $u \in A^*$ and $v \in A^\infty$. Then u is a prefix of v, denoted $u \sqsubseteq v$ if there exists some word $w \in A^\infty$ such that $v = u \cdot w$. For any word $u \in A^\infty$ there exists a unique prefix of length k for all $k \leq |u|$. This prefix is denoted by $u \restriction k$.*

Definition 3 (Limit of a sequence of finite words) *Let $(u_i)_{i \geq 1} \in (A^*)^{\mathbb{N}}$ be an infinite sequence of finite words. The limit $\lim_{i \geq 1} u_i$ of $(u_i)_{i \geq 1}$, is the maximal word satisfying the following: for each j there exists an index r such that the j-th letter of $\lim_{i \geq 1} u_i$ equals the j-th letter of u_p for every $p \geq r$. Note that the $\lim_{i \geq 1} u_i$ can be either finite or infinite.*

We recall now some basic definitions on games. For more details and basic results, we refer to [11,15].

Infinite two-player game. Let $G = (V, E)$ be a possibly infinite graph. Let $V = V_E \cup V_A$ be a partition of the nodes among two players, Eve and Adam. A *play* from some node v_0 proceeds as follows: if $v_0 \in V_E$, Eve chooses a successor v_1 such that $(v_0, v_1) \in E$. Otherwise, it is Adam's turn to choose a successor v_1. If there is no such v_1, then the play ends in v_0, otherwise the player to whom v_1 belongs tries to move to some v_2 and so on. Therefore a play starting from v_0 is a finite or infinite sequence $v_0 v_1 v_2 \cdots$ such that $(v_i, v_{i+1}) \in E$ for all i. In the case where the play is finite, we require that there is no $v \in V$ such that $(v_n, v) \in E$, if v_n was the last node of the play. A *partial play* is a prefix of a play.

A *winning condition* for Eve in G is a subset $\Omega \subseteq V^\omega$. An infinite play is winning for Eve with respect to the winning condition Ω if it belongs to Ω. A finite play is won by the player that has made the last move (in other words, the player that cannot move looses the play). In the rest of the paper, we will suppose that all plays are infinite as the games we consider can easily be reduced to equivalent games where all plays are infinite. An *infinite two-player game* is a tuple $\mathcal{G} = (G, V_E, V_A, \Omega)$ where $G = (V, E)$ is a graph, $V = V_E \cup V_A$ and Ω is a winning condition for Eve in G.

Strategies, winning positions. Let $\mathcal{G} = (G, V_E, V_A, \Omega)$ be an infinite two-player game. A strategy for Eve is a partial function $\varphi : V^* V_E \to V$ such that

for any partial play $\lambda \in V^+$ ending in some node v and such that $\varphi(\lambda) = v'$, we have $(v, v') \in E$. In other words, a strategy is a function that associates with any partial play a valid move in G. Eve *respects a strategy* φ in a play $\lambda = v_0 v_1 v_2 \cdots$ if for all $v_i \in V_E$, $v_{i+1} = \varphi(v_0 v_1 \cdots v_i)$. A strategy φ is a *winning strategy* for Eve from some position v_0 if Eve wins all the plays starting from v_0 and where she respects φ. A position v is *a winning position* for Eve if she has a winning strategy from it. Symmetrically one defines the strategies and winning positions for Adam. We will denote by W_E and W_A the respective sets of winning positions for Eve and Adam. Martin's determinacy theorem [9] tells that $V = W_E \cup W_A$ whenever $\Omega \subseteq V^\omega$ is a Borel set.

2.1 Games on Pushdown Graphs and the Winning Condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$

Pushdown process. A *pushdown process* is a tuple $\mathcal{P} = (Q, \Gamma, \bot, \hookrightarrow)$ where Q is the finite set of states, Γ is the finite set of stack symbols, $\bot \in \Gamma$ is a special stack symbol (bottom) and $\hookrightarrow$ is the transition relation. A configuration of $\mathcal{P}$ is a pair (q, u) with $q \in Q$ and $u \in (\Gamma \setminus \{\bot\})^* \bot$ (the top stack symbol is the leftmost symbol of u). The bottom stack symbol $\bot$ is never put nor removed from the stack. The transition relation that can be applied to the set of configurations of $\mathcal{P}$ have one of the following form (note that we push or pop only one letter at each transition.):

- $Push(q, b)$: $(p, a) \hookrightarrow (q, ba)$, where $p, q \in Q$, $a \in \Gamma$ and $b \in (\Gamma \setminus \{\bot\})$.
- $Pop(q)$: $(p, a) \hookrightarrow (q, \varepsilon)$, where $p, q \in Q$ and $a \in (\Gamma \setminus \{\bot\})$.

By $(p, v) \to (q, w)$ we mean that from the configuration (p, v) the pushdown process can go in one step to (q, w). Note that the emptiness test of the stack corresponds to a push rule with $a = \bot$. This naturally leads to associate with any pushdown process an infinite graph as follows:

Pushdown graph. With any pushdown process $\mathcal{P}$, one can associate a *pushdown graph* defined as the directed graph having the set of configurations of $\mathcal{P}$ as nodes and the edges of which are given by the relation $\to$.

To define a *two-player game* on a pushdown graph $G = (V, \to)$ associated with a pushdown process $\mathcal{P}$, we need a partition $Q_E \cup Q_A$ of the set of states Q. From this partition follows a partition $V_E \cup V_A$ of the nodes V of G among the two players: the nodes of Eve are those whose control state belongs to Q_E and the others are Adam's nodes.

From now on we assume that we are given a pushdown process $\mathcal{P} = (Q, \Gamma, \bot, \hookrightarrow)$ and a partition $Q_E \cup Q_A$ of Q.

For a node $v = (p, u)$ of V we define $|v|$ to be the length of u. In a play $v_1 v_2 v_3 \cdots$, the stack is *strictly unbounded* if the stack size converges to $+\infty$. More formally we require that for all $k \geq 0$, there exists some i such that: for all $j \geq i$, $|v_j| > k$.

If the stack in a play $v_1 v_2 \ldots$ is strictly unbounded, we will consider its *limit* $u = a_1 a_2 a_3 \cdots$, $a_i \in \Gamma$, which is formally defined by: for all $k \geq 1$, there

exists some i such that for all $j \geq i$, there exists $w_j \in \Gamma^*$ such that $v_j = (p_j, w_j \cdot a_1 \cdots a_k)$.

Let us now recall the classical notion of deterministic pushdown automaton:

Deterministic pushdown automaton. A *deterministic pushdown automaton* with input from Σ^∞ is a tuple $\mathcal{A} = (Q, \Gamma, \Sigma, \bot, q^{in}, \delta)$, where Q is the finite set of states, Γ is the finite set of stack symbols, $\bot \in \Gamma$ is a special stack symbol (bottom), Σ is the input alphabet, $q^{in} \in Q$ is the initial state and $\delta : Q \times \Gamma \times \Sigma \to M$ is the transition function where $M = \{pop(q) \mid q \in Q\} \cup \{push(q, a) \mid q \in Q,\ a \in \Gamma\}$. A *run* of $\mathcal{A}$ on an infinite word $u = a_0 \cdot a_1 \cdots$ starts from the configuration $(q^{in}, \bot)$. $\mathcal{A}$ starts reading a_0 and applies the rule corresponding to $\delta(q^{in}, \bot, a_0)$ (that is the control state and the stack contents are changed), then it reads a_1 and so on. One can in addition equip such an automaton with a classical acceptance condition, for instance a Büchi condition.

Now let us consider a collection $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of deterministic pushdown automata. Let $\mathcal{B}$ be a deterministic pushdown automaton equipped with a Büchi acceptance condition. We require the following properties on the input and stack alphabets of $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{B}$:

- $\mathcal{A}_1$ has Γ as input alphabet.
- For all $i \geq 1$, the stack alphabet of $\mathcal{A}_i$ is the input alphabet of $\mathcal{A}_{i+1}$.
- The stack alphabet of $\mathcal{A}_n$ is the input alphabet of $\mathcal{B}$.

For given $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{B}$ we define the winning condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ as follows. Eve wins an infinite play in a pushdown graph G associated with a pushdown process $\mathcal{P}$, if and only if the following conditions are satisfied:

- The stack of $\mathcal{P}$ is strictly unbounded and therefore the sequence of stack contents converges to some limit $u_0 \in \Gamma^\omega$.
- For all $1 \leq i \leq n$, when $\mathcal{A}_i$ reads u_{i-1}, its stack is strictly unbounded and the sequence of stack contents converges to some limit u_i.
- $\mathcal{B}$ accepts u_n.

In particular, Eve wins an infinite play in G with the winning condition $\Omega_{\mathcal{B}}$ if and only if the stack is strictly unbounded and converges to some limit accepted by $\mathcal{B}$. For instance, if $\mathcal{B}$ accepts all infinite words, $\Omega_{\mathcal{B}}$ is the winning condition for Adam considered in [4].

2.2 Games on Finite Graphs and the Winning Condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$

We give now a version of the winning condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ for a game played on a finite game graph $G = (V = V_E \cup V_A, E)$.

Let us consider a collection $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of deterministic pushdown automata. Let $\mathcal{B}$ be a deterministic pushdown automaton equipped with a Büchi acceptance condition. We require the same properties on the input and stack alphabets of $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{B}$ as in the preceding case. In addition, we require that $\mathcal{A}_1$ (we set $\mathcal{A}_1 = \mathcal{B}$ if $n = 0$) has V as input alphabet.

$\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{B}$ induce a winning condition on G denoted by $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ and defined as follows. Eve wins a play in G if and only if the following conditions are satisfied:

- The play is infinite and is therefore a word $u_0 \in V^\omega$.
- For all $1 \leq i \leq n$, when $\mathcal{A}_i$ reads u_{i-1}, its stack is strictly unbounded and the sequence of stack contents converges to some limit u_i.
- $\mathcal{B}$ accepts u_n.

In particular, Eve wins an infinite play in G with the winning condition $\Omega_\mathcal{B}$ if and only if the play seen as an element of V^ω is accepted by $\mathcal{B}$. For instance, such a condition can require that a play in G is won if and only if any partial play contains more nodes from V_E than V_A.

Let $G = (V_E \cup V_A, E)$ be a finite game graph equipped with the winning condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ (with $n \geq 1$). Let $\mathcal{A}_1 = (Q_1, V, \Gamma_1, q^{in}_{\mathcal{A}_1}, \delta)$. Let us define a pushdown process $\mathcal{P} = (Q, \Gamma, \bot, \hookrightarrow)$ where $Q = V \times Q_1$, $\Gamma = \Gamma_1$ and $\hookrightarrow$ is such that:

- $((v,q),a) \hookrightarrow ((v',q'),\varepsilon)$ if and only if $(v,v') \in E$ and $\delta(q,v,a) = pop(q')$.
- $((v,q),a) \hookrightarrow ((v',q'),ba)$ if and only if $(v,v') \in E$ and $\delta(q,v,a) = push(q',b)$.

Let $G \otimes \mathcal{A}_1$ be the pushdown graph induced by $\mathcal{P}$ equipped with the partition $Q = Q_E \cup Q_A$ where $Q_E = V_E \times Q_1$ and $Q_A = V_A \times Q_1$.

The following lemma reduces the game on the finite graph to a pushdown game with a simpler winning condition:

Lemma 1. *Let G = (V,E) be a finite game graph. Eve wins in G from some position $v \in V$ with the winning condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ if and only if she wins in $G \otimes \mathcal{A}_1$ from $((v, q^{in}_{\mathcal{A}_1}), \bot)$ with the winning condition $\Omega_{\mathcal{A}_2 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ (respectively Büchi if $n = 0$).*

3 Borel Complexity

3.1 Borel Hierarchy

Let Σ be a (possibly infinite) alphabet. We consider the set Σ^ω of infinite words on the alphabet Σ and we equip it with the usual Cantor topology where the open sets are those of the form $W \cdot \Sigma^\omega$ where $W \subseteq \Sigma^*$ is a language of finite words on the alphabet Σ. The *finite Borel hierarchy* $(\Sigma_1, \Pi_1), (\Sigma_2, \Pi_2), \cdots$ is inductively defined as follows:

- $\Sigma_1 = \{W \cdot \Sigma^\omega \mid W \subseteq \Sigma^*\}$ is the set of open sets.
- For all $n \geq 1$, $\Pi_n = \{\overline{S} \mid S \in \Sigma_n\}$ consists of the complements of Σ_n-sets.
- For all $n \geq 1$, $\Sigma_{n+1} = \{\bigcup_{i \in \mathbb{N}} S_i \mid \forall i \in \mathbb{N},\ S_i \in \Pi_n\}$ is the set of countable union of Π_n-sets.

3.2 Borel Complexity of a Winning Condition

Let Ω be a winning condition. Ω is a *Σ_n-winning condition* if and only if Ω is a Σ_n-set. In the following, we may consider winning conditions given in a more abstract way than subsets of infinite words. For instance, we may consider

a Büchi winning condition, that is a condition where we require to infinitely visit some final nodes. Such a condition can be defined independently of the graph (except that implicitly, a subset of final nodes has to be defined). Such an abstract condition is a *Σ_n-winning condition* if there exists some graph for which the set of winning plays is a Σ_n-set. Note also that we do not focus on the set of effective winning plays but on the set of winning plays (some may not be effectively possible). In the same way, one defines Π_n-winning conditions.

Example 1. Consider a Büchi winning condition (Eve wins if and only if she infinitely visits nodes belonging to some subset $F \subseteq V$). Such a condition is a Π_2-winning condition, as the set of winning plays for Eve is $\bigcap_{n\geq 0}[(V^j V^* F)V^\omega]$.

Consider now a strict unboundedness winning condition for pushdown games. The corresponding condition for Adam is the one considered in [4]: Adam wins if and only if some configuration (equivalently some stack size) is infinitely repeated. Therefore, if one denotes by V_n the set of configurations of stack size n, the set of winning plays for Adam is $\bigcup_{n\geq 0}\bigcap_{m\geq 0}[(V^m V^* V_n)V^\omega]$ Therefore, the set of winning plays for Adam is a Σ_3-set and thus the strict unboundedness winning condition is a Π_3-winning condition for Eve.

Borel sets can be equipped with a reduction notion, inducing a notion of completeness [5]:

Definition 4 (Wadge Reduction, Complete Sets) *We say that $X \subseteq \Sigma_X^\omega$ Wadge reduces to $Y \subseteq \Sigma_Y^\omega$, denoted $X \leq_W Y$, if and only if there exists a continuous function $f : \Sigma_X^\omega \to \Sigma_Y^\omega$ such that $X = f^{-1}(Y)$. If $X \leq_W Y$ and $Y \leq_W X$ then we say that X and Y are Wadge equivalent and we denote it by $X \equiv_W Y$. A set $S \in \Sigma_n$ is Σ_n-complete if and only if $X \leq_W S$ for all $X \in \Sigma_n$.*

Example 2. Let $\Sigma = \{a, b\}$. Let $X \subseteq \Sigma^\omega$ be the set of infinite words that contain infinitely many a. X is a Π_2-complete set. Effectively $X = \bigcap_{i\geq 0} \Sigma^{\geq i} a \Sigma^\omega$: X is a Π_2 set. Let Y be a Π_2 set on some alphabet Γ: $Y = \bigcap_{i\geq 0} Y_i$ for some family $(Y_i)_{i\geq 0}$ of open sets, where $Y_i = Z_i \Gamma^\omega$. We define a function $f : \Gamma^\omega \to \{a, b\}^\omega$ by setting $f(x) = a_1 \cdot a_2 \cdots$, where $a_i = a$ if and only if $x \restriction i \in Z_k \Gamma^*$, where k is the number of letters $a_1, a_2, \ldots, a_{i-1}$ equals to a. We have that $Y = f^{-1}(X)$ and that f is continuous.

3.3 Borel Complexity of $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$

In [5], Duparc has introduced an operation on sets that allows to define sets of arbitrary high Borel complexity in a uniform way. Once reading some special letter, the letter that was just before it is erased. Applying this operation to some set, gives, under several conditions, a set having a higher Borel complexity. This operation can be iterated and therefore allows to construct sets of arbitrary Borel complexity. The winning conditions $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ can simulate the n-times iterated version of the eraser operator. Intuitively, popping the top symbol of a stack is the same that erasing in a word the last letter.

We have the two following results concerning the Borel complexity of $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$:

Theorem 1 *For all n, there exists a collection of deterministic pushdown automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and a deterministic Büchi pushdown automaton $\mathcal{B}$ such that $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ is a Π_{n+3}-complete winning condition for* pushdown games.

Theorem 2 *For all n, there exists a collection of deterministic pushdown automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ and a deterministic Büchi pushdown automaton $\mathcal{B}$ such that $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ is a Π_{n+2}-complete winning condition for games on* finite graphs.

4 Decidability

In this section we show that pushdown games equipped with the winning condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ are decidable.

Theorem 3 *Let $\mathcal{P} = (Q, \Gamma, \bot, \hookrightarrow)$ be a pushdown process, let $Q_E \cup Q_A$ be a partition of Q and let $G = (V, \rightarrow)$ be the associated pushdown graph. For any collection $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of deterministic pushdown automata, for any deterministic Büchi pushdown automaton $\mathcal{B}$, and for all $q \in Q$, it is decidable whether Eve has a winning strategy from $(q, \bot)$ in the pushdown game $(G, V_E, V_A, \Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}})$.*

The proof follows from iterating the result of Proposition 1 below and finally using the known algorithms on Büchi pushdown games [14].

Consider automata $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{B}$ inducing a winning condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$. We reduce the problem of solving a game with the condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ to that of solving an exponentially larger pushdown game $\widetilde{G} \times \mathcal{A}$ equipped with the simpler winning condition $\Omega_{\mathcal{A}_2 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ if $n > 0$, resp. a Büchi condition if $n = 0$.

We define $\mathcal{A}$ to be $\mathcal{A}_1$ if $n > 0$ and to be $\mathcal{B}$ otherwise. In addition, $\Gamma_{\mathcal{A}}$ will denote the stack alphabet of $\mathcal{A}$, $q_{\mathcal{A}}^{in}$ its initial state and $\delta_{\mathcal{A}}$ its transition function. The game $\widetilde{G} \times \mathcal{A}$ is presented in Figure 1.

For readability we will use the abbreviations *CS* and *CM* for *Choose_sets* and *Choose_move*. We give here an intuitive meaning of this graph. Intuitively, a node $[(p, a, R), t, u]$ represents a position (p, av) in G, such that Eve can play so that, if the top symbol a is eventually popped, it leads to a node (r, v) for some $r \in R$. In addition reading v^R (the mirror image of v) by $\mathcal{A}$ leads to the configuration (t, u) of $\mathcal{A}$. Thus, $\widetilde{G} \times \mathcal{A}$ encodes an on-the-fly computation of $\mathcal{A}$. Such a node belongs to Eve if and only if $p \in Q_E$.

CS and *CM* nodes are for push moves: once the player to whom belongs $[(p, a, R), t, u]$ has decided to push b and to change the state to p', which is represented by the node $[CS(p, a, R, p', b), t, u]$, Eve, by moving to some node $[CM(p, a, R, p', b, S), t, u)]$, announces the set S of states that she can ensure to reach if b is eventually popped. Then, if Adam wants b to be popped he moves to some state $[(s, a, R), t, u]$ for some $s \in S$. Otherwise, if he does not want

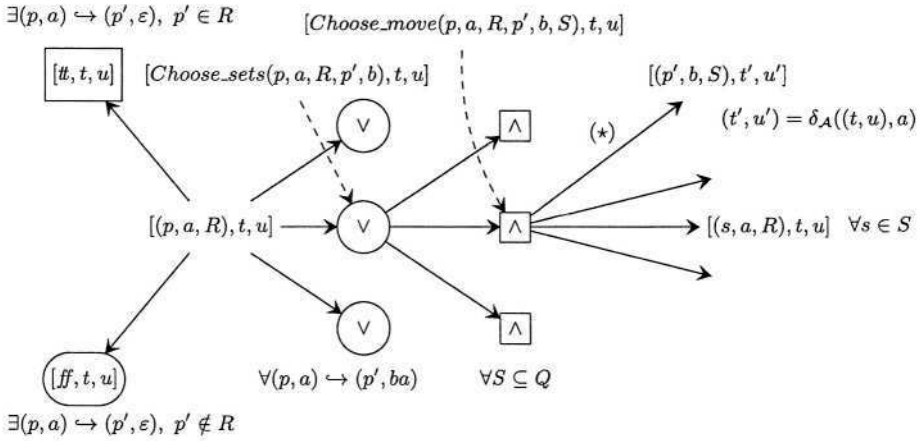


Fig. 1. $\widetilde{G} \times \mathcal{A}$: oval nodes belong to Eve, square for Adam.

to pop b or if he believes that S is incorrect, he moves to $[(p',b,S),t',u']$ (the edge he follows is called a *push edge*). Intuitively in that case, a is fixed forever and therefore, the on-the-fly computation of $\mathcal{A}$ must be updated: (t',u') is the configuration reached in $\mathcal{A}$ from (t,u) by reading a. In the special case where $n = 0$, $\mathcal{A} = \mathcal{B}$ is equipped with a Büchi condition. Therefore, a push edge is marked final ($\star$ on the figure) if and only if t' is a final state of $\mathcal{B}$.

We have the following key result:

Proposition 1 *Eve wins from $(q,\bot)$ in the game played on G with the winning condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ if and only if she wins from $[(q,\bot,\emptyset),q_{\mathcal{A}}^{in},\bot]$ in the pushdown game $\widetilde{G} \times \mathcal{A}$ equipped with the winning condition $\Omega_{\mathcal{A}_2 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ if $n > 0$ (resp., equipped with the Büchi condition if $n = 0$).*

Sketch of proof. Let Λ be some play in the pushdown game.

Λ can be represented graphically by the evolution of the stack during the play. Such a representation can be decomposed in a unique way into bumps (from some level the stack increases and eventually return to the level) and push moves that leaves some level forever. Such a decomposition is called a *P/B Factorization.* Figure 2 gives an example of such a factorization for a prefix of some play.

A play in $\widetilde{G} \times \mathcal{A}$ can be considered as a P/B factorization where in addition one performs an on-the-fly computation of $\mathcal{A}$ on some prefix of the limit (once some push move had been made the level will no longer be visited and therefore some prefix of the limit is known). In the case $n = 0$, the strict unboundedness and the acceptance of the limit is ensured by the fact that a final state only appears after a push move (implies strict unboundedness) inducing a prefix leading to a final state in $\mathcal{B}$. From these remarks follows easily the direct implication. In the case $n > 0$, the winning condition $\Omega_{\mathcal{A}_2 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$ implies unboundedness

($\mathcal{A}$'s stack has to be unbounded which implies that an infinite number of push edges were taken in $\widetilde{G} \times \mathcal{A}$). Cascade acceptance by $\mathcal{A}_1, \ldots, \mathcal{A}_n, \mathcal{B}$ is due to the on-the-fly computation of $\mathcal{A}_1$ that implies a cascade acceptance by $\mathcal{A}_2, \ldots, \mathcal{A}_n, \mathcal{B}$.

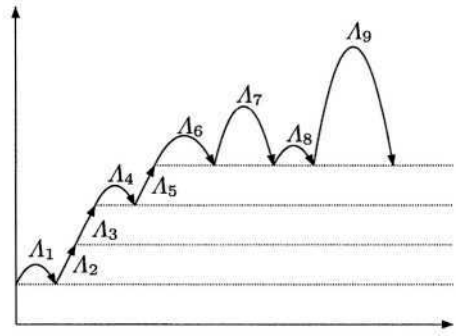


Fig. 2. P/B Factorization

For the converse implication, one notes that winning in $\widetilde{G} \times \mathcal{A}$, allows to have a winning strategy in the original pushdown game. For this, one needs to use a stack as memory that stores partial plays in $\widetilde{G} \times \mathcal{A}$ and allows to reconstruct the bumps (recall that $\widetilde{G} \times \mathcal{A}$ only represents the factorization). This direction is much more technical than the preceding one. This construction also implies effectivity of the winning strategy in the original game.

The result for games on finite graphs, is shown similarly, using Lemma 1 and Theorem 3.

5 Complexity

We first start with some definitions:

Definition 5 *Let $h, N \geq 0$. $tow(h, N)$ is defined inductively by:*

- $tow(0, N) = N$.
- $tow(h, N) = 2^{tow(h-1,N)}$ *for* $h \geq 1$.

Definition 6 (h-DEXPTIME) *Let consider a problem P and a deterministic Turing Machine deciding in $\mathcal{O}(tow(h, N))$ steps whether some instance of the problem P is true, where N is polynomial in the size of the instance. Then the problem P belongs to the class h-DEXPTIME.*

We have the following results for pushdown games equipped with winning conditions of the form $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_k \triangleright \mathcal{B}}$:

Proposition 2 *Let $\mathcal{A}_1, \ldots, \mathcal{A}_k$ be a collection of deterministic pushdown automata, and let $\mathcal{B}$ be a deterministic Büchi pushdown automaton. Let $\mathcal{G}$ be a pushdown game equipped with the winning condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_k \triangleright \mathcal{B}}$. Deciding the winner in such a game is an $(k+2)$-DEXPTIME problem.*

Proof. By induction on k and noting that the construction given in the proof of Proposition 1 induces an exponential blow up, and that deciding the winner in a Büchi pushdown game is a DEXPTIME problem.

Proposition 3 *The problem of deciding the winner in a pushdown game equipped with a winning condition of the form* $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_k \triangleright \mathcal{B}}$, *with* $k \geq 0$, *is a* $(k+1)$*-DEXPTIME hard problem.*

Therefore we have the following result:

Theorem 4 *The problem of deciding the winner in a pushdown game (resp. a finite game graph) equipped with a winning condition of the form* $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_k \triangleright \mathcal{B}}$ *is non-elementary complete (in the size of the winning condition).*

6 Winning Positions and Strategies

We first give some results on the form of the set of winning positions for pushdown games equipped with the winning condition $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$.

In [10,3], it is shown that the set of winning positions in a parity pushdown game is a regular language. In fact, using the same techniques, one can prove a similar result for various winning conditions, as for instance for unboundedness or strict unboundedness.

For the conditions we study in this paper, the set of winning positions may not be regular. For instance, every deterministic context-free language may occur as a winning set:

Proposition 4 *Let* $\mathcal{A}$ *be some deterministic pushdown automaton on finite words. There exists a deterministic Büchi automaton* $\mathcal{B}$, *a pushdown process* $\mathcal{P} = (Q, \Gamma, \bot, \hookrightarrow)$, *a state* $q \in Q$ *and a partition* $Q = Q_E \cup Q_A$ *such that, in the induced pushdown game equipped with the winning condition* $\Omega_{\mathcal{B}}$, *the set* $\{u \mid (q,u) \in W_E\}$ *is exactly the set of words recognized by* $\mathcal{A}$.

First, let us recall what a persistent strategy is:

Definition 7 (Persistent strategy) *[8] A strategy* φ *for Eve is persistent if for each play* $v_1 v_2 \ldots v_k$ *played by Eve according to this strategy, if* $v_i = v_j$, *for some* $1 \leq i, j < k$, *and* v_i *is a vertex where Eve is to move, then* $v_{i+1} = v_{j+1}$.

In other words, a persistent strategy may require memory but once a choice is made, it is done forever. For the winning conditions of the form $\Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}}$, we show that Eve has a persistent winning strategy. More precisely:

Theorem 5 *Let* $\mathcal{P} = (Q, \Gamma, \bot, \hookrightarrow)$ *be a pushdown process, let* $Q_E \cup Q_A$ *be a partition of* Q *and let* $G = (V, \rightarrow)$ *be the associated pushdown graph. For any collection* $\mathcal{A}_1, \ldots, \mathcal{A}_n$ *of deterministic pushdown automata, for any deterministic Büchi pushdown automaton* $\mathcal{B}$, *Eve has a persistent winning strategy from any winning position in the pushdown game* $\mathcal{G} = (G, V_E, V_A, \Omega_{\mathcal{A}_1 \triangleright \cdots \triangleright \mathcal{A}_n \triangleright \mathcal{B}})$.

7 Conclusion

We have provided a family of winning conditions that have an arbitrary high Borel complexity while remaining decidable for pushdown games and games on finite graphs. Deciding the winner for such a winning condition is an non-elementary complete problem. In addition, for pushdown games, it gives an example of decidable winning conditions inducing non regular sets of winning positions. The exact form of the winning sets remains open. Finally, we have shown that there are persistent winning strategies for pushdown games equipped with these winning conditions. The existence of positional strategies remains open.

Acknowledgments. I gratefully acknowledge Jacques Duparc for suggesting me to study this family of conditions. His advices and knowledge of Borel complexity were very important in this research. I would also like to express my thanks to Anca Muscholl for her help while writing this paper.

References

1. A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controlers with partial observation. *Theoretical Computer Science,* 303(1):7–34, 2003.
2. A. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with the unboundedness and regular conditions. In *Proceedings of FST TCS 2003,* volume 2914 of *Lecture Notes in Computer Science,* pages 88–99. Springer, 2003.
3. T. Cachat. Uniform solution of parity games on prefix-recognizable graphs. volume 68 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 2002.
4. T. Cachat, J. Duparc, and W. Thomas. Solving pushdown games with a Σ_3 winning condition. In *Proceedings of CSL 2002,* volume 2471 of *Lecture Notes in Computer Science,* pages 322–336. Springer, 2002.
5. J. Duparc. Wadge hierarchy and Veblen hierarchy. part I: Borel sets of finite rank. *Journal of Symbolic Logic,* 66(1):56–86, 2001.
6. E.A. Emerson, C.S. Jutla, and A.P. Sistla. On model-checking for the mu-calculus and its fragments. *Theoretical Computer Science,* 258(1-2):491–522, 2001.
7. M. Jurdziński. Small progress measures for solving parity games. In *Proceeding of STACS 2000,* volume 1770 of *Lecture Notes in Computer Science,* pages 290–301. Springer-Verlag, 2000.
8. J. Marcinkowski and T. Truderung. Optimal complexity bounds for positive LTL games. In *Proceedings of CSL 2002,* volume 2471 of *Lecture Notes in Computer Science,* pages 262–275. Springer, 2002.
9. D.A. Martin. Borel determinacy. *Annals of Mathematics,* 102(363-371), 1975.
10. O. Serre. Note on winning positions on pushdown games with ω-regular conditions. *Information Processing Letters,* 85:285–291, 2003.
11. W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of STACS '95,* volume 900 of *Lecture Notes in Computer Science,* pages 1–13. Springer, 1995.
12. W. Thomas. Infinite games and verification (extended abstract of a tutorial). In *Proceedings of CAV 2002,* volume 2404 of *Lecture Notes in Computer Science,* pages 58–64. Springer, 2002.

13. J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *Proceedings of CAV 2000,* volume 1855 of *Lecture Notes in Computer Science,* pages 202–215. Springer-Verlag, 2000.
14. I. Walukiewicz. Pushdown processes: games and model checking. *Information and Computation,* 157:234–263, 2000.
15. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science,* 200(1-2):135–183, 1998.

Propositional PSPACE Reasoning with Boolean Programs Versus Quantified Boolean Formulas*

Alan Skelley**

Department of Computer Science, University of Toronto
10 King's College Road,
Toronto, ON M5S 3G4 Canada
skelley@acm.org

Abstract. We present a new propositional proof system based on a somewhat recent characterization of polynomial space (PSPACE) called Boolean programs, due to Cook and Soltys. The Boolean programs are like generalized extension atoms, providing a parallel to extended Frege. We show that this new system, BPLK, is polynomially equivalent to the system *G,* which is based on the familiar but very different quantified Boolean formula (QBF) characterization of PSPACE due to Stockmeyer and Meyer. This equivalence is proved by way of two translations, one of which uses an idea reminiscent of the ϵ-terms of Hilbert and Bernays.

1 Introduction

When formulated in a Gentzen sequent style, many known propositional proof systems can be seen to be very similar, with the only difference between them being the computational power of what can be written at each line of the proof (or alternatively, what is allowed in the **cut** rule). Examples are Boolean formulas in Frege systems, single literals in resolution, Boolean circuits in extended Frege systems. Another example is the system G [9], which is a sequent-based system where formulas in the sequents are quantified boolean formulas (QBFs). These formulas have propositional variables and also propositional quantifiers. In this case, then, since evaluating QBFs is PSPACE-complete [12], the computational power that can be harnessed in sequents is PSPACE. We can restrict G to G_i by restricting the number of alternations of quantifiers allowed in the formulas, and the reasoning power is then that of Σ_i^p predicates. G and its subsystems are related to Buss' theories U_2^1, S_2^i and T_2^i [1] by translation and provability of reflection principles [8],[10].

Boolean programs were introduced by Cook and Soltys in [3]. A Boolean program defines a sequence of *function symbols,* where each function symbol is defined using a boolean formula that can include, in addition to the arguments to the function, invocations of the previously defined symbols. The authors of that

* An expanded form [11] of this paper is available from ECCC in the 'Theses' section
** Research supported by Canadian Natural Sciences and Engineering Research Council grant PGSA–208264–1998

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1163–1175, 2004.

paper showed that the problem of evaluating an invocation of a function symbol defined in this way, given the inputs and the Boolean program, is PSPACE complete. The question, then, is whether a proof system formulated around Boolean programs would be equivalent to *G*. For this to occur, not only would Boolean programs and quantified Boolean formulas need to characterize the same complexity class, but there would need to be an effective way of translating between the two.

This paper answers that question in the affirmative. We define a new system BPLK in a straightforward way to take advantage of the expressive power of Boolean programs, and give translations in both directions between it and *G*. Although it is possible to restrict Boolean programs to be equivalently expressive as Σ_i^q QBFs (i.e. of a particular number of alternations), we do not address the possibility of matching G_i by a subsystem of BPLK; this would probably be much more technical but not much more interesting. And although BPLK seems convenient to work with in practice, we do not consider issues of its precise relative efficiency to *G*. It is hoped only that a new and exotic PSPACE proof system to contrast with *G* may be illuminating.

2 Preliminaries

We assume standard terminology about propositional proof systems and the sequent calculus PK, for example from [7]. Recall that the inference rules are **weakening, exchange, contraction,** left- and right- introduction rules for each connective, and **cut.** *G* is the PK with the addition of left- and right- introduction rules for the propositional quantifiers. We adopt the convention of separate sets of free and bound variables as in [2] and call a *semiformula* any formula containing a free occurrence of a bound variable. It should be noted that although *G* and its subsystems derive tautological statements of quantified propositional logic, in this paper we shall consider them only as proof systems for propositional tautologies.

2.1 Boolean Programs

Boolean programs were introduced in [3] and are a way of specifying Boolean functions. They are something like a generalization of the technique of using new atoms to replace part of a Boolean formula, which idea is the basis of extended Frege systems. The following definition is from that thesis:

Definition 1 (Cook-Soltys). *A* Boolean Program *P is specified by a finite sequence* $\{f_1, ..., f_m\}$ *of function symbols, where each symbol* f_i *has an associated arity* k_i*, and an associated defining equation*

$$f_i(\overline{p_i}) := A_i$$

where $\overline{p_i}$ *is a list* $p_1, ..., p_{k_i}$ *of variables and* A_i *is a formula all of whose variables are among* $\overline{p_i}$ *and all of whose function symbols are among* $f_1, ..., f_{i-1}$*. In*

this context the definition of a formula *is expanded to allow the use of function symbols as connectives.*

The semantics are as for propositional formulas, except that when evaluating an occurrence $f_i(\overline{\phi})$ of a function symbol, the value is defined, using the defining equation, to be $A_i(\overline{\phi})$. There is no free/bound distinction between variables in the language of Boolean programs.

An interesting property of Boolean programs that demonstrates their comparability to quantified Boolean formulas is the following theorem from [3]:

Theorem 2 (Cook-Soltys). *A Language L is in PSPACE iff L is computed by some uniform polynomial size family of Boolean programs.*

2.2 Notational Conventions

We shall use the following conventions of notation: Lower case Latin letters will represent atoms, with $x, y, z, \ldots$ reserved for bound variables, and with the further exception of $f, g, h, \ldots$ to be used for function symbols. Capital letters and lower case Greek letters will be used for formulas. An overline indicates a list: $\overline{a}$ is a list of variables $(a_1, \ldots)$ and $\overline{\overline{A}}$ is a list of lists of formulas $(\overline{A_1} = \{A_{1,1}, A_{1,2}, \ldots\}, \overline{A_2}, \ldots)$. A formula A may have free variables $\overline{p}$, and when we wish to emphasize that fact we shall write $A(\overline{p})$, although we may not explicitly display all free variables of A. $A(\overline{\phi})$ denotes the result of substituting the list of formulas ϕ for the free variables of A. Since we have separated bound and free variables, in the quantified case we are automatically assured that $\overline{\phi}$ is free for $\overline{p}$ in $A(\overline{p})$, which is to say that no free variables of $\overline{\phi}$ will end up bound by any of A's quantifiers in the substitution.

We shall use the following symbols: '=' and '$\supset$' are not in the language of either system we consider, but we shall use them as abbreviations. $\overline{A} = \overline{B}$ abbreviates $((\neg A_1 \vee B_1) \wedge (\neg B_1 \vee A_1) \wedge \ldots \wedge (\neg A_k \vee B_k) \wedge (\neg B_k \vee A_k))$. $A \supset B$ abbreviates $\neg A \vee B$. The symbol '$\equiv$' will be used to denote syntactic equality.

3 BPLK and *G*

In this section we introduce the sequent system BPLK, which is basically PK enhanced with the reasoning power of Boolean programs. We then state two lemmas about BPLK and G that show that certain classes of proofs are easy to find in the two systems, and which will simplify arguments later on. The proofs of these lemmas are technical and have been omitted.

Definition 3 (BPLK). *The system BPLK is like the propositional system PK, but with the following changes:*

1. *In addition to sequents, a proof also includes a Boolean program that defines functions. A BPLK-proof is a pair $\langle \pi, P \rangle$ of the proof (sequents) and the Boolean program defining the function symbols occurring in the sequents.*

2. *Formulas in sequents are formulas in the context of Boolean programs, as defined earlier.*
3. *If the Boolean program contains a definition of the form* $f(\overline{p}) := A(\overline{p})$, *then the new rules*

$$f : \textbf{left} \quad \frac{A(\overline{\phi}), \Gamma \longrightarrow \Delta}{f(\overline{\phi}), \Gamma \longrightarrow \Delta} \qquad \textit{and} \qquad f : \textbf{right} \quad \frac{\Gamma \longrightarrow \Delta, A(\overline{\phi})}{\Gamma \longrightarrow \Delta, f(\overline{\phi})}$$

may be used, where $\overline{\phi}$ *are precisely as many formulas as* $\overline{p}$ *are variables.*
4. *(***Substitution Rule***) The new inference rule* **subst**

$$\frac{\Delta(q, \overline{p}) \longrightarrow \Gamma(q, \overline{p})}{\Delta(\phi, \overline{p}) \longrightarrow \Gamma(\phi, \overline{p})}$$

may be used, where all occurrences of q *have been substituted for.*

Simultaneous substitutions can be simulated with several applications of **subst.** It is an interesting open problem to eliminate **subst** altogether, perhaps along the lines of the simulation of substitution Frege by extended Frege, as in [4].

Lemma 4 (Propositional Reasoning). *If* $\overline{a}$ *are variables and* $\overline{A}$ *are formulas, and if* $\Gamma'(\overline{a}) \longrightarrow \Delta'(\overline{a})$ *follows from* $\Gamma(\overline{a}) \longrightarrow \Delta(\overline{a})$ *via a proof of size* k, *then a proof of the sequent* $\Gamma'(\overline{A}) \longrightarrow \Delta'(\overline{A})$ *from the sequent* $\Gamma(\overline{A}) \longrightarrow \Delta(\overline{A})$ *can be found in time* $O(k(|\overline{A}|)^3)$ *(*$O(k(|\overline{A}|+|P|)^3)$ *in the case of BPLK, with P defining all relevant function symbols), and whose length is thus similarly bounded.*

Lemma 5. *Substitution is a derived rule in G (i.e. it can be simulated in G by a derivation of size polynomial in the size of the resulting sequent).*

4 BPLK P-Simulates *G*

In this section we give a procedure to translate a sequent in the language of *G* into a semantically equivalent one in the language of BPLK, along with an accompanying Boolean program. This new Boolean program defines function symbols that occur in the translated sequent, and which replace the quantifiers and bound variables from the original one. We then translate a *G*-proof line-by-line into the language of BPLK, and we show how to fill in the gaps to form a correct proof in the latter system. Incidentally, this technique could easily be adapted to provide an alternate proof of the PSPACE-completeness of Boolean programs, as the proof in [3] did not involve a reduction from QBFs.

We do not use Boolean programs to compute Skolem functions, which is what first springs to mind, because the same formula would have different translations depending on which side of the sequent it occurred in (because of the semantics of sequents). Instead, we adopt a translation similar to the ϵ-calculus of Hilbert and Bernays [5],[6].

4.1 Special Notation

In the translation in this section, we shall use function symbols whose names are based on formulas of G. Because of the delicate nature of these names, we shall be especially detailed and careful about substitutions and the names of free variables. We shall therefore, in this section only, modify our notation:

When a formula A is displayed as $A(\overline{p})$, we shall mean that $\overline{p}$ are *all* the free variables of A. We shall never use the notation $A(q)$ to mean a substitution, but instead that A has the single free variable q. All substitutions will be denoted with a postfix operator $[B/p]$ which means that the formula B (which may be a single variable) is substituted for p in the formula that precedes the operator. We may write several of these in a row, and the meaning is that they are performed in sequence from left to right. We may also interject free variable lists, as in the example $A(a,b)[B(q,r,s)/b](q,r,s,a)[C/s][D/r]$.

4.2 A Translation from the Language of G to That of BPLK

To aid in translating we introduce the following definition, which gives us a well-defined merging operator that says how to combine several Boolean programs into one, eliminating duplicate definitions:

Definition 6 (Merging $P \diamond Q$ of Boolean Programs). *If P and Q are Boolean programs, or at least fragments (not necessarily defining all function symbols used in definitions), then the merging $P \diamond Q$ of P and Q is obtained by first concatenating P and Q, and then deleting all but the first definition of each function symbol.*

We shall use this merging operator in the following translation, and later on in the actual simulation. However, whenever we merge two Boolean programs that both define a particular function symbol, it will always be the case that the two definitions are identical. Thus, which of the definitions is kept is immaterial.

Now we can present the actual translation, defined first for single formulas:

Definition 7 (Translation $\langle \ulcorner\phi\urcorner, P[\phi]\rangle$ of ϕ). *We recursively define a translation of a quantified propositional* semiformula *into a semantically equivalent quantifier-free* formula *in the language of BPLK, together with a Boolean program defining the function symbols in that formula.*

- *If ϕ is an atom p then $\langle \ulcorner\phi\urcorner, P[\phi]\rangle$ is $\langle \phi, \emptyset\rangle$.*
- *If ϕ is $\psi \wedge \theta$ ($\psi \vee \theta$) then $\langle \ulcorner\phi\urcorner, P[\phi]\rangle$ is $\langle \ulcorner\psi\urcorner \wedge \ulcorner\theta\urcorner, P[\psi] \diamond P[\theta]\rangle$ ($\langle \ulcorner\psi\urcorner \vee \ulcorner\theta\urcorner, P[\psi] \diamond P[\theta]\rangle$).*
- *If ϕ is $\neg\psi$ then $\langle \ulcorner\phi\urcorner, P[\phi]\rangle$ is $\langle \neg\ulcorner\psi\urcorner, P[\psi]\rangle$.*
- *If ϕ is $\exists x \psi(x, \overline{p})$, then $\langle \ulcorner\phi\urcorner, P[\phi]\rangle$ is $\langle f_\phi(\overline{p}), P[\psi] \diamond P'[\phi]\rangle$ Here, $P'[\phi]$ is a Boolean program fragment with the following two definitions:*

$$\epsilon_\phi(\overline{p}) := \ulcorner\psi\urcorner[1/x](\overline{p}) \quad \textit{and} \quad f_\phi(\overline{p}) := \ulcorner\psi\urcorner[\epsilon_\phi(\overline{p})/x](\overline{p}).$$

– *If* ϕ *is* $\forall x\psi(x,\overline{p})$, *then* $\langle \ulcorner\phi\urcorner, P[\phi]\rangle$ *is* $\langle f_\phi(\overline{p}), P[\psi] \diamond P'[\phi]\rangle$ *Here,* $P'[\phi]$ *is a Boolean program fragment with the following two definitions:*

$$\epsilon_\phi(\overline{p}) := \ulcorner\psi\urcorner[0/x](\overline{p}) \quad \textit{and} \quad f_\phi(\overline{p}) := \ulcorner\psi\urcorner[\epsilon_\phi(\overline{p})/x](\overline{p}).$$

Thus, in the case where a quantifier is the top-level connective of $\phi = Qx\psi(x,\overline{p})$, we first pick a function symbol whose name depends on the formula ϕ (i.e., including more deeply nested quantifiers and free variables). This choice encodes both which variable is the most newly quantified one, and also what kind of quantifier it is. We call this function symbol a witnessing function for ψ because its value will satisfy ψ if possible, in case Q is existential, or else it will falsify ψ if possible.

We then substitute this function symbol, applied to all the variables free in ϕ, into the *translation* of ψ. Finally, we define a new function symbol using the resulting formula, and use it instead of the formula. We do so because $\ulcorner\psi\urcorner[\epsilon_\phi(\overline{p})/x](\overline{p})$ has more occurrences of free variables than $\ulcorner\psi\urcorner(\overline{p})$, and so without the new function symbol, subsequent substitutions of this form may increase the length of the formula exponentially.

Note that if ϕ is quantifier-free, then it is unchanged by the translation and the Boolean program that results is empty.

Definition 8. *If* S *is the sequent* $\Gamma_1, ..., \Gamma_j \longrightarrow \Delta_1, ..., \Delta_k$ *then* $\langle \ulcorner S\urcorner, P[S]\rangle$ *is*

$$\langle \ulcorner\Gamma_1\urcorner, ..., \ulcorner\Gamma_j\urcorner \longrightarrow \ulcorner\Delta_1\urcorner, ..., \ulcorner\Delta_k\urcorner, \qquad P[\Gamma_1] \diamond ... \diamond P[\Delta_k]\rangle.$$

We call $P[\phi]$ or $P[S]$ the Boolean program *arising* from the translation. As a final lemma in this subsection, we observe that translations are polynomial size:

Lemma 9. *If* S *is a sequent in the language of* G, *then* $|\langle \ulcorner S\urcorner, P(S)\rangle| \in O(|S|^3)$.

4.3 A Simulation of *G* by BPLK

First, we observe that the translations defined above are semantically equivalent to the original formulas.

Lemma 10. *In the presence of the Boolean program* $P[\phi]$ *defining the function symbols as above (arising from the translation), a quantified Boolean formula* ϕ *is semantically equivalent to its translation* $\ulcorner\phi\urcorner$.

Now, the operations of substitution and translation do not commute directly even in the case of substituting only a single variable, so for example if $A(a) \equiv \exists x(a \vee x)$, then its translation is $\ulcorner A\urcorner(a) \equiv f_{\exists x(a \vee x)}(a)$. If we then substitute b for a we obtain $\ulcorner A\urcorner[b/a] \equiv f_{\exists x(a \vee x)}(b)$. On the other hand, if we did these things in the reverse order we would get $\ulcorner A[b/a]\urcorner \equiv f_{\exists x(b \vee x)}(b)$, which is different. A technical lemma is needed to resolve this difficulty:

Lemma 11. *Let $A(s,\overline{p},\overline{r})$ be a semiformula and $B(\overline{p},\overline{q})$ a formula, both in the language of G, so B is automatically free for s in A, and let $\overline{p}$ be all free variables except s that are common to A and B. (s may be in $\overline{q}$) Then proofs of*

$$\ulcorner A[B/s]\urcorner(\overline{p},\overline{q},\overline{r}) \longrightarrow \ulcorner A\urcorner[\ulcorner B\urcorner/s](\overline{p},\overline{q},\overline{r})$$

and

$$\ulcorner A\urcorner[\ulcorner B\urcorner/s](\overline{p},\overline{q},\overline{r}) \longrightarrow \ulcorner A[B/s]\urcorner(\overline{p},\overline{q},\overline{r})$$

can be found in time polynomial in the size of translations and the size of the Boolean program arising from them.

The (quite technical) proof of this lemma is by induction on the structure of A and has been omitted. The main result of the section follows:

Theorem 12. *If S is a quantifier-free sequent with a proof π_1 in G, then S has a BPLK-proof $\langle \pi_2, P[\pi_1]\rangle$ that, given π_1, can be found in time polynomial in $|\pi_1|$ (and thus has polynomial size).*

Proof. First of all, $P[\pi_1]$ is formed by merging the Boolean programs arising from translating all the sequents in π_1. More precisely, if π_1 is $S_1,...,S_k$, then $P[\pi_1] = P[S_1] \diamond ... \diamond P[S_k]$.

We then form π_2 directly by translating π_1 sequent-by-sequent into the language of BPLK, and when necessary adding some extra sequents between the translated ones. We have the following cases, depending on the inference rule used:

Observe that all initial sequents of G are their own translations, and are also initial sequents of BPLK.

If S is non-initial and is inferred by *any* rule except a quantifier introduction with hypothesis(es) T (and U), then $\ulcorner S\urcorner$ follows from $\ulcorner T\urcorner$ (and $\ulcorner U\urcorner$) by the same rule. This is because translations of identical formulas are identical, and the translation operator commutes with the connectives $\vee$, $\wedge$ and $\neg$.

If S is inferred from T by the introduction of a universal quantifier on the right, or that of an existential quantifier on the left, then considering the first case we have that T is $\Gamma \longrightarrow \Delta, C(a,\overline{p})$ and S is $\Gamma \longrightarrow \Delta, \forall x(C[x/a](x,\overline{p}))$. Their translations are thus $\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, \ulcorner C\urcorner(a,\overline{p})$ and $\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, f_{\forall x C[x/a]}(\overline{p})$, respectively. For convenience, let $A \equiv C[x/a]$. In this notation, $\ulcorner T\urcorner$ is $\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, \ulcorner A[a/x]\urcorner(a,\overline{p})$. First, $\ulcorner A[a/x]\urcorner \longrightarrow \ulcorner A\urcorner[\ulcorner a\urcorner/x]$ follows from lemma 11, taking $B \equiv a$, and $s \equiv x$. $\ulcorner T\urcorner$ and **cut** then yield $\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, \ulcorner A\urcorner[a/x]$. Next, observe that a cannot occur in Γ or Δ (or therefore their translations) due to the restriction on $\forall$: **right.** So, when we apply **subst** to this sequent, substituting $\epsilon_{\forall x A}(\overline{p})$ for a to obtain $\ulcorner\Gamma\urcorner \longrightarrow \ulcorner\Delta\urcorner, \ulcorner A\urcorner[\epsilon_{\forall x A}(\overline{p})/x]$, $\ulcorner\Gamma\urcorner$ and $\ulcorner\Delta\urcorner$ are unchanged. Finally, $\ulcorner A\urcorner[\epsilon_{\forall x A}(\overline{p})/x] \longrightarrow f_{\forall x A}(\overline{p})$ follows from the definition of the function symbol $f_{\forall x A}$ using the introduction **right** rule for that symbol. $\ulcorner S\urcorner$ follows with **weakening** and **cut.**

The interesting case is when S follows from T by the introduction of an existential quantifier on the right, or that of a universal quantifier on the left. The two cases are symmetrical, so consider the first. T is $\Gamma \longrightarrow \Delta, A(x,\overline{p})[B/x]$ and S is

$\Gamma \longrightarrow \Delta, \exists x(A(x,\overline{p}))$. The translations $\ulcorner T \urcorner$ and $\ulcorner S \urcorner$ are $\ulcorner \Gamma \urcorner \longrightarrow \ulcorner \Delta \urcorner, \ulcorner A[B/x] \urcorner$ and $\ulcorner \Gamma \urcorner \longrightarrow \ulcorner \Delta \urcorner, f_{\exists x A}(\overline{p})$. Thus, it suffices to prove $\ulcorner A[B/x] \urcorner \longrightarrow f_{\exists x A}(\overline{p})$ and apply **cut.** First, we derive $\ulcorner A[B/x] \urcorner \longrightarrow \ulcorner A \urcorner [\ulcorner B \urcorner / x]$ using lemma 11. The next task is to prove $\ulcorner A \urcorner [\ulcorner B \urcorner / x] \longrightarrow \ulcorner A \urcorner [\epsilon_{\exists x A}(\overline{p})/x]$. This sequent is proved by reasoning by cases about the value of $\epsilon_{\exists x A}(\overline{p})$ but this proof is omitted for brevity.

Lastly, $\ulcorner A \urcorner [\epsilon_{\exists x A}(\overline{p})/x] \longrightarrow f_{\exists x A}(\overline{p})$ follows from the definition of $f_{\forall x A}$.

Now, starting with $\ulcorner T \urcorner$ and using each of these last three sequents in turn with **weakening** and **cut**, one obtains the sequent $\ulcorner S \urcorner$, as desired.

Finally, in the case that S is quantifier-free (for example, the final sequent in a proof), then the translation of S is S, and this completes the proof of the theorem. □

5 *G* P-Simulates BPLK

In this section we define a translation of Boolean programs into formulas in the language of G, that is, using propositional quantifiers, inductively on the length of the program. To avoid exponential blowup of the translations, we must be careful to use exactly one previous translation at each step, so our construction produces one formula simultaneously translating all the function symbols.

The method we shall use to "multiplex" is inspired by a trick used by Stockmeyer and Meyer in [12] to show that the problem of evaluating a given quantified Boolean formula (QBF) is PSPACE-complete. The idea is to abbreviate $\phi(x_1, y_1) \wedge \phi(x_2, y_2)$ as $\forall x, y[((x = x_1 \wedge y = y_1) \vee (x = x_2 \wedge y = y_2)) \supset \phi(x, y)]$. This is not exactly what is needed but is quite close.

5.1 A Translation from the Language of BPLK to That of *G*

Reserve $\overline{v}$, $\overline{u}$, $\overline{z}$ of the bound type and $\overline{t}$, $\overline{d}$ of the free type as new variables not occurring in the BPLK-proof being translated. There will be variables v_A and t_A for each formula A in the language of BPLK, and we shall replace occurrences of function symbols by these two classes of variables as part of our translation using the two operators defined below. One is used to translate the Boolean program, and the other to translate the sequents of the BPLK proof.

Definition 13 (Hat and Corner Operators). *If $A(\overline{p})$ is a formula in the language of BPLK ($\overline{p}$ are the variables free in A), then let $\widehat{A}(\overline{p}, \overline{v})$ be the result of replacing, in A, every maximal subformula $f(\overline{B})$ of A whose main connective is a function symbol by the corresponding variable $v_{f(\overline{B})}$. This new formula will have some subset of the original $\overline{p}$ and also some v's as free variables. The corner operator (e.g. $\ulcorner A \urcorner$) is defined similarly, except that corresponding t variables are used instead of v's. When either operator is applied to a set of formulas, we mean that it should be applied to each formula in the set in turn.*

Example 14 (Hat Operator). If $\psi(\overline{p})$ is the formula $f(p_1) \vee g(f(p_1 \wedge p_2)) \wedge \neg p_3$, then $\widehat{\psi}(\overline{p}, \overline{v})$ is $v_{f(p_1)} \vee v_{g(f(p_1 \wedge p_2))} \wedge \neg p_3$.

Let us fix a Boolean program defining functions $f_1, f_2, \ldots$ such that each function symbol in the Boolean program is defined as $f_i(\overline{p}) := A_i(\overline{p})$. (In other words, A_i is the defining formula of f_i). Now, $\phi_k(\overline{\overline{z}}, \overline{u})$ will be the translation of the Boolean program up to and including the definition of f_k. The meaning of $\phi_k(\overline{z_1}, \ldots, \overline{z_k}, u_1, \ldots, u_k)$ will be that this formula is satisfied if and only if $f_1(\overline{z_1}) = u_1, \ldots$, and $f_k(\overline{z_k}) = u_k$.

Definition 15. *Consider a Boolean program defining $f_1 \ldots f_k$. Define $\phi_0 := 1$. Now, say the definition of f_i includes the function symbol occurrences $f_{j_1}(\overline{B_1})$, $\ldots$, $f_{j_m}(\overline{B_m})$ in order, some of which may be nested in others. (Here $j_1, j_2, \ldots, j_m$ are just the indexes of the function symbols, i.e., a sequence of m integers, not necessarily distinct, each smaller than i). Then define*

$$
\begin{aligned}
&\phi_i(\overline{z_1}, \ldots, \overline{z_i}, u_1, \ldots, u_i) := \\
&\quad \exists \overline{v_{f_j(B_j(\overline{z_i}))}} [\widehat{A_i}(\overline{z_i}, \overline{v_{f_j(B_j(\overline{z_i}))}}) = u_i \wedge \\
&\qquad \forall \overline{z'_1}, \ldots, \overline{z'_{i-1}}, u'_1, \ldots, u'_{i-1} [\phi_{i-1}(\overline{\overline{z'}}, \overline{u'}) \supset [\\
&\qquad\quad \left.\begin{array}{l} (\overline{z'_{j_1}} = \widehat{\overline{B_1(\overline{z_i})}} \supset u'_{j_1} = v_{f_{j_1}(B_1(\overline{z_i}))}) \wedge \\ (\overline{z'_{j_2}} = \widehat{\overline{B_2(\overline{z_i})}} \supset u'_{j_2} = v_{f_{j_2}(B_2(\overline{z_i}))}) \wedge \\ \vdots \\ (\overline{z'_{j_m}} = \widehat{\overline{B_m(\overline{z_i})}} \supset u'_{j_m} = v_{f_{j_m}(B_m(\overline{z_i}))}) \wedge \end{array}\right\} (*) \\
&\qquad\quad \left.\begin{array}{l} (\overline{z'_1} = \overline{z_1} \supset u'_1 = u_1) \wedge \\ \vdots \\ (\overline{z'_{i-1}} = \overline{z_{i-1}} \supset u'_{i-1} = u_{i-1}) \end{array}\right\} (**) \\
&\qquad\quad] \\
&\qquad] \\
&\quad].
\end{aligned}
$$

Lemma 16. *For each i, $\phi_i(\overline{\overline{z}}, \overline{u})$ is semantically equivalent to $f_1(\overline{z_1}) = u_1 \wedge \ldots \wedge f_i(\overline{z_i}) = u_i$. (The part $(*)$ above recursively uses ϕ_{i-1} to help evaluate the ith function symbol, and $(**)$ passes through the evaluation of the previous ones.)*

We can now define the translation of sequents. This translation is in the context of a BPLK-proof, so the Boolean program and the rest of the sequents in the proof are already fixed. Exactly which proof a particular translation is relative to is not indicated in the notation, but it will always be clear from the context.

Definition 17 (Translation $\ulcorner S \urcorner$ of the sequent S relative to π). *Fix a BPLK-proof π and its associated Boolean program defining $f_1, \ldots, f_k$. Let $\overline{f_{j_i}(\overline{C_i})}$ be a list of all subformulas in π whose main connective is a function symbol. ($\overline{C_i}$ are arguments to f_{j_i}, and again j_i are simply indexes).*

Then if S is the sequent $\Gamma \longrightarrow \Delta$, it is translated as the sequent $\ulcorner S \urcorner$:

$$\phi_k(0, ..., 0, \overline{\ulcorner C_1 \urcorner}, 0, ..., 0, d_1^1, ..., d_{j_1-1}^1, t_{f_{j_1}(\overline{C_1})}, d_{j_1+1}^1, ..., d_k^1), ...$$
$$\phi_k(0, ..., 0, \overline{\ulcorner C_m \urcorner}, 0, ..., 0, d_1^m, ..., d_{j_m-1}^m, t_{f_{j_m}(\overline{C_m})}, d_{j_m+1}^m, ..., d_k^m), \ulcorner \Gamma \urcorner \longrightarrow \ulcorner \Delta \urcorner.$$

Here the $\overline{\ulcorner C_i \urcorner}$ and the corresponding t's are in the correct places to be the arguments to, and the values of, the function symbol f_{j_i}. The d are dummy variables. We could use $t_{f_i(\overline{0})}$ instead of d_i^l (since d_i^l will be constrained to the value $f_i(\overline{0})$) but it will be convenient later on that the d's are distinct. We shall call the occurrences of ϕ_k above the prefix *of the translation, and the remainder the* suffix.

Now, these translations may have free variables that the original ones did not (t's and d's). We cannot, therefore, assert semantic equivalence of the two. However, we are concerned with proving valid sequents, and we can say something nearly as good:

The idea is that if the translation of a sequent is satisfied by some assignment, then either one of the t or d variables has an incorrect value, falsifying the corresponding instance of ϕ_k, or else they all have the correct values and the remainder of the translated sequent is satisfied. In that case, the original sequent is satisfied by the same assignment. Conversely, if the original sequent is valid, then every assignment to the translation will either falsify one of the ϕ_k's, or else all the t's will have the correct value and thus the remainder of the sequent will be satisfied. Therefore,

Claim. For any sequent S from the language of BPLK, S is valid if and only $\ulcorner S \urcorner$ is.

Lemma 18. *Let S be any sequent from the BPLK-proof $\langle \pi, P \rangle$. Then $|\ulcorner S \urcorner| \in O(|P|^2|\pi|^2)$.*

5.2 A Simulation of BPLK by *G*

We first show that proofs of sequents from two special classes are efficient to find.

Lemma 19 (Existence and Uniqueness Sequents). *The* existence sequents E_i*:*

$$\longrightarrow \forall \overline{z} \exists \overline{u} \phi_i(\overline{z}, \overline{u})$$

and uniqueness sequents U_i

$$\begin{aligned} \longrightarrow \forall \overline{z_1}, \overline{z_2}, \overline{u_1}, \overline{u_2} [\phi_i(\overline{z_1}, \overline{u_1}) \wedge \phi_i(\overline{z_2}, \overline{u_2}) \supset [\\ (\overline{z_{1,1}} = \overline{z_{2,1}} \supset u_{1,1} = u_{2,1}) \wedge \\ \vdots \\ (\overline{z_{1,i}} = \overline{z_{2,i}} \supset u_{1,i} = u_{2,i}) \\ ] \\ ] \end{aligned}$$

have proofs that can be found in polynomial time, and thus are of polynomial size.

The two parts of this lemma are proved by induction in parallel. One more lemma, analogous to lemma 11 of the previous section, is needed because substitution does not commute with translation. The proofs of these lemmas are omitted.

Lemma 20. *If $T(p)$ is a sequent in a BPLK-proof and ψ is a BPLK formula in which p does not occur, then a G-proof of $\ulcorner T(\psi)\urcorner$ from $\ulcorner T\urcorner(\ulcorner\psi\urcorner)$ can be found in time polynomial in the size of its endsequent.*

Finally, we can state and prove the main result:

Theorem 21. *If S has a BPLK-proof π_1, then S has a G-proof π_2 that, given π_1, can be found in time polynomial in $|\pi_1|$ (and thus has polynomial size).*

Proof. We construct π_2 directly by translating π_1, sequent-by-sequent, into the language of G, relative to the Boolean program of π_1. If necessary, we insert sequents to prove the translation of a sequent from the translations of its hypotheses.

First of all, if S is an initial sequent of BPLK, then it is function symbol free and so its translation is itself, and thus already an initial sequent of G.

Now consider a non-initial sequent S inferred from previous ones. If the inference was **weakening, contraction, cut,** or introduction of $\neg$, $\wedge$ or $\vee$, then the same rule yields $\ulcorner S\urcorner$.

If $S = T(\psi)$ is inferred from $T(p)$ by **subst,** then note that without loss of generality we may assume that p does not occur in ψ. Otherwise we could modify π_1 to perform **subst** twice; once to substitute $\psi(q)$ for p (q is a variable that does not occur in T) and then again to substitute p for q. To simulate the substitution in G, first use lemma 5 to substitute $\ulcorner\psi\urcorner$ for p in $\ulcorner T\urcorner(p,\vec{t})$, obtaining $\ulcorner T\urcorner(\ulcorner\psi\urcorner,\vec{t})$. Finally, apply lemma 20.

The last case in the proof is when S is inferred by f_i-introduction, introducing $f_i(\overline{B(\overline{p})})$. Then clearly the sequent $\phi_k(...),... \longrightarrow \ulcorner A_i(\overline{B})\urcorner(\overline{p},\overline{t}) = t_{f_i(\overline{B})}$, together with some propositional reasoning, will produce $\ulcorner S\urcorner$ (basically just by using **cut**). We derive the desired sequent as follows: First, the following is straightforward:

$$\phi_k(\overline{0},\overline{\ulcorner B\urcorner},\overline{u},t_{f_i(\overline{B})},\vec{d}) \longrightarrow \phi_i(\overline{0},\overline{\ulcorner B\urcorner},\overline{u},t_{f_i(\overline{B})},\vec{d}).$$

Next, we add the rest of the prefix:

$$\phi_k,... \longrightarrow \phi_i(\overline{0},\overline{\ulcorner B\urcorner},\overline{u},t_{f_i(\overline{B})},\vec{d}).$$

Expanding the ϕ_i,

$$\phi_k,... \longrightarrow \exists\overline{v}[\widehat{A_i(\overline{z_i})}(\overline{\ulcorner B\urcorner},\overline{v}) = t_{f_i(\overline{B})} \wedge ....].$$

Note that the A_i occurrence above contains t variables, from the $\overline{\ulcorner B \urcorner}$ substituted for the $\overline{z_i}$, and also v variables, from function symbols occurring in the definition of f_i. Uniqueness for ϕ_{i-1} and

$$\phi_k(\overline{0}, \overline{\ulcorner C(\overline{B}) \urcorner}, \overline{u}, t_{f_j(\overline{C(\overline{B})})}, \overline{d}) \longrightarrow \phi_{i-1}(\overline{0}, \overline{\ulcorner C(\overline{B}) \urcorner}, \overline{u}, t_{f_j(\overline{C(\overline{B})})}, \overline{d}),$$

one sequent for each function symbol occurrence $f_j(\overline{C(\overline{(z_i)})})$ in the definition of f_i, allow us to rename the $v_{f_j(\overline{C(\overline{z_i})})}$ in the occurrence of $\widehat{A_i}$ above to $t_{f_j(\overline{C(\overline{B})})}$, producing $\ulcorner A_i(\overline{B}) \urcorner$, and then we drop the existential quantifier and some conjuncts to get $\phi_k(...), ... \longrightarrow \ulcorner A_i(\overline{B}) \urcorner = t_{f_i(\overline{B})}$, which is the desired sequent.

Nearing the end of the proof now, if S is the last sequent of the proof, then it is function symbol-free. We need only remove the prefix from $\ulcorner S \urcorner$ to obtain S. The t variable corresponding to the outer-most function symbol occurrence in π_1 (there may be many outer-most occurrences) is defined by an occurrence of ϕ_k, but it is not used in the definition of any of the other t variables. We may thus use $\exists$: **left** on the t and the d's, followed by $\forall$: **left** on the B's and the 0's, to change this occurrence into $\forall \overline{\overline{z}} \exists \overline{u} \phi_k(\overline{\overline{z}}, \overline{u})$, which we can cut away with the existence sequent and **weakening.** We can now do the same for the next most outer function symbol occurrence, and so on. The resulting sequent at the end of this process is S, which completes the proof. □

Acknowledgment. The author wishes to thank the anonymous referees and, in the strongest possible terms, his supervisor, Stephen Cook, for countless helpful discussions.

References

[1] S. Buss. *Bounded Arithmetic.* Bibliopolis, Naples, 1986.

[2] Samuel R. Buss, editor. *Handbook of Proof Theory.* Elsevier Science B. V., Amsterdam, 1998.

[3] Stephen Cook and Michael Soltys. Boolean programs and quantified propositional proof systems. *Bulletin of the Section of Logic,* 28(3), 1999.

[4] Martin Dowd. Model theoretic aspects of $\mathrm{P} \neq \mathrm{NP}$. Typewritten manuscript, 1985.

[5] D. Hilbert and P. Bernays. *Grundlagen der Mathematik I.* Springer, Berlin, 1934.

[6] D. Hilbert and P. Bernays. *Grundlagen der Mathematik II.* Springer, Berlin, 1939.

[7] Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory.* Cambridge University Press, 1995.

[8] Jan Krajíček and Pavel Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic,* 54(3):1063–1079, 1989.

[9] Jan Krajíček and Pavel Pudlák. Quantified propositional calculi and fragments of bounded arithmetic. *Zeitschr. f. Mathematikal Logik u. Grundlagen d. Mathematik,* 36:29–46, 1990.

[10] Jan Krajíček and Gaisi Takeuti. On bounded Σ_1^1 polynomial induction. In S. R. Buss and P. J. Scott, editors, *FEASMATH: Feasible Mathematics: A Mathematical Sciences Institute Workshop,* pages 259–80. Birkhauser, 1990.

[11] Alan Skelley. Relating the PSPACE reasoning power of Boolean programs and quantified Boolean formulas. Master's thesis, University of Toronto, 2000. Available from ECCC in the 'theses' section.

[12] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Conference Record of Fifth Annual ACM Symposium on Theory of Computing,* pages 1–9, Austin, Texas, 30 April–2 May 1973.

LA, Permutations, and the Hajós Calculus

Michael Soltys

Department of Computing and Software, McMaster University
1280 Main Street West, Hamilton, Ontario L8S4K1, CANADA
soltys@mcmaster.ca

Abstract. **LA** is a simple and natural field independent system for reasoning about matrices. We show that **LA** extended to contain a matrix form of the pigeonhole principle is strong enough to prove a host of matrix identities (so called "hard matrix identities" which are candidates for separating Frege and extended Frege). **LAP** is **LA** with matrix powering; we show that **LAP** extended with quantification over permutations is strong enough to prove theorems such as the Cayley-Hamilton Theorem. Furthermore, we show that **LA** extended with quantification over permutations expresses **NP** graph-theoretic properties, and proves the soundness of the Hajós calculus. A corollary is that a fragment of Quantified Permutation Frege (a novel propositional proof system that we introduce in this paper) is p-equivalent of extended Frege. Several open problems are stated.

1 Introduction

The theory **LA** ([4,1,5]) is a field-independent logical theory for expressing and proving matrix properties. **LA** proves all the ring properties of matrices (e.g., $A(BC) = (AB)C$). In this paper, we restrict **LA** to the two element field GF(2).

While **LA** is strong enough to prove all the ring properties of matrices, its propositional proof complexity is low: all the theorems of **LA** translate into $\mathbf{AC}^0[2]$-Frege proofs (see [5] for this result, and [2] for the background). **LA** seems too weak to prove those universal matrix identities which require reasoning about inverses, e.g., $AB = I \supset BA = I$ (which we shall denote by IP_n, the Inversion Principle for $n \times n$ matrices), proposed by Cook as a candidate for separating Frege and extended Frege (this separation remains an important open problem of computer science).

In section 2 we present the theory **LA,** and several of its extensions. In section 3 we show that **LA** strengthened to contain the matrix form of the pigeonhole principle can prove IP_n. It was shown in [6] that a feasible bounded-depth Frege proof of IP_n would lead to a feasible bounded-depth Frege proof of the functional form of the pigeonhole principle, which is not possible, and hence no feasible bounded depth proofs of IP_n exist. Section 3 presents a weak converse of that result.

In section 4 we give a proof of the Cayley-Hamilton Theorem (CHT) based on induction over formulas with quantification over matrix permutations. This

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1176–1187, 2004.

improves the proof of the CHT given in [5], where we used quantification over general matrices. We call the theory that formalized the new proof $\exists P\mathbf{LAP}$ (it is defined in section 2).

In section 5 we show how to express **NP** and co-**NP** graph-theoretic properties in $\exists P\mathbf{LA}$ and $\forall P\mathbf{LA}$. In section 6, we prove the soundness of the Hajós calculus in $\forall P\mathbf{LA}$. In section 7 we obtain a corollary which states that a fragment of Quantified Permutation Frege—a novel proof system that we introduce in this paper—is equivalent to extension Frege. We end with a list of open problems in section 8.

2 The Theory LA and Its Extensions

LA is a three-sorted logical theory designed for reasoning about matrices. It is strong enough to prove all the ring properties of matrices (i.e., commutativity of matrix addition, associativity of matrix products, etc.). The original definition of **LA** had no quantification; in this paper we consider a conservative extension with bounded index quantifiers. This allows us to express that a given matrix is a permutation matrix. A full description of **LA** can be found in [4,1,5].

The three sorts are indices, field elements, and matrices. All the usual axioms for equality are in **LA.** We have the usual axioms of Robinson's arithmetic in **LA** together with axioms defining div, rem, and cond, for elements of type index. The axioms for field elements are the usual field axioms, plus the extra axiom: $a = 0 \vee a = 1$, since in this paper we are interested in **LA** restricted to the two element field.

LA is closed under the usual Frege rules for propositional consequence, as well as two special rules. **Induction:** $\alpha(i) \supset \alpha(i+1) \vdash \alpha(0) \supset \alpha(n)$, note that i must be an index variable which does not occur free on the right-hand side of the rule. **Equality:** $r(A) = r(B), c(A) = c(B), e(A,i,j) = e(B,i,j) \vdash A = B$, where i, j are index variables that do not occur free on the right-hand side of the rule.

The theorems of **LA** translate into families of propositional tautologies with $\mathbf{AC}^0[2]$-Frege proofs ([5]). However, in this paper we use a (conservative) extension of **LA** which has bounded index quantifiers. Fortunately, it turns out that the translation result still holds for the extended **LA.** We prove this in the next lemma, which will be used in the proof of corollary 2.

Lemma 1. *The theorems of* **LA***-with-bounded-index-quantifiers, and over the field of two elements, translate into families of tautologies with* $\mathbf{AC}^0[2]$*-Frege proofs.*

Proof. Let σ assign values to the index parameters of a formula, and let $|\sigma|$ be the largest value in the assignment σ. Let $\|\alpha\|_\sigma$ be the translation of α into a family of propositional tautologies, parametrized by σ.

We know from [5], that if α is a formula over the language of **LA,** then, there exists a polynomial p_α and a constant d_α such that for every σ, the size of $\|\alpha\|_\sigma$ is bounded by $p_\alpha(|\sigma|)$, and the depth of $\|\alpha\|_\sigma$ is bounded by d_α. If α

is a true formula (in the standard model) then, the propositional formula $\|\alpha\|_\sigma$ is a tautology. Furthermore, if α is a theorem of **LA**-without-index-quantifiers, then, there exists a polynomial q_α and a positive integer d_α such that for every σ, $\|\alpha\|_\sigma$ has an $\mathbf{AC}^0[2]$-Frege derivation $\pi_{\alpha,\sigma}$ such that the size of $\pi_{\alpha,\sigma}$ is bounded by $q_\alpha(|\sigma|)$ and the depth of $\pi_{\alpha,\sigma}$ is bounded by the constant d_α.

Now consider **LA** formulas with bounded index quantifiers. We translate quantifiers in the obvious manner:

$$\|(\exists i \le n)\alpha\|_\sigma \longmapsto \bigvee_{1\le j\le \|n\|} \|\alpha\|_{\sigma(i/j)} \qquad \|(\forall i \le n)\alpha\|_\sigma \longmapsto \bigwedge_{1\le j\le \|n\|} \|\alpha\|_{\sigma(i/j)}$$

where $\sigma(i/j)$ is σ with i replaced by j. As in any **LA** proof the number of quantifiers is bounded (and hence in particular the number of *alternations* of quantifiers is bounded), we still have a bounded depth d_α.

Furthermore, $(Q_1 i_1 \le n_1)(Q_2 i_2 \le n_2)\ldots(Q_k i_k \le n_k)\alpha$, where $Q_i \in \{\forall, \exists\}$ are alternating quantifiers, translates into a formula of size

$$O(\|n_1\|_\sigma \cdot \|n_2\|_\sigma \cdot \ldots \cdot \|n_k\|_\sigma \cdot \mathrm{size}(\|\alpha\|_\sigma)) \tag{1}$$

where in any **LA** proof, the k is bounded by a constant, and so (1) is bounded by some polynomial in $|\sigma|$.

The reason why we want bounded index quantification is that it allows us to state that a given matrix P is a permutation matrix:

$$[r(P) = c(P)] \wedge [(\forall i \le r(P))(\exists! j \le c(P))e(P,i,j) = 1] \wedge [PP^t = I] \tag{2}$$

(as we are dealing with a field of two elements, if $e(P,i,j) \neq 1$, it follows that $e(P,i,j) = 0$). Let (2) be abbreviated by Perm(P). Then, $(\exists P \le n)\alpha$ abbreviates $(\exists P)[r(P) \le n \wedge c(P) \le n \wedge \mathrm{Perm}(P) \wedge \alpha]$. Similarly, $(\forall P \le n)\alpha$ abbreviates the same formula but with the last "$\wedge$" replaced by "$\supset$."

Definition 1. *Let* $\exists P\mathbf{LA}$ *denote the theory* **LA** *with bounded* existential *permutation quantification; in particular,* $\exists P\mathbf{LA}$ *allows induction over formulas of the form* $(\exists P \le n)\alpha$. *Let* $\forall P\mathbf{LA}$ *be an analogous theory, but with bounded* universal *permutation quantification instead.*

Definition 2. *Let* **LAP** *be the theory* **LA** *with the matrix powering function* **P,** *which is defined by the axioms:* $\mathbf{P}(0, A) = I$ *and* $\mathbf{P}(n+1, A) = \mathbf{P}(n, A) * A$. *Let* $\exists P\mathbf{LAP}$ *and* $\forall P\mathbf{LAP}$ *be the extensions of* **LAP** *that allow bounded existential, respectively universal, permutation quantification.*

3 Matrix Form of the Pigeonhole Principle

The functional form of the **Pigeonhole Principle (PHP)** states that an injective function from a finite set into itself must necessarily be surjective. Over the

field GF(2), there are 2^{n^2} matrices of size $n \times n$, and so the **Matrix form of the Pigeonhole Principle (MPHP)** states that any injective function from the set of $n \times n$ matrices (over a fixed finite field) into itself must be surjective.

The constructed terms of **LA,** i.e., terms of the form $\lambda ij\langle n, n, t\rangle$, define functions from matrices to matrices in a very natural way: $A \longmapsto \lambda ij\langle n, n, t(A)\rangle$ is a function from the set of all matrices into the set of $n \times n$ matrices. If we restrict A to be an $n \times n$ matrix, we obtain a function from the set of $n \times n$ matrices into itself. This observation can be used to define the MPHP in **LA,** with bounded matrix quantification. We can state that the above mapping is injective as follows:

$$(\forall X_1 \leq n)(\forall X_2 \leq n)[\lambda ij\langle n, n, t(X_1)\rangle = \lambda ij\langle n, n, t(X_2)\rangle \supset X_1 = X_2] \quad (3)$$

and we can state that it is surjective with:

$$(\forall Y \leq n)(\exists X \leq n)[\lambda ij\langle n, n, t(X)\rangle = Y] \quad (4)$$

Notice that we could have stated the above more generally for $n \times m$ matrices, but the resulting formulas would be less readable, as we would have to state $(\forall X_1)[r(X_1) \leq n \wedge c(X_1) \leq m]$, instead of the handy $(\forall X_1 \leq n)$. In any case, square matrices are sufficient for what we want, and rectangular matrices can be padded to become square. We define MPHP to be the scheme of formulas (3) $\supset$ (4) for all n, t. We let $\mathbf{LA}^{\mathrm{MPHP}}$ be **LA** with the MPHP scheme.

Note that despite the fact that we employed bounded matrix quantification to express MPHP in **LA,** the theory $\mathbf{LA}^{\mathrm{MPHP}}$ is still allowed to have induction over formulas *without* quantifiers only.

An important reason why **LA** was designed in the first place was to study the proof theoretic complexity of the derivations of **hard matrix identities.** These are universal matrix identities, stated without quantifiers but implicitly universally quantified, that seem to require reasoning about inverses to prove them. The canonical example is IP_n, which can be stated in **LA** as follows:

$$\lambda ij\langle n, n, \Sigma\lambda kl\langle 1, n, A_{il}B_{lj}\rangle\rangle = I_n \supset \lambda ij\langle n, n, \Sigma\lambda kl\langle 1, n, B_{il}A_{lj}\rangle\rangle = I_n \quad (5)$$

where I_n is given by $\lambda ij\langle n, n, \mathrm{cond}(i = j, 1, 0)\rangle$.

It turns out that there are a host of matrix identities, that can be derived with "basic" properties from the IP_n, such as $AB = I \wedge AC = I \supset B = C$ or $AB = I \supset (AC = 0 \supset C = 0)$ (see [5] for more examples). All these identities are equivalent to IP_n in **LA** (hence they can be shown equivalent with basic ring properties). Let $\mathbf{LA}^{\mathrm{ID}}$ be **LA** extended by some matrix identity ID (formally, ID is any **LA**-formula). We say that ID is a hard matrix identity if $\mathbf{LA}^{\mathrm{IP}_n} = \mathbf{LA}^{\mathrm{ID}}$.

We can prove hard matrix identities in **LA** if at least one matrix is symmetric (next lemma). It remains an open question whether **LA** can prove hard matrix identities for general matrices, but we conjecture that it cannot. On the other hand, **LAP** can prove hard matrix identities for triangular matrices, since **LAP** proves the CHT for such matrices.

Lemma 2. **LA** *proves hard matrix identities for symmetric matrices.*

Proof. If at least one of A, B is symmetric ($A = A^t$ or $B = B^t$), and $AB = I$, then $(AB)^t = I^t = I$. And $(AB)^t = B^tA^t$. Suppose A is the symmetric one, then $B^tA = I$. Since $AB = I$ implies in **LA** that $A(BA - I) = 0$, it follows that $BA - I = 0$, so $BA = I$. Similar argument if B is the symmetric matrix.

In [6] we showed that IP_n does not have a bounded depth Frege proof, since we can derive from IP_n (in bounded depth Frege) the functional form of the PHP, which does not have a bounded depth Frege proof. Here we show a weak converse of that result; **LA** with the matrix form of the pigeonhole principle can prove IP_n(overthe field of two elements, and over any finite field).

Lemma 3. $\mathbf{LA}^{\mathrm{MPHP}}$ *proves hard matrix identities.*

Proof. Suppose that we want to prove $AB = I \supset BA = I$. Given $AB = I$, let $f_A(X) := XA$. The function f_A can be defined in **LA** with a constructed term. If $XA = YA$, then $(XA)B = (YA)B$, so by associativity $X(AB) = Y(AB)$, so $X = Y$. Hence f_A is 1-1. By the PHP, $(\exists X)f_A(X) = I$, so $XA = I$. This gives us a left-inverse for A. Since $AB = I$ implies (in **LA**) that $A(BA - I) = 0$, it follows from this that $BA - I = 0$, so $BA = I$. Since all the hard matrix identities can be shown equivalent in **LA** (by definition), we have the result.

4 The Cayley-Hamilton Theorem

We show that the CHT can be proven in the theory $\exists P\mathbf{LAP}$. In fact, $\forall P\mathbf{LAP}$ also proves the CHT, as the two theories prove the same theorems in the language of **LAP**. Many other universal properties of matrices follow from the CHT within **LAP** (see [4, Chapter 5]), so we have their proofs in $\exists P\mathbf{LAP}$ as well.

The characteristic polynomial of a matrix A can be given as a term p_A in the language of **LAP,** using Berkowitz's algorithm (see [4, Chapter 4]). Let $p_A(A)$ be the **LAP**-term expressing the result of plugging A into its characteristic polynomial. The CHT states that $p_A(A) = 0$.

If A is a square matrix, define $A[n]$ to be the n-th principal submatrix of A; that is, $A[1]$ is A with the first row and column removed, $A[2]$ is A with the first two rows and columns removed, and so on until $A[r(A)-1]$ which is just the 1×1 matrix consisting of the bottom-right corner entry of A (here $r(A) = c(A) =$ rows and columns of A). Formally in **LAP**,

$$A[n] =_{\mathrm{def}} \lambda kl\langle r(A) - n, c(A) - n, e(A, n + k, n + l)\rangle.$$

Note that $A[0] = A$.

Let $CH(A, n)$ be a **LAP** formula stating that the CHT holds for the matrices

$$A[n], A[n + 1], \ldots, A[r(A) - 1].$$

Formally, $CH(A, n)$ is given by

$$(\forall n \leq i < r(A))p_{A[i]}(A[i]) = 0$$

Note that the ∀-index quantifier could be replaced with a λ-construction, but we assume that we have bounded index quantifiers.

Lemma 4. $\exists P\mathbf{LAP}$ *proves the following:*

$$\neg CH(A,n) \supset (\exists P \leq r(A))\neg CH(PAP^t, n+1). \tag{6}$$

Proof. If $\neg CH(A,n)$, then there exists a $k \in \{n, n+1, \ldots, r(A)-1\}$ such that

$$p_{A[k]}(A[k]) \neq 0.$$

We choose the *largest* such k, and consider two cases.

Case 1 If $k \neq n$, then $k \geq n+1$, so let $P = I$, and clearly $\neg CH(A^\sigma, n+1)$ holds.

Case 2 If $k = n$, then by definition of k,

$$p_{A[n+1]}(A[n+1]) = \ldots = p_{A[r(A)-1]}(A[r(A)-1]) = 0 \tag{7}$$

We now find the *first* non-zero column of $p_{A[n]}(A[n])$, and call it j. Note that $j \neq 1$ since $p_{A[n+1]}(A[n+1]) = 0$, and we know by [4, lemma 8.2.1] that in that case the first column of $p_{A[n]}(A[n])$ must be zero. Thus $1 < j \leq r(A)-n$. Let I_k be the matrix obtained from the identity matrix by permuting rows k and $k+1$. I_k can be easily expressed with a λ-construction. We now run the program given in Figure 1 for finding a permutation P and an integer $0 \leq i < n$ such that $p_{(PAP^t)[n+j-i]}((PAP^t)[n+j-i]) \neq 0$.

The program clearly terminates (in at most $j \leq r(A)$ steps). It must output a correct P before i reaches the value $j-1$, since otherwise it would follow that

$$p_{(PAP^t)[n+1]}((PAP^t)[n+1]) = 0 \text{ with } P = I_n I_{n+1} \cdots I_{n+j-1}.$$

This is not possible, since it means that column j of A is in position n of PAP^t, and

$$p_{(PAP^t)[n+1]}((PAP^t)[n+1]) = 0$$

so again by [4, lemma 8.2.1] it would follow that the j-th column is zero. This contradicts the original assumption about the j-th column of A.

Note that the program is a search over finitely many matrices, using iterated matrix products. Thus, it can be formalized in **LAP.** Since $j > 1$ and $i \geq 0$,

$$p_{(PAP^t)[n+j-i]}((PAP^t)[n+j-i]) \neq 0$$

implies $\neg CH(PAP^t, n+1)$.

This ends the two cases and the proof of (6).

Theorem 1. $\exists P\mathbf{LAP}$ *proves the CHT, i.e.,* $\exists P\mathbf{LAP} \vdash p_A(A) = 0$.

$P \leftarrow I$
$i \leftarrow 0$
while $i < j$
  if $p_{(PAP^t)[n+j-i]}((PAP^t)[n+j-i]) = 0$ **then**
    $P \leftarrow I_{n+j-i-1}P$
    $i \leftarrow i+1$
  else
    output P
    break

Fig. 1. Program for computing the permutation P.

Proof. From (6) we can easily obtain:

$$(\exists P \leq r(A))\neg CH(PAP^t, n) \supset (\exists P \leq r(A))\neg CH(PAP^t, n+1) \qquad (8)$$

So now suppose that the CHT theorem fails for some matrix A, so $p_A(A) \neq 0$. Then $\neg CH(A, 0)$, so certainly $(\exists P \leq r(A))\neg CH(PAP^t, 0)$, where we can take $P = I$. This is our basis case, and (6) is our induction step, so we can conclude by the induction rule that $\neg CH(A, r(A) - 1)$. But that means that the CHT fails for a 1×1 matrix. It is easy to show in **LAP** that the CHT holds for 1×1 matrices, and so we obtain a contradiction.

The above theorem is also provable with the following induction hypothesis:

$$(\forall P \leq r(A))\neg CH(PAP^t, n+1) \supset (\forall P \leq r(A))\neg CH(PAP^t, n)$$

and so it follows, as was stated in the first paragraph of this section, that $\forall P\mathbf{LAP}$ proves the CHT as well.

Since $\exists P\mathbf{LAP}$ proves the CHT, it follows (by [5, theorem 4.1]) that $\exists P\mathbf{LAP}$ ($\forall P\mathbf{LAP}$) also proves hard matrix identities, and, by further results in [5], the multiplicativity of the determinant.

Corollary 1. $\exists P\mathbf{LAP}$ *proves hard matrix identities and the multiplicativity of the determinant.*

5 Expressing Graph-Theoretic Properties

In this section we show that the theories $\exists P\mathbf{LA}$ and $\forall P\mathbf{LA}$ are very well suited for expressing graph-theoretic properties. In the next section we show that $\forall P\mathbf{LA}$ can actually prove the soundness of the Hajós Calculus. Not surprisingly, $\exists P\mathbf{LA}$ can express **NP** graph problems, and $\forall P\mathbf{LA}$ can express co-**NP** graph problems.

Recall that **Graph Isomorphism (GI)** is the decision problem of whether two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, on the same set of nodes V, are isomorphic. That is, whether there is a permutation (i.e., *re-labeling*) π of the nodes V such that $G_2 = \pi(G_1)$, where $\pi(G_1) = (V, \{(\pi(u), \pi(v))|(u, v) \in E_1\})$.

GI is one of the few examples of decision problems that are in **NP** and not believed to be in **P** or **NP**-complete.

We can express GI succinctly in $\exists P\mathbf{LA}$ as follows:

$$(\exists P \leq r(A))[A = PBP^t]$$

here A and B are the **adjacency** matrices for graphs G_1 and G_2 (recall that A is the adjacency matrix for $G = (V, E)$ if $r(A) = c(A) = |V|$ and $e(A, i, j) = 1$ iff $(i, j) \in E$). Note that the (i, j)-**th** entry of PBP^t, $(PBP^t)_{ij}$, is given by $\sum_{1\leq k,l\leq n} P_{ik}B_{kl}P^t_{lj} = \sum_{1\leq k,l\leq n} P_{ik}B_{kl}P_{jl}$ (assuming that A, B, P are $n \times n$ matrices). Note that $P^t_{lj} = P_{jl}$ since for permutation matrices $P^t = P^{-1}$. Since P is a permutation matrix, it can be regarded as a function $P : [n] \longrightarrow [n]$ where $P(i) = j$ iff $P_{ij} = 1$. Hence, $(PBP^t)_{ij} = B_{P(i)P(j)}$.

We can also express the decision problem **Path** in $\exists P\mathbf{LA}$. Path on input (G, s, t, k) decides if there is a path in G from node s to node t of length k. If there is such a path, then there is a sequence of nodes $s = i_1, i_2, \ldots, i_k = t$ such that $(i_j, i_{j+1}) \in E$ for all j. Given $i_1, i_2, \ldots, i_k$, there is a re-labeling π of the nodes so that in $\pi(G)$ we have $\pi(s) = 1, 2, \ldots, k = \pi(t)$, and $(i, i+1)$ is an edge in $\pi(G)$. Thus, Path can be expressed in $\exists P\mathbf{LA}$ as follows:

$$(\exists P \leq r(A))[(\forall 0 < i < k)e(PAP^t, i, i+1) = 1 \wedge Ps = e_1 \wedge Pt = e_k]$$

The formula $(\forall 0 < i < k)e(PAP^t, i, i+1) = 1$ in the above expression is stating that the upper-left $k \times k$ corner of PAP^t has 1s on the diagonal above the main diagonal.

Hamiltonian Path (HP) can be stated as:

$$(\exists P \leq r(A))(\forall 0 < i < r(A))[e(PAP^t, i, i+1) = 1]$$

The idea is that we have 1s above the main diagonal, so that for $1 \leq i \leq n-2$ there is an edge $(i, i+1)$ in the re-labeled graph.

For example, in the undirected graph G given in Fig. 2, if we re-label the nodes according to the permutation P: $1 \mapsto 1, 2 \mapsto 5, 3 \mapsto 4, 4 \mapsto 3, 5 \mapsto 2$, we obtain the graph G' on the right with a HP 1-2-3-4-5 indicated by the arrows.

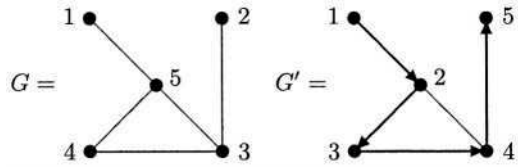


Fig. 2. Graph G and its re-labeling G'.

We can express the k-**Colorability** of graphs in $\exists P\mathbf{LA}$. Let 0_k denote the $k \times k$ matrix of zeros. Let G be a graph, and A_G its corresponding adjacency matrix. We can state that G is k-colorable, for any fixed k, as follows:

$$(\exists P \leq r(A_G))(\exists i_1, i_2, \ldots, i_k \leq r(A_G))[PA_GP^t = \left[\begin{array}{c|c|c|c} 0_{i_1} & * & \cdots & * \\ \hline * & 0_{i_2} & \cdots & * \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline * & * & * & 0_{i_k} \end{array}\right]]$$

The unspecified entries in the above graph (i.e., the entries in the blocks labeled by "$*$") can be anything. For $k = 3$, let **Non-3-Col**(A) be the negation of the above formula, stating that the graph whose adjacency matrix is A is *not* 3 colorable. Note that **Non-3-Col**(A) is a formula in the language of $\forall P\mathbf{LA}$.

Hamiltonian Cycle, Vertex Cover and **Clique** can also be stated using similar techniques.

6 The Hajós Calculus

In this section we will show that the theory $\forall P\mathbf{LA}$ proves the soundness of the Hajós calculus. The Hajós calculus is a very simple non-deterministic procedure for building non-3-colorable graphs. It can also be used as a propositional refutation system, and as such it is *p*-equivalent to extended Frege—see [3].

The Hajós calculus has the 4-clique as its only axiom: let K_4 denote the 4-clique (a complete graph on 4 vertices). $\forall P\mathbf{LA}$ can show that K_4 is not 3-colorable, that is $\forall P\mathbf{LA} \vdash$ **Non-3-Col**(A_{K_4}). Furthermore, the Hajós calculus has the following three rules for building bigger non-3-colorable graphs:

1. **Addition Rule:** Add any number of vertices and/or edges.
2. **Join Rule:** Let G_1 and G_2 be two graphs with disjoint sets of vertices. Let (i_1, j_1) and (i_2, j_2) be edges in G_1 and G_2, respectively. Construct G_3 as follows: remove edges (i_1, j_1) and (i_2, j_2), and add the edge (j_1, j_2), and contract vertices i_1 and i_2 into the single vertex i_1. See Fig. 3 for an example.

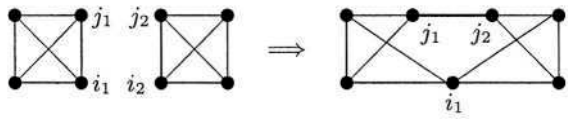


Fig. 3. The join rule applied to two K_4 graphs.

3. **Contraction Rule:** Contract two nonadjacent vertices into a single vertex, and remove the resulting duplicated edges. The new vertex can be either of the two original vertices.

A **derivation** in the Hajós calculus is a sequence of graphs $\{G_1, G_2, \ldots, G_n\}$ such that each G_i is either K_4, or follows from previous G_j's by one of the three rules. G_n is the graph being derived, i.e., the conclusion. The Hajós calculus is both **complete** (any non-3-colorable graph can be derived in it), and **sound**

(only non-3-colorable graphs can be derived). See [3] for proofs of completeness and soundness.

Lemma 5. $\forall P\mathbf{LA}$ *proves the soundness of the rules of the Hajós Calculus.*

Before giving the proof, note that a formula stating the completeness of the Hajós calculus would have to be a $\forall X \exists Y$-formula (for *any* A, if **Non-3-Col**(A), then *there exists* a derivation (i.e., there exists a long matrix encoding a derivation) of A). Thus, completeness cannot be expressed in $\forall\mathbf{LA}$. (Furthermore, a matrix encoding a derivation is not going to be a permutation matrix in general.) We conjecture that the stronger theory $\exists\mathbf{LA}$ can prove completeness.

Proof (of lemma 5). For the addition rule, let G' be G with new vertices/edges. This can be stated as follows:

$$r(A_G) \leq r(A_{G'}) \wedge (\forall i,j \leq r(A_G))[e(i,j,A_G) = 1 \supset e(i,j,A_{G'}) = 1]$$

So, $A_{G'}$ contains A_G in its upper-left corner, with, possibly, certain 0s replaced by 1s, and so it is easy to derive the sequent $\mathbf{Non\text{-}3\text{-}Col}(A_G) \rightarrow \mathbf{Non\text{-}3\text{-}Col}(A_{G'})$.

For the join rule, let G_1 and G_2 be the two graphs as in the statement of the rule, and A_{G_1} and A_{G_2} the corresponding adjacency matrices. Suppose that $e(A_{G_1}, i_1, j_1) = e(A_{G_2}, i_2, j_2) = 1$. Then A_G is given by a constructed matrix with $r(A_{G_1}) + r(A_{G_2}) - 1$ rows (and columns), and of the form:

$$\left[\begin{array}{cc|c} A_{G_1}[i_1|i_1] & & D_1 \\ & A_{G_2}[i_2|i_2] & D_2 \\ \hline D_1^t & D_2^t & 0 \end{array}\right] \tag{9}$$

where $A[i|j]$ is standard notation for a matrix with row i and column j removed, and D_1 is a column vector with a 1 in position j iff $e(A_{G_1}, i_1, j) = 1$, and D_2 is a column vector with a 1 in position j iff $e(A_{G_2}, i_2, j) = 1$. Matrix (9) can be given as a constructed matrix over **LA.** It is not difficult to derive the sequent:

$$\mathbf{Non\text{-}3\text{-}Col}(A_{G_1}) \wedge \mathbf{Non\text{-}3\text{-}Col}(A_{G_2}) \rightarrow \mathbf{Non\text{-}3\text{-}Col}(A_G)$$

The soundness of the contraction rule can be shown in a similar way.

There are two versions of the Hajós calculus: with labeled and un-labeled graphs. The two versions are *p*-equivalent. In fact, this can be shown in $\forall P\mathbf{LA}$, as we can derive $(\forall Q \leq r(A))\mathbf{Non\text{-}3\text{-}Col}(QAQ^t)$ from **Non-3-Col**(A) (this derivation is very easy, practically by definition of **Non-3-Col**).

Theorem 2. $\forall P\mathbf{LA}$ *proves the soundness of the Hajós Calculus.*

Proof. A derivation in the Hajós Calculus is given by a sequence of graphs $\{G_1, G_2, \ldots, G_n\}$, where G_n is the conclusion, and each G_i is either K_4 or follows from previous G_j's by the application of one of the three rules. We can encode the derivation as a long matrix $[A_{G_1} A_{G_2} \ldots A_{G_n}]$. Using induction on n, lemma 5, and the observation that $\forall P\mathbf{LA} \vdash \mathbf{Non\text{-}3\text{-}Col}(A_{K_4})$, we can show $\mathbf{Non\text{-}3\text{-}Col}(A_{G_n})$.

7 Quantified Permutation Frege

Permutation Frege is a well studied propositional proof system where, besides the usual Frege rules for propositional consequence, we have a restricted substitution rule $\alpha \vdash \alpha^\pi$ which allows us to permute the variables of α (according to π) to obtain α^π. As the permutation rule is a kind of restricted substitution, we know that Permutation Frege can be *p*-simulated by extended Frege. It remains an interesting open problem whether Permutation Frege and extended Frege are in fact *p*-equivalent, or whether Permutation Frege is strictly weaker.

We introduce a novel propositional proof system, which we call **Quantified Permutation Frege (QPF).** As the name suggests, QPF allows quantification over permutations, just as Quantified Frege ([2, §4.6]) allows quantification over variables.

In this paper we shall consider two fragments of QPF, namely $\exists\sigma$-Frege and $\forall\sigma$-Frege. In $\exists\sigma$-Frege we allow propositional formulas plus formulas of the form $\exists\sigma_S\alpha$, where α is a propositional formula *without* any quantifiers. Here σ_S denotes an automorphism (permutation) of the variables in the finite set S, and $\sigma_S|_{S^c} = \mathrm{id}$. Similarly, $\forall\sigma$-Frege consists of propositional formulas plus formulas of the form $\forall\sigma_S\alpha$. Note that the restriction on quantification is *strict* in the sense that we allow one quantifier in prenex form only. (Since S can consist of any finite number of variables, we can always express a block of existential (universal) permutation quantifiers with one existential (universal) permutation quantifier.)

The semantic of $\exists\sigma_S\alpha$ is as follows: given a truth value assignment t, $t \models \exists\sigma_S\alpha$ iff there exists a permutation of the variables in S such that $t^{\sigma_S} \models \alpha$, where $t^{\sigma_S}(x) = t(\sigma_S(x))$. The semantic of $\forall\sigma_S\alpha$ is defined analogously.

The rules of $\exists\sigma$-Frege and $\forall\sigma$-Frege are the usual Frege rules plus the following four sequent rules for introducing permutation quantifiers:

$$\frac{\Gamma \to \Delta, \alpha}{\Gamma \to \Delta, \exists\sigma_S\alpha^{\pi_S}} \qquad \frac{\alpha, \Gamma \to \Delta}{\exists\sigma_S\alpha^{\pi_S}, \Gamma \to \Delta} \qquad \frac{\Gamma \to \Delta, \alpha}{\Gamma \to \Delta, \forall\sigma_S\alpha^{\pi_S}} \qquad \frac{\alpha, \Gamma \to \Delta}{\forall\sigma_S\alpha^{\pi_S}, \Gamma \to \Delta}$$

where π_S is some permutation of the variables in S, and α^{π_S} is α with the variables permuted according to π_S. There are the following *restrictions:* α may not contain any (permutation) quantifiers, and for $\exists\sigma_S$ introduction left and $\forall\sigma_S$ introduction right, the variables in S are *not free* in the bottom sequent. The variables in a finite set S are **not free** in a given formula β if either they do not occur in β at all, or β is of the form $\exists\sigma_Q\gamma$ (or $\forall\sigma_Q\gamma$), with γ having no quantifiers, and $S \subseteq Q$.

It is easy to check that the four rules are sound. It is an open question whether, with the given definition of restriction, the system is complete (i.e., can we prove all true $\exists\sigma$ and $\forall\sigma$ sequents?). However, here we use $\exists\sigma$-Frege and $\forall\sigma$-Frege to prove propositional formulas without quantifiers, and for these formulas completeness follows from the completeness of Frege. The permutation quantifiers allow us to (apparently) shorten the proofs considerably.

We leave it as future research the definition of a general QPF system, where we allow arbitrary alternations of quantifiers, with a restriction that renders it

complete. We conjecture that a well defined QPF system should be equivalent to the general Quantified Frege (called G in [2, §4.6]).

Define $\exists\sigma$-Frege* and $\forall\sigma$-Frege* to be the same as $\exists\sigma$-Frege and $\forall\sigma$-Frege, but with the additional requirement that the proofs have to be tree-like (i.e., each sequent in the proof occurs at most once). Theorem 2 allows us to prove the following interesting corollary.

Corollary 2. *$\exists\sigma$-Frege* and $\forall\sigma$-Frege* are p-equivalent to extended Frege.*

Proof. $\exists\sigma$-Frege* and $\forall\sigma$-Frege* can be simulated by G_1^* which is p-equivalent to extended Frege ([2, section 4.6]). Conversely, extended Frege and the Hajós calculus are p-equivalent ([3]), and by theorem 2, $\forall P\mathbf{LA}$ proves the soundness of the Hajós calculus. On the other hand, the proof of theorem 2 can be translated into $\forall\sigma$-Frege* (and $\exists\sigma$-Frege*), as can be seen by noting that the theorems of **LA** translate into $\mathbf{AC}^0[2]$-Frege (by lemma 1), and by noting that universal permutation quantifiers occur in the form $(\forall P \leq n)\alpha(PAP^t)$, and so they can be easily translated into $\forall\sigma_A\|\alpha(A)\|_{\sigma'}$. (Note that σ and σ' are different objects; one is permutation quantification, and the other a translation parameter.) It follows that $\forall\sigma$-Frege* (and $\exists\sigma$-Frege*) can simulate the Hajós calculus, and hence extended Frege.

8 Open Problems

Is there an **LAP** proof of the CHT? A related question is: can we prove hard matrix identities in **LAP**? Can we show that hard matrix identities are independent of **LA**? What would be a natural definition of QPF (one that ensures soundness and completeness)? Is (a good definition of) QPF p-equivalent to G?

Acknowledgments. The author would like to thank Toniann Pitassi and Alasdair Urquhart for fruitful discussions that led to this paper.

References

1. Stephen A. Cook and Michael Soltys. The proof complexity of linear algebra. In *Seventeenth Annual IEEE Symposium on Logic in Computer Science (LICS 2002),* 2002.
2. Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory.* Cambridge, 1995.
3. Toniann Pitassi and Alasdair Urquhart. The complexity of the Hajós calculus. *SIAM J. Disc. Math.,* 8(3):464–483, August 1995.
4. Michael Soltys. *The Complexity of Derivations of Matrix Identities.* PhD thesis, University of Toronto, 2001.
5. Michael Soltys and Stephen Cook. The complexity of derivations of matrix identities. To appear in the Annals of Pure and Applied Logic, 2004.
6. Michael Soltys and Alasdair Urquhart. Matrix identities and the pigeonhole principle. *Archive for Mathematical Logic,* 43(3):351–357, April 2004.

A Calibration of Ineffective Theorems of Analysis in a Hierarchy of Semi-classical Logical Principles (Extended Abstract)

Michael Toftdal

Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark.
toftdal@daimi.au.dk

Abstract. We classify a number of nonconstructive mathematical theorems by means of a hierarchy of logical principles which are not included in intuitionistic logic. The main motivation is the development of the logical hierarchy by Akama Y. et al. and its connection to the so-called limit computable mathematics and proof animations developed by Hayashi S. et al. The results presented give insights in both the scope of limit computable mathematics and its subsystems, and the nature of the theorems of classical mathematics considered.

1 Introduction

This paper deals with the computational strength of a series of theorems of analysis. The work is motivated by [1] where a hierarchy of logical principles is developed. A weak part of this hierarchy has an interpretation in the so-called Limit Computable Mathematics (LCM); a semi-constructive theory in which a method for testing formal proofs and their formalisations is proposed in [4]. Roughly speaking, LCM considers not only the computable functions but also the limit computable functions — that is, the limits of convergent, computable sequences of computable functions (the gain of this stems from the fact that we do not require the rate of convergence to be computable).

To gain insight into LCM, it is useful to connect the logical principles from the aforementioned hierarchy, by which LCM can be formally described, to well-known ineffective theorems from classical mathematics.

This task resembles work from reverse mathematics (cf. [15]), in the sense that we shall aim at characterising what is needed (and sufficient) to prove certain theorems of classical mathematics. But where reverse mathematics focuses on which set theoretic axioms are needed and makes use of full classical logic, we are concerned with the amount of logic we need to add to intuitionistic logic, and we are more liberal in terms of set theoretic axioms — in particular the axiom of choice which in a sense is incorporated in the constructive idea behind intuitionistic logic (cf. the comments following Def. 3).

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1188–1200, 2004.

The project also borrows from Brouwerian counterexamples which is a technique used in various branches of constructive mathematics to show that certain statements are inherently nonconstructive. The idea is to show that if the principle in question were constructively valid, then also some classical and nonconstructive, logical principle would be derivable. That is, the mathematical principle considered entails nonconstructive logic and it therefore cannot be proved in constructive mathematics. However, we are also interested in precisely characterising mathematical principles, hence, Brouwerian counterexamples are, at best, only half of the calibration. The other half is in many cases established by formalising a standard proof from classical analysis.

2 A Constructive Setting

As the formal constructive theory, to deal with the questions introduced above, we use intuitionistic first-order arithmetic HA (Heyting arithmetic). It has the Peano axioms of arithmetic but is based on intuitionistic logic instead of classical logic — the corresponding system based on classical logic is termed PA. HA thus comes with primitive recursion, full induction and it is possible to do a substantial amount of arithmetic, including the definition of coding of finite sequences (and prove the elementary properties of the codings) and reducing any quantifier-free formula $A_0(x)$, with only x free, to a term t_A of HA such that $\forall x(t_A(x) = 0 \leftrightarrow A_0(x))$ is provable.

A detailed description of both intuitionistic logic and HA can be found in [18, 1.1–1.3].

We are going to deal with principles of analysis which involve properties of the real numbers, sequences of reals and functions mapping reals to reals. The rational numbers are constructed as pairs of natural numbers, and a real number is defined as a rational Cauchy sequence with fixed rate of convergence. So to reason about real numbers and functions, we shall use a neutral variant of intuitionistic arithmetic in all finite types, which we will denote HA^ω.[1] The system is neutral in the sense that it has both models with extensional and models with intensional equality.

We use superscripts to denote the type of an object; 0 is the type of (the natural) numbers and $1 = (0 \to 0)$ is the type of number theoretic functions. In general, $\sigma \to \tau$ is the type of functional mapping objects of type σ to objects of type τ. It is convenient to let any type 1 function represent a real number to avoid some implicative assumptions, and let quantification over reals be quantification over type 1 functions. This is obtained by a construction in [7, p. 48], and is to be implicitly understood.

We express that a real number x^1 lies in, for instance, the unit interval by $x \in_{\mathbb{R}} [0,1]$, which means that the Cauchy sequence represented by x satisfies $0_{\mathbb{R}} \leq_{\mathbb{R}} x \leq_{\mathbb{R}} 1_{\mathbb{R}}$. We shall also use $q \in_{\mathbb{Q}} [0,1]$ to say that q is a rational and between the rationals 0 and 1. The relation $<_{\mathbb{R}}$ is a Σ^0_1 statement (cf. Sect. 3.1)

[1] The system we use is from [18, 1.6.15].

and $\leq_{\mathbb{R}}$ is defined as $\not>$, and hence a Π^0_1 statement. Also $=_{\mathbb{R}}$ is a Π^0_1 statement. $0_{\mathbb{R}}$ is the constant Cauchy sequence of the rational representation of 0.

The only real-valued functions we are going to consider, will be uniformly continuous and defined on the unit interval; we denote this space of functions by $C[0,1]$. Functions in $C[0,1]$ are given to us with a modulus of uniform continuity, i.e. $f \in C[0,1]$ means that we have a function $f^{1\to 1}$ and a number theoretic function ω such that

$$\forall k^0 \forall x, y \in_{\mathbb{R}} [0,1](|x-y| < 2^{-\omega(k)} \to |f(x) - f(y)| < 2^{-k}) \ .$$

3 Logical and Mathematical Principles

We introduce the logical principles that are to be looked into later. They fall in three groups; restrictions of the law of the excluded middle $A \vee \neg A$ (LEM), the double negation elimination principle $\neg\neg A \to A$ (DNE) and a strengthening of the separation principle $\neg(A \wedge B) \to \neg A \vee \neg B$. It is well-known that adding either of the first two principles in their unrestricted form to intuitionistic logic results in a formulation of classical logic. But when proving this fact one might notice that a more complicated instance of DNE is needed to get a certain LEM instance. The actual content of this observation is in the hierarchy from [1] which we present in Sect. 3.2.

Next we formulate the relevant principles of classical analysis. The formalisation of the principles forces us to make explicit what we for instance mean when we say "least upper bound". Constructive reasoning distinguishes between the classically equivalent formulations as we shall see an example of in Sect. 4.

3.1 Logic and the Axiom of Choice

To define the logical principles to be examined we need the following.

Definition 1. *Let t be a term of* HA *representing a type* 1 *function and n a natural number. We define the following two formulas with alternating quantifiers*

(i) $\Sigma^0_n(t) :\equiv \exists k_1 \forall k_2 \ldots Qk_n(t(k_1, k_2, \ldots, k_n) = 0)$, *where Q is $\exists$ if n is odd, $\forall$ if n is even.*

(ii) $\Pi^0_n(t) :\equiv \forall k_1 \exists k_2 \ldots Qk_n(t(k_1, k_2, \ldots, k_n) = 0)$, *where Q is $\forall$ if n is odd, $\exists$ if n is even.*

Σ^0_n and Π^0_n refer to the classes of all $\Sigma^0_n(t)$ respectively $\Pi^0_n(t)$ formulas.

For instance, $\Sigma^0_1(f) \leftrightarrow \exists k(f(k) = 0)$ and $\Pi^0_2(f) \leftrightarrow \forall k \exists l(f(k,l) = 0)$.

By restricting well-known principles of classical logic to the above mentioned formula classes we obtain the following schemata.

Definition 2. *Let Γ be one of Σ and Π.*

(i) The Law of Excluded Middle principle for Γ^0_n formulas is given by:

$$\Gamma^0_n\text{-LEM}(t) :\equiv \Gamma^0_n(t) \vee \neg\Gamma^0_n(t) \ .$$

(ii) The Double Negation Elimination principle is given by

$$\Gamma^0_n\text{-}\mathsf{DNE}(t) :\equiv \neg\neg\Gamma^0_n(t) \to \Gamma^0_n(t) \ .$$

(iii) The Lesser Limited Principle of Omniscience is given by

$$\Sigma^0_n\text{-}\mathsf{LLPO}(t_1,t_2) :\equiv \neg(\Sigma^0_n(t_1) \wedge \Sigma^0_n(t_2)) \to \Pi^0_n(\overline{\mathrm{sg}}(t_1)) \vee \Pi^0_n(\overline{\mathrm{sg}}(t_2)) \ ,$$

Here $\overline{\mathrm{sg}}$ denotes the complement of the signum function sg, ie. $\mathrm{sg}(0) = 0$ and $\mathrm{sg}(x+1) = 1$, and $\overline{\mathrm{sg}}(0) = 1$ and $\overline{\mathrm{sg}}(x+1) = 0$.

Remark 1. (i) The more vivid name 'principle of omniscience' was introduced in [2] for the full LEM principle, and 'limited principle of omniscience' (LPO) for a principle stating:

> If $\{a_n\}$ is any decision sequence [i.e. a nondecreasing sequence of 0's and 1's], then either all $a_n = 0$ or some $a_n = 1$.

Furthermore the term 'weak limited principle of omniscience' (WLPO) is used in [10] for the principle:

> If $\{a_n\}$ is any decision sequence, then either all $a_n = 0$ or it is contradictory that some $a_n = 1$.

The restriction to decision sequences is inessential (which is seen using a bit of primitive recursion), and one therefore notices the resemblance to the schemata Σ^0_1-LEM and Π^0_1-LEM. The difference is explained below.

(ii) Π^0_{n+1}-DNE is equivalent to Σ^0_n-DNE in HA and Π^0_1-DNE is provable in HA since $\mathsf{HA} \vdash \neg\neg\forall k A(k) \to \forall k \neg\neg A(k)$, wherefore we shall only consider instances of Σ^0_n-DNE.

For historical reasons Σ^0_1-DNE is also known as Markov's principle.

An instance of a schema S is given by a term t in HA of appropriate type, possibly with number parameters. In this way $\forall x \mathsf{S}(t[x])$ is an instance of S. However, function parameters are not considered, since this would not allow a precise measurement of the strength of a semi-constructive interpretation like limit computable mathematics; e.g. Σ^0_1-LEM has a realizer among the limit computable functions, but if function parameters were allowed it would be possible to iterate $\forall f^1\, \Sigma^0_1\text{-}\mathsf{LEM}(f)$ and get Σ^0_n-LEM for any n in the presence of $\mathsf{AC}^{0,0}$.

This is the important difference between Σ^0_1-LEM and LPO, Π^0_1-LEM and WLPO; LPO and WLPO are considered with function parameters.

Definition 3. *We define* $\mathsf{AC}^{0,0}$ *(Axiom of Choice of type* 0,0 *) by*

$$\forall x^0 \exists y^0 A(x,y) \to \exists f^1 \forall x^0 A(x, f(x)) \ .$$

The full choice axiom is the $\mathsf{AC}^{0,0}$ *generalised to all finite types.*

The intuitive, constructive content of a proof of $\forall x \exists y$ is a procedure p that finds a witness $p(x)$ for the existential quantifier. Thus the axiom of choice is justified constructively. See for instance [18, 3.7] for formal results regarding intuitionistic theories and choice.

3.2 Semi-constructive Hierarchy

Theorem 1 ([1]). *The implications in Fig. 1 are derivable in* HA. *In the diagram* $\longrightarrow\!\!\!\!\!\rightarrow$ *denotes that the implication cannot be reversed, and* $\not\longrightarrow$ *refers to underivability.*

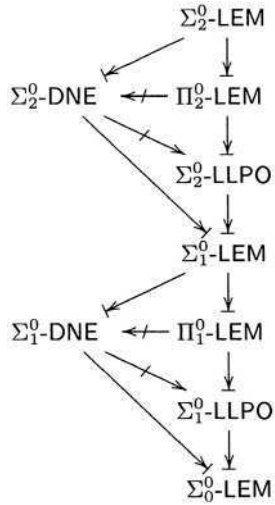


Fig. 1. Semi-constructive hierarchy of logical principles; cf. Theorem 1.

3.3 Analysis

As with the logical principles above, we allow number parameters in instances of the following schemata, but not function parameters. To remind ourselves and others of this, we add a $^-$ to the schema name when referring to this restricted form of it.

Definition 4. *We consider the principle of monotone convergence stating that a bounded sequence of rationals has a limit.*

$$\begin{aligned}\mathsf{PCM}(f) :\equiv &\forall n\,(0 \leq_{\mathbb{Q}} f(n+1) \leq_{\mathbb{Q}} f(n)) \\ &\to \exists g^1 \forall k \forall m \left(|f(g(k)) -_{\mathbb{Q}} f(g(k)+m)|_{\mathbb{Q}} <_{\mathbb{Q}} 2^{-k}\right) .\end{aligned}$$

Definition 5. *We define a weak supremum property for a function* $f^{0\to 1}$ *representing a sequence of real numbers in* [0,1].

$$\mathsf{WSup}(f) :\equiv \forall n(f(n) \in_{\mathbb{R}} [0,1]) \to \exists x^1 \forall y^1 (\forall n^0 (y \geq_{\mathbb{R}} f(n)) \leftrightarrow y \geq_{\mathbb{R}} x) .$$

The weakness of WSup (from a constructive viewpoint) lies in the fact that we do not require that numbers arbitrarily close to the least upper bound can be found.

Definition 6. *The principle of bounded variation considered here states that a uniformly continuous function on the unit interval, which is of bounded variation, is the difference of two increasing functions.*

$$\mathsf{PBV}(f) :\equiv f \in C[0,1] \to \big(\exists b \in \mathbb{Q} \forall \underline{r}^0 (V_{\underline{r}}(f) \leq_{\mathbb{R}} b) \to \exists g, h^{1\to 1} \forall x, y \in_{\mathbb{R}} [0,1]$$
$$[x \leq_{\mathbb{R}} y \to (g(x) \leq_{\mathbb{R}} g(y) \wedge h(x) \leq_{\mathbb{R}} h(y)) \wedge f(x) =_{\mathbb{R}} g(x) - h(x)]\big) \ ,$$

where

$$V_{r_1,\ldots,r_m}(f) :\equiv \sum_{i=0}^{m-1} |f(r_{i+1}) - f(r_i)| \, .$$

We consider two constructively different (but classically equivalent) definitions of the limit superior principle, corresponding to the two following formalisations of "x is lima sup of the sequence of real numbers represented by $f^{0\to 1}$". The first part of the conjunction expresses that x is a limit point (or cluster point). That x is the largest limit point can be expressed by saying that for every larger candidate there is a bound to how many points in the sequence will be close to it. Or negatively, any larger candidate is not a limit point.

Definition 7. *Positively:*

$$\forall k \left[\forall m \exists n >_0 m \left(|x -_{\mathbb{R}} f(n)|_{\mathbb{R}} \leq_{\mathbb{R}} 2^{-k}\right) \wedge \exists l \forall i > l \left(f(i) \leq_{\mathbb{R}} x + 2^{-k}\right)\right] \ . \quad (1)$$

Negatively:

$$\forall k \left[\forall m \exists n > m \left(|x - f(n)| \leq 2^{-k}\right) \wedge \neg \forall l \exists i > l \left(f(i) > x + 2^{-k}\right)\right] \ . \quad (2)$$

Now define $\mathsf{Limsup}_{\mathrm{pos}}(f)$ *for a sequence of rationals represented as* f^1 *by*

$$\forall n (f(n) \in_{\mathbb{Q}} [0,1]) \to \exists x \ s.t. \ (1) \ ,$$

and $\mathsf{Limsup}_{\mathrm{neg}}(f)$ *by*

$$\forall n (f(n) \in_{\mathbb{Q}} [0,1]) \to \exists x \ s.t. \ (2) \ .$$

When an operation or relation subscripted by $\mathbb{R}$ has a rational argument r (as above) it is implicit that r is to be replaced by its trivial $\mathbb{R}$ representation as a constant sequence, $\lambda n.r$.

Definition 8. *Let* $\mathsf{Max}(f)$ *denote the principle stating that if* f *is defined and uniformly continuous on* [0,1] *then it attains its maximum:*

$$\mathsf{Max}(f) :\equiv f \in C[0,1] \to \exists x \in_{\mathbb{R}} [0,1] (f(x) =_{\mathbb{R}} \sup_{y \in [0,1]} f(y)) \ .$$

Definition 9. *We define the intermediate value theorem as*

$$\mathsf{IVT}(f) :\equiv f \in C[0,1] \to ((f(0) \leq_{\mathbb{R}} 0 \wedge f(1) \geq_{\mathbb{R}} 0) \to \exists x \in_{\mathbb{R}} (f(x) =_{\mathbb{R}} 0)) \ .$$

There are two different, common formulations of the Bolzano-Weierstraß principle in the literature. One is that any bounded sequence of rationals has a convergent subsequence. The other states the existence of a limit point. In this treatment we shall look at the simpler, latter one.

Definition 10.

$$\mathsf{BW}(f) :\equiv \forall n(f(n) \in_{\mathbb{Q}} [0,1]) \to \exists x \in_{\mathbb{R}} [0,1] \forall k, m \exists n > m(|f(n) -_{\mathbb{R}} x|_{\mathbb{R}} \leq_{\mathbb{R}} 2^{-k}) \ .$$

We need only consider rational sequences, for given a sequence of reals x_n it is clear that a limit point of the sequence of n'th approximations $\mathrm{approx}(n, x_m)$ is a limit point of x_m; where $\mathrm{approx}(n, x_m)$ interprets x_m as a Cauchy sequence of rationals and gives the n'th rational approximation.

4 Results

This section presents the main results of [17]. Theorems are joined by a description of previous and related work.

4.1 Principle of Monotone Convergence

The first result we present states that PCM^- is equivalent to Σ^0_1-LEM. In [11] one finds the statement that PCM is equivalent to the principle LPO in Bishop's constructive mathematics (cf. [6] for a survey on Bishop-style reverse mathematics). There is, however, a substantial difference in that [11], as mentioned, considers all principles with function parameters, which allows for iteration of the principles. This iteration can cause climbing up in the hierarchy depicted above.

We do a more refined analysis in which LPO splits up in several principles of which the first could be seen as LPO^-, i.e. LPO without function parameters. In the notation used in this paper LPO^- is Σ^0_1-LEM. The area of limit computable mathematics shows this refinement to be not only a curious method to gain improved understanding of principles of mathematics, but a very natural one, in that it has practical meaning.

For classical reverse mathematics the same refinement was carried out and described in [9]. In reverse mathematics (as dealt with in [15]) it is shown that PCM is equivalent to full arithmetical comprehension, i.e. $\exists f^1 \forall x^0 (f(x) = 0 \leftrightarrow A(x))$, for any arithmetical formula A, over weak fragments of second-order arithmetic. The corresponding refined result is that PCM^- is equivalent to comprehension for Σ^0_1 formulas (without function parameters); cf. [9, 5.7].

In [19, 5.4.4] a Brouwerian counterexample to PCM is given, by showing it to imply LPO. When having a closer look at the proof, one finds that the construction given also shows PCM^- to imply Σ^0_1-LEM, which is the second half of Theorem 2.

For the first half it is natural to consider the classical standard proof of PCM^-. But since this is a proof by contradiction we constructively get a statement of the form $\forall\neg\neg\exists\forall$ (ignoring the implicative assumption for the sake of simplicity). Instead we would need $\forall\exists\forall$ to which we then could apply $\mathsf{AC}^{0,0}$ and get PCM^-. So it seems we need Σ^0_2-DNE to get the desired result. But since we know from the hierarchy above that Σ^0_2-DNE is strictly stronger than Σ^0_1-LEM, this is not satisfactory. However, by an essentially different induction proof that only uses Σ^0_1-LEM, one can get the first half of the following theorem.

Theorem 2. *(i)* $\mathsf{HA}^\omega + \mathsf{AC}^{0,0} + \Sigma^0_1\text{-LEM} \vdash \mathsf{PCM}^-$
(ii) $\mathsf{HA}^\omega + \mathsf{PCM}^- \vdash \Sigma^0_1\text{-LEM}$

4.2 Weak Supremum Property and Principle of Bounded Variation

Rather recently, functions of bounded variation have been studied in constructive mathematics; for instance in [3] and [13]. The last-mentioned paper gives a Brouwerian counterexample to PBV by showing it to imply WLPO, and the converse is shown to hold in the presence of $\mathsf{AC}^{0,0}$. The former has a refined counterpart in Theorem 3(i). The latter is established by showing WLPO to imply WSup (in the presence of $\mathsf{AC}^{0,0}$) and that WSup in turn implies PBV.

The first of these two implications and its reverse is refined in the last part of Theorem 3. The other direction from [13] has not yet been refined and it is not known whether a stronger principle than Π^0_1-LEM is actually needed to prove PBV^-.

Theorem 3. *(i)* $\mathsf{HA}^\omega + \mathsf{PBV}^- \vdash \Pi^0_1\text{-LEM}$
(ii) $\mathsf{HA}^\omega + \mathsf{AC}^{0,0} + \Pi^0_1\text{-LEM} \vdash \mathsf{WSup}^-$
(iii) $\mathsf{HA}^\omega + \mathsf{WSup}^- \vdash \Pi^0_1\text{-LEM}$

4.3 The Limit Superior

The limit superior principle has not been studied in constructive mathematics since its nonconstructive nature is obvious; e.g. it implies PCM^-. When formalising it, one notices that there are two formulations which occur in classical mathematics but in a constructive setting give rise to two different principles. The fact that the two are not equivalent follows from the next two theorems and the logical hierarchy presented above.

Theorem 4. *(i)* $\mathsf{HA}^\omega + \mathsf{AC}^{0,0} + \Pi^0_2\text{-LEM} \vdash \mathsf{Limsup}^-_{\mathrm{neg}}$
(ii) $\mathsf{HA}^\omega + \mathsf{Limsup}^-_{\mathrm{neg}} \vdash \Pi^0_2\text{-LEM}$

Theorem 5. *(i)* $\mathsf{HA}^\omega + \mathsf{AC}^{0,0} + \Sigma^0_2\text{-LEM} \vdash \mathsf{Limsup}^-_{\mathrm{pos}}$
(ii) $\mathsf{HA}^\omega + \mathsf{Limsup}^-_{\mathrm{pos}} \vdash \Sigma^0_2\text{-LEM}$

4.4 Attainment of the Maximum and the Intermediate Value Theorem

In [5] it is shown that in Bishop's constructive mathematics LLPO, the attainment of the maximum principle and weak Königs lemma are pairwise equivalent. The refined version of part of this is half of Theorem 6.

It is easily checked that the attainment of the maximum implies the intermediate value theorem. For consider an instance $\mathsf{IVT}^-(f)$ for $f \in C[0,1]$ with $f(0) \leq 0 \wedge f(1) \geq 0$. Then $g := -|f| \in C[0,1]$ too, and a point where g attains its maximum is a root of f. The other direction does not come as easy, which is demonstrated by a curious property found in recursive mathematics:

It is known that for a computable function f, $f(0) \leq_{\mathbb{R}} 0$ and $f(1) \geq_{\mathbb{R}} 0$, there exists (in a classical sense) a computable root. We sketch the proof. There are two cases: Either f has some rational root, or it does not. The rational numbers are computable, which closes the first case. In the second case we split the interval in two and evaluate f at the point of division c. This is a rational point and so we know that $f(c) \neq_{\mathbb{R}} 0$. By Markov's principle, which is accepted in recursive mathematics, $f(c) <_{\mathbb{R}} 0 \vee f(c) >_{\mathbb{R}} 0$. In either of these sub-cases we end in a situation like the initial. Thus we can define a Cauchy sequence converging towards a root of f. For full details see [12, 0.6.8] or [15, II.6.6].

This rather positive result does not have a counterpart for the Max^- principle. In [16] a computable function with no computable point of maximum is constructed. In spite of this mismatch between Max^- and IVT^-, the following theorem shows the two to be equivalent in our constructive setting.

Theorem 6. *(i)* $\mathsf{HA}^\omega + \mathsf{AC}^{0,0} + \Sigma^0_1\text{-LLPO} \vdash \mathsf{Max}^-$
(ii) $\mathsf{HA}^\omega + \mathsf{Max}^- \vdash \Sigma^0_1\text{-LLPO}$
(iii) $\mathsf{HA}^\omega + \mathsf{AC}^{0,0} + \Sigma^0_1\text{-LLPO} \vdash \mathsf{IVT}^-$
(iv) $\mathsf{HA}^\omega + \mathsf{IVT}^- \vdash \Sigma^0_1\text{-LLPO}$

As an immediate consequence of this we have the following.

Corollary 1. *(i)* $\mathsf{HA}^\omega + \mathsf{AC}^{0,0} \vdash \mathsf{IVT}^- \leftrightarrow \mathsf{Max}^-$
(ii) $\mathsf{HA}^\omega + \mathsf{QF\text{-}AC}^{0,0} \nvdash \mathsf{IVT}^- \rightarrow \mathsf{Max}^-$

Remark 2. For two schemata S_1 and S_2, $\mathsf{H} \vdash \mathsf{S}_1 \rightarrow \mathsf{S}_2$ means "for all terms $t_2[n]$ of HA there exists a term $t_1[m]$ of HA such that $\mathsf{H} \vdash \forall m \mathsf{S}_1(t_1[m]) \rightarrow \forall n \mathsf{S}_2(t_2[n])$".

One might think that the project of this paper, which could be seen as a first attempt to do a refined, intuitionistic reverse mathematics, only distinguishes more mathematical principles than classical reverse mathematics. The argument being, that intuitionistic logic is a restriction of classical logic, therefore proves fewer equivalences, and the refinement only contributes to this. From the example above we see that this is not the case; reverse mathematics distinguishes Max^- and IVT^-, but intuitionistically they should be identified.

The corollary pinpoints the reason to this. Identifying Max^- with IVT^- requires application of axiom of choice for arithmetical formulas, $\mathsf{AC}^{0,0}_{\mathrm{ar}}$. As $\mathsf{AC}^{0,0}_{\mathrm{ar}}$

only makes the interpretation of the logical connectives and quantifiers of intuitionistic logic explicit, $\mathsf{HA}^\omega + \mathsf{AC}^{0,0}_{\mathrm{ar}}$ is a reasonable constructive and robust system. Adding $\mathsf{AC}^{0,0}_{\mathrm{ar}}$ to a classical system (like PA^ω), on the other hand, has the unwanted consequence that both Max^- and IVT^- become provable (and therefore trivially equivalent) along with full arithmetical comprehension — which in turn gives much stronger theorems like BW and existence of lim sup.

4.5 Bolzano-Weierstraß

In [11] it was shown that PCM and BW are equivalent to LPO. The next theorem and the hierarchy in Sect. 3.2 shows this result not to hold in a refined sense of PCM^- and BW^-. Instead BW^- fits in nicely between $\mathsf{Limsup}^-_{\mathrm{neg}}$ and PCM^-.

Classical reverse mathematics proves BW to be equivalent to arithmetical comprehension ([15, III.2.2]) and the existence of lim sup.

In the refined sense, it is shown in [9] that in terms of provably total functions, BW^- corresponds exactly to $\mathsf{PRA} + \Sigma^0_1\text{-}\mathsf{IA}$ over a weak second order system — where PRA is PA with only quantifier-free induction and $\Sigma^0_1\text{-}\mathsf{IA}$ is induction restricted to Σ^0_1 formulas. On the other hand $\mathsf{Limsup}^-_{\mathrm{pos}}$ corresponds to $\mathsf{PRA} + \Sigma^0_2\text{-}\mathsf{IA}$, which, for instance, proves Ackermann's function to be total.

Theorem 7. *(i)* $\mathsf{HA}^\omega + \mathsf{AC}^{0,0} + \Sigma^0_2\text{-}\mathsf{LLPO} \vdash \mathsf{BW}^-$
(ii) $\mathsf{HA}^\omega + \mathsf{BW}^- \vdash \Sigma^0_2\text{-}\mathsf{LLPO}$

We give a proof of Theorem 7(ii) to illustrate how proofs of the preceeding results go. The proof of Theorem 7(i) is the standard bisection argument, observing that the statement "either the left or the right subinterval contains infinitely many points" is derivable from $\Sigma^0_2\text{-}\mathsf{LLPO}$ since "infinitely many" is a Π^0_2 property $(\forall m \exists n > m\, P(m,n))$.

Proof (of (ii)). Let the $\Sigma^0_2\text{-}\mathsf{LLPO}$ instance be given by two terms t_1 and t_2, such that the premise in $\Sigma^0_2\text{-}\mathsf{LLPO}$ is satisfied, i.e.

$$\neg\,(\exists a \forall b (t_1(a,b) = 0) \wedge \exists a \forall b (t_2(a,b) = 0)) \quad .$$

Define

$$a(n) := \max \left\{ a \leq n \;\middle|\; \begin{array}{r} a = 0 \vee \forall a' \leq a \exists b \leq n (t_1(a',b) \neq 0) \vee \\ \forall a' \leq a \exists b \leq n (t_2(a',b) \neq 0) \end{array} \right\}$$

and then

$$f(n) := \begin{cases} 1 + 2^{-a(n)} & \text{if } \forall a' \leq a(n) \exists b \leq n (t_1(a',b) \neq 0) \\ 2^{-a(n)} & \text{otherwise} \end{cases} \quad .$$

Let x be the limit point guaranteed by $\mathsf{BW}^-(f)$. We are going to prove $x \geq 0 \wedge x \leq 1 \wedge \neg(x > 0 \wedge x < 1)$. We then get $x = 0 \vee x = 1$. Finally we shall show

$$[x = 1 \to \forall a \exists b (t_1(a,b) \neq 0)] \wedge [x = 0 \to \forall a \exists b (t_2(a,b) \neq 0)] \quad .$$

But first assume $x > 1$. We then easily get a k_0 such that

$$\forall m \exists n > m(f(n) \geq 1 + 2^{-k_0}) \ .$$

Hence

$$\forall m \exists n > m(a(n) \leq k_0) \ ,$$

which by the monotonicity of $a(\cdot)$ gives

$$\forall n(a(n) \leq k_0) \ ;$$

that is,

$$\forall n(\exists a \leq k_0 + 1 \forall b \leq n(t_1(a,b) = 0) \wedge \exists a \leq k_0 + 1 \forall b \leq n(t_2(a,b) = 0)) \ . \quad (3)$$

Now assume $\neg\exists a \leq k_0 + 1 \forall b(t_1(a,b) = 0)$. It is clear that BW^- implies PCM^-, so by Theorem 2 also Σ^0_1-LEM, and then in turn we get Σ^0_1-DNE.[2] Therefore we get the following more positive statement

$$\forall x \leq k_0 + 1 \exists b(t_1(a,b) \neq 0) \ .$$

We can then bound b (proved by induction) and get a contradiction to (3). This way we get

$$\neg\neg\exists a \leq k_0 + 1 \forall b \leq n(t_1(a,b) = 0) \wedge \neg\neg\exists a \leq k_0 + 1 \forall b \leq n(t_2(a,b) = 0) \ .$$

By pulling out the double negations we get a contradiction to the premise of Σ^0_2-LLPO. The same way we can get a contradiction from $x > 0 \wedge x < 1$. Since $x \geq 0$ is trivial we now, by the preliminary arguments, have $x = 0 \vee x = 1$.

Assume $x = 1$. We then get $\forall k, m \exists n > m(|1 - f(n)| \leq 2^{-k})$, hence

$$\forall k \geq 2 \forall m \exists n > m(2^{-a(n)} \leq 2^{-k} \wedge \forall a' \leq a(n) \exists b \leq n(t_1(a',b) \neq 0)) \ ,$$

which implies $\forall a \exists b(t_1(a,b) \neq 0)$.

If the "otherwise" case in the definition of f occurs, we must have $\forall a' \leq a(n) \exists b \leq n(t_2(a',b) \neq 0)$ by definition of $a(n)$. Analogously we thus get $\forall a \exists b(t_2(a,b) \neq 0)$ if $x = 0$. □

5 Concluding Remarks

It is well-known that from a proof in HA of a statement $\forall x \exists y A(x,y)$ one can extract a recursive function f such that $\forall x A(x, f(x))$ (program extraction). It is

[2] Note that we do not need function parameters in BW^- to get this. The required (finite) sequence of Σ^0_1-DNE instances we use *and* the limit point x of f can all be derived from one number parameterised instance of BW^-. One applies BW^- to a term t' that has one more number parameter than f and a term t which must be found in the construction in the proof of Theorem 2(ii). The term t' is to be defined from f and t by cases. The curious reader can find t in [19, 5.4.4].

also known that this property in general does not even extend to $\mathsf{HA}+\Sigma^0_1\text{-}\mathsf{LLPO}$. However, the hierarchy does enjoy a generalised extraction property which is used in [1]: From proofs using $\Sigma^0_1\text{-}\mathsf{LEM}$ one can extract functions that are recursive in the jump $\emptyset'$ (cf. [14]) and using $\Sigma^0_2\text{-}\mathsf{LEM}$ one gets functions that are recursive in $\emptyset''$ etc. From the limit lemma of recursion theory it is known that limit recursion corresponds to recursion in $\emptyset'$, and the results of this paper therefore readily provide information on the effectiveness that is preserved by the examined mathematical principles in general, and the scope of LCM in particular.

A central topic of mathematical logic is the area called proof theory. Part of this deals with the extraction of computational information from ineffective theorems of mathematics. This information might give algorithms or computable bounds on existential quantifiers. E.g. a $\forall x \exists y A(x,y)$ statement for an arbitrary A proved using $\Pi^0_1\text{-}\mathsf{LEM}$ has a recursive bounding function f such that $\forall x \exists y \leq f(x) A(x,y)$; generalised results of this kind are in e.g. [8], which also gives similar results for $\mathsf{HA}+\Sigma^0_1\text{-}\mathsf{DNE}+\Sigma^0_1\text{-}\mathsf{LLPO}$. The results of this paper therefore provide simple and precise correspondences between the proof theoretic systems studied for purposes of proof theory and principles from mathematical practice.

But we shall also stress that our work provides a robust context for calibrating mathematical principles. We have seen that measuring the amount of classical logic incorporated in mathematical principles, causes distinctions also found by the refined classical calibration, as dealt with in [9]. But in this paper the distinctions appear to have a simpler formulation. We have also seen that the reverse mathematics of [15] and that of this paper are incomparable in the sense that neither of them simply distinguishes more than the other. Instead the intuitionistic *tends* to make more distinctions, but also some new identifications.

Acknowledgement. This work would not have been possible without the fruitful discussions with, and ideas of, my supervisor Ulrich Kohlenbach. I am also grateful to him for essential remarks on drafts of this paper and my master's thesis.

References

1. Y. Akama, S. Berardi, S. Hayashi, and U. Kohlenbach. An arithmetical hierachy of the laws of excluded middle and related principles. In *Proc. of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004),* 2004. To appear.
2. E. Bishop. *Foundations of constructive analysis.* McGraw-Hill Book Co., New York, 1967.
3. D. S. Bridges. A constructive look at functions of bounded variation. *Bull. London Math. Soc.,* 32(3):316–324, 2000.
4. S. Hayashi and M. Nakata. Towards limit computable mathematics. In *TYPES,* pages 125–144, 2000.
5. H. Ishihara. An omniscience principle, the König lemma and the Hahn-Banach theorem. *Z. Math. Logik Grundlag. Math.,* 36(3):237–240, 1990.
6. H. Ishihara. Informal Constructive Reverse Mathematics. Research Report Series 229, Centre for Discrete Mathematics and Theoretical Computer Science, 2004.

7. U. Kohlenbach. Effective moduli from ineffective uniqueness proofs. An unwinding of de la Vallée Poussin's proof for Chebycheff approximation. *Ann. Pure Appl. Logic,* 64(1):27–94, 1993.
8. U. Kohlenbach. Relative constructivity. *J. Symbolic Logic,* 63(4):1218–1238, 1998.
9. U. Kohlenbach. Things that can and things that can't be done in PRA. *Annals of Pure and Applied Logic,* 102(3):223–245, 2000.
10. M. Mandelkern. Constructive continuity. *Mem. Amer. Math. Soc.,* 42(277), 1983.
11. M. Mandelkern. Limited omniscience and the Bolzano-Weierstrass principle. *Bull. London Math. Soc.,* 20(4):319–320, 1988.
12. M. B. Pour-El and J. I. Richards. *Computability in analysis and physics.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1989.
13. F. Richman. Omniscience principles and functions of bounded variation. *MLQ Math. Log. Q.,* 48(1):111–116, 2002.
14. J. R. Shoenfield. *Recursion theory,* volume 1 of *Lecture Notes in Logic.* Association for Symbolic Logic, Urbana, IL, 2001. Reprint of the 1993 original.
15. S. G. Simpson. *Subsystems of second order arithmetic.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1999.
16. E. Specker. Der Satz vom Maximum in der rekursiven Analysis. In *Constructivity in mathematics: Proceedings of the colloquium held at Amsterdam, 1957 (edited by A. Heyting),* Studies in Logic and the Foundations of Mathematics, pages 254–265, Amsterdam, 1959. North-Holland Publishing Co.
17. M. Toftdal. Calibration of ineffective theorems of analysis in a constructive context. Master's thesis, Department of Computer Science, University of Aarhus, May 2004.
18. A. S. Troelstra, editor. *Metamathematical investigation of intuitionistic arithmetic and analysis.* Springer Verlag, Berlin, 1973.
19. A. S. Troelstra and D. van Dalen. *Constructivism in mathematics. Vol. I,* volume 121 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., Amsterdam, 1988.

Efficiently Computing Succinct Trade-Off Curves

Sergei Vassilvitskii[1] and Mihalis Yannakakis[2]

[1] Stanford University, Stanford, CA
sergei@cs.stanford.edu
[2] Columbia University, New York, NY
mihalis@cs.columbia.edu

Abstract. Trade-off (aka Pareto) curves are typically used to represent the trade-off among different objectives in multiobjective optimization problems. Although trade-off curves are exponentially large for typical combinatorial optimization problems (and infinite for continuous problems), it was observed in [PY1] that there exist polynomial size ϵ approximations for any $\epsilon > 0$, and that under certain general conditions, such approximate ϵ-Pareto curves can be constructed in polynomial time. In this paper we seek general-purpose algorithms for the *efficient approximation* of trade-off curves *using as few points as possible.* In the case of two objectives, we present a general algorithm that efficiently computes an ϵ-Pareto curve that uses at most 3 times the number of points of the smallest such curve; we show that no algorithm can be better than 3-competitive in this setting. If we relax ϵ to any $\epsilon' > \epsilon$, then we can efficiently construct an ϵ'-curve that uses no more points than the smallest ϵ-curve. With three objectives we show that no algorithm can be c-competitive for any constant c unless it is allowed to use a larger ϵ value. We present an algorithm that is 4-competitive for any $\epsilon' > (1+\epsilon)^2 - 1$. We explore the problem in high dimensions and give hardness proofs showing that (unless P=NP) no constant approximation factor can be achieved efficiently even if we relax ϵ by an arbitrary constant.

1 Introduction

When evaluating different solutions from a design space, it is often the case that more than one criterion come into play. For example, when choosing a route to drive from one point to another, we may care about the time it takes, the distance travelled, the complexity of the route (e.g. number of turns), etc. When designing a (wired or wireless) network, we may consider its cost, its capacity (the load it can carry), its coverage, etc. When solving computational problems we care about their use of resources such as time, memory, and processors.

Such problems are known as *multicriteria* or *multiobjective* problems. The area of multiobjective optimization has been extensively investigated for many years with a number of conferences and books (e.g. [Cli,Ehr]). In such problems we are interested in the trade-off between the different objectives. This is captured by the *trade-off* or *Pareto curve,* the set of all feasible solutions whose vector of the various objectives is not dominated by any other solution.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1201–1213, 2004.

The trade-off curve represents the range of reasonable possibilities in the design space. Typically we have a small number of objectives (2, 3, ...) and we wish to plot the trade-off curve to get a sense of the design space. Unfortunately, often the trade-off curve has exponential size for discrete optimization problems even for two objectives (and it is typically infinite for continuous problems).

Recently we started a systematic study of multiobjective optimization based on an approximation that circumvents the aforementioned exponential size problem [PY1,PY2]. The approach is based on the notion of an ϵ-*Pareto curve* (for $\epsilon > 0$), which is a set of solutions that approximately dominate every other solution. More specifically, for every solution s, the ϵ-Pareto curve contains a solution s' that is within a factor $(1 + \epsilon)$ of s, in all of the objectives. Such an approximation was studied before for certain specific problems, most notably for multicriteria shortest paths, where Hansen [Han] and Warburton [Wa] showed how to construct an ϵ-Pareto curve in polynomial time.

It was shown in [PY1] that every multiobjective optimization problem with a fixed number of polynomially computable objective functions (as is commonly the case) posesses an ϵ-Pareto curve of size polynomial in the size of the instance and $1/\epsilon$, for every $\epsilon > 0$. Generally, however such an approximate curve may not be constructible in polynomial time. A necessary and sufficient condition for its efficient computability is the existence of an efficient algorithm for the following multiobjective version of the *Gap* problem: Given a vector of values b, either compute a solution that dominates b, or determine that there is no solution that is better than b by at least a factor of $1 + \epsilon$ in all objectives. Several classes of problems (including specifically shortest paths, spanning trees, matching, and others) are shown in [PY1,PY2] to satisfy this property and hence have polynomially constructible ϵ-Pareto sets.

Although the theorem and construction of [PY1] yield a polynomial size ϵ-Pareto set, the set is not exactly "small": for d objectives, it has size roughly $(m/\epsilon)^{d-1}$, and the construction requires $(m/\epsilon)^d$ calls to the *Gap* routine. Here m is the number of bits used to represent the values in the objective functions. (We give the precise definitions of the framework and the parameters in the next section.)

Note that an ϵ-Pareto set is not unique: many different subsets may qualify and it is quite possible that some are very small while others are very large (without containing any redundant points). Having a small approximate Pareto set gives a succinct outline of the trade-offs involved and is important for many reasons. For example, often the representative set of solutions is investigated further by humans to assess the different choices and pick a suitable one, based on factors that are perhaps not quantifiable.

Suppose that our problem instance has a small ϵ-Pareto set. Can we find one? Furthermore, can we find one, while spending time that is proportional to the size of the small computed set, rather than the worst case set? These are the questions we investigate in this paper. We seek general algorithms that apply in all polynomial cases, i.e. whenever a *Gap* routine as above is available.

In the next section, we define the framework. In Section 3 we study the case of two objectives. We present a general algorithm that for any $\epsilon > 0$ computes an ϵ-Pareto set that has size at most 3 times k, the size of the smallest ϵ-Pareto set. This algorithm uses only $O(k \log(m/\epsilon))$ calls to a *Gap* routine (this is the dominant factor in the running time). We show a matching lower bound on the approximation ratio, i.e. there is no general algorithm that can do better than 3. However, if we relax ϵ to any $\epsilon' > \epsilon$, then we can efficiently construct an ϵ'-curve that uses no more points than k points.

We also discuss the dual problem: Given a bound, k, on the number of points we are willing to have, how good of an approximation (how small of an ϵ) can we get? For example, if $k = 1$, this is the so-called *knee* problem: if we pick one point to minimize the ratio for all objectives, what should that compromise point be, and what is the ratio? We show that the ratio can be approximated arbitrarily closely.

In Section 4 we study the case of three objectives. We show that no general algorithm can be within any constant factor c of the smallest ϵ-Pareto set unless it is allowed to use a larger ϵ-value. We present an algorithm that achieves a factor of 4 for any $\epsilon' > (1+\epsilon)^2 - 1$ ($\approx 2\epsilon$ for small ϵ). Furthermore, our algorithm again uses only $O(k \log(m/\epsilon))$ *Gap* calls.

Finally in Section 5 we discuss the case of an arbitrary number of objectives. We show that even if the solution points are given to us explicitly in the input, we cannot efficiently approximate the size of the smallest ϵ-Pareto curve: the problem is equivalent to the Set Cover problem. Furthermore, no constant factor approximation can be efficiently achieved, even if we relax ϵ by an arbitrary constant.

2 Preliminaries

A multiobjective optimization problem has a set of *instances,* every instance x has a set of solutions $S(x)$. There are d objective functions, $f_1, \ldots, f_d$, each of which maps every instance x and solution $s \in S(x)$ to a positive rational number $f_j(x, s)$. The problem specifies for each objective whether it is to be maximized or minimized. We say that a d-vector u *dominates* another d-vector v if it is at least as good in all the objectives, i.e. $u_j \geq v_j$ if f_j is to be maximized ($u_j \leq v_j$ if f_j is to be minimized); the domination is strict if at least one of the inequalities is strict. Similarly, we define domination between any solutions according to the d-vectors of their objective values. Given an instance x, the *Pareto set* $P(x)$ is the set of undominated d-vectors of values of the solutions. As usual we are also interested in solutions that realize these values, but we will often blur the distinction and refer to the Pareto set also as a set of solutions that achieve these values.

We say that a d-vector u *c-covers* another d-vector v if u is at least as good as v up to a factor of c in all the objectives, i.e. $u_j \geq v_j/c$ if f_j is to be maximized ($u_j \leq cv_j$ if f_j is to be minimized). Given an instance x and $\epsilon > 0$, an *ϵ-Pareto set* $P_\epsilon(x)$ is a set of d-vectors of values of solutions that $(1+\epsilon)$-cover all vectors in

$P(x)$; i.e., for every $u \in P(x)$, there exists a $u' \in P_\epsilon(x)$ such that u' $(1+\epsilon)$-covers u. For a given instance, there may exist many ϵ-Pareto sets, and they may have very different sizes.

To study the complexity of the relevant computational problems, we assume as usual that instances and solutions are represented as strings, that solutions are polynomially bounded and polynomially recognizable in the size of the instance, and that the objective functions are polynomially computable. In particular this means that each value $f_j(x,s)$ is a positive rational whose numerator and denominator have at most m bits, where $m \leq p(|x|)$, for some polynomial p. It is shown in [PY1] that for every multiobjective problem in this framework, for every instance x and $\epsilon > 0$ there exists an ϵ-Pareto set $P_\epsilon(x)$ of size at most $O((4m/\epsilon)^{d-1})$. Furthermore, for every fixed d, there is an algorithm for constructing a $P_\epsilon(x)$ in time polynomial in $|x|$ and $1/\epsilon$ (i.e., a fully polynomial time approximation scheme - FPTAS) if and only if there is a subroutine GAP that solves the following problem in time polynomial in $|x|$ and $1/\epsilon$: given x and d-vector b, either return a solution whose vector dominates b or report that there does not exist any solution whose vector is better than b by more than a $(1+\epsilon)$ factor in all of the coordinates. For simplicity, we will usually drop the instance x from the notation and use $\mathrm{GAP}_\epsilon(\mathrm{b})$ to denote the solution returned by the subroutine. To make the presentation easier, we will also say that GAP returns YES if it returns a solution, and returns NO otherwise.

We will assume from now on that the routine GAP exists, and will present our algorithms using this subroutine as a black box. We say that such an algorithm is *generic*, as it is not geared to a particular problem, but applies to all of the problems for which we can construct an ϵ-Pareto set in polynomial time.

3 Two Objectives

3.1 Lower Bound

Theorem 1. *There is no generic algorithm that approximates the size of the smallest ϵ-Pareto set to a factor better than 3 in the biobjective case. In particular, there is a biobjective problem with a polynomial time* GAP *procedure that cannot be approximated within a factor better than 3 unless P=NP.*

Proof. Suppose we have minimization objectives (the same holds for maximization or mixed objectives). Here we exploit the fact that there are points b on which $\mathrm{GAP}_\delta(\mathrm{b})$ is not uniquely defined. Consider the following set of points $p = (p_x, p_y)$, $q = \left(p_x(1+\epsilon)+1, \frac{p_y}{1+\epsilon}\right)$ and $r = \left(p_x(1+\epsilon)+1, \frac{p_y-1}{(1+\epsilon)}\right)$. Let $P = \{p, q\}$ and $Q = \{p, q, r\}$. The smallest ϵ-Pareto set for P consists of only one point, while the smallest ϵ-Pareto set for Q must include at least two points.

Consider the points b where $\mathrm{GAP}_\delta(\mathrm{b})$ can return r; these are the points which r dominates in both objectives. Now if we throw out the points where GAP_δ can also return q, we notice that for the points remaining $\mathrm{GAP}_\delta(\mathrm{b})$ can return NO. But then, using the GAP_δ function as a black box, we cannot say whether

or not r is part of the solution, and thus are forced to take at least two points, even when we are presented with the set P.

We can make a symmetric observation if instead of q and r we have $q' = \left(\frac{p_x}{1+\epsilon}, p_y(1+\epsilon)+1\right)$ and $r' = \left(\frac{p_x-1}{(1+\epsilon)}, p_y(1+\epsilon)+1\right)$. Here again using the GAP_δ routine we cannot decide if the point r' is in the solution space or not. Combining the two bad cases, we see that we cannot tell if the size of the optimal solution is one point, as it is if $P = \{p, q, q'\}$ or if it is three points, as it is when $P = \{p, q, r, q', r'\}$.

We can turn this into an NP-hardness proof by defining a suitable biobjective problem so that the points r, r' are present iff a given instance of the Partition problem has a solution. Then it is NP-hard to determine whether the smallest ϵ-Pareto set has 1 point or needs 3 points. Finally, if we wish, we can expand the solution set replicating this configuration a number of times k on the plane far apart from each other, and show that, for any k, it is NP-hard to determine whether the smallest ϵ-Pareto set has k points or needs $3k$ points. (Details omitted.)

Note however, that the lower bound above is brittle, for if the algorithm is allowed to return an ϵ'-Pareto set for any $\epsilon' > \epsilon$, the proof no longer holds. In fact we will show below, that for any $\epsilon' > \epsilon$ there is an algorithm that finds an ϵ'-Pareto set $P_{\epsilon'}$, of size no bigger than the optimal ϵ-Pareto set.

3.2 2-Objective Algorithms

We assume for concreteness that both objectives are to be minimized; the algorithm is similar in the other cases. We recall here the original algorithm of [PY1, PY2]. To compute an ϵ-Pareto set, and in fact prove a polynomial bound on its size, consider the following scheme. Divide the space of objective values geometrically into rectangles, such that the ratios of the large to the small coordinates is $(1+\epsilon')=\sqrt{1+\epsilon}$ in all dimensions; equivalently if we switch to a log-log scale of the objective vaues, the plane is partitioned arithmetically into squares of size $\log(1+\epsilon')$ ($\approx \epsilon/2$ for small ϵ). Proceed to call $\mathrm{GAP}_{\epsilon'}$ on all of the rectangle corner points, and keep an undominated subset of all points returned. It is easy to see that this forms an ϵ-Pareto set. (To prove that this set cannot be too large, note that we can discard points until there is at most one remaining in each of the rectangles.) If m is the maximum number of bits in the numerator and denominator of the objective functions, then the ratio of the largest to the smallest possible objective value is 2^{2m}, hence the number of subdivisions in each dimension is $2m/\log(1+\epsilon') \approx 4m/\epsilon$ for small ϵ.

This algorithm gives no guarantees on the size of the ϵ-Pareto set it returns with respect to P^*_ϵ, the smallest ϵ-Pareto set. To compute a 3-approximation to P^*_ϵ we will proceed in two phases. We will first compute an ϵ'-Pareto set for a particular $\epsilon' < \epsilon$ and then delete points from this set until we are left with a small ϵ-Pareto set.

We begin again by partitioning the space into rectangles, this time with a coordinate ratio of $(1+\epsilon') = \sqrt[4]{1+\epsilon}$. Look at the set of corner points, as a set of

points on the x-y plane. The algorithm consists of two repeating steps. ZAG(b) returns a corner point p with minimum y value, such that $\mathrm{GAP}_{\epsilon'}(\mathrm{p}) = \mathrm{YES}$ and $x(p) \leq x(b)$,where $x(p)$ is simply the x-coordinate of p. The second step, ZIG(b) returns a corner point p with minimum x value, such that $\mathrm{GAP}_{\epsilon'}(\mathrm{p}) = \mathrm{YES}$ and $y(p)/(1+\epsilon') \leq y(b)$.

In the first phase, the algorithm will discover the Pareto set by scanning the range of possible values in order of decreasing x (increasing y) coordinate. Let p be the point that has the maximum possible value in both objectives. Clearly $\mathrm{GAP}_{\epsilon'}(\mathrm{p}) = \mathrm{YES}$. We then find the point $\mathrm{q_1} = \mathrm{ZIG(ZAG(p))}$. The point q_i is found as $\mathrm{ZIG(ZAG}(\mathrm{q_{i-1}}/(1+\epsilon')))$. The first phase terminates when the ZAG step returns NO. We then return the set $Q = \{q_1, q_2, \ldots, q_m\}$.

Lemma 1. *The* ZIGZAG *algorithm above returns a set Q that $(1+\epsilon')^2$-covers the set P of all solution points.*

Proof. The algorithm maintains several invariants. First, the x coordinates of points $q_1, \ldots, q_m$ form a strictly decreasing sequence, while the y coordinates form a strictly increasing sequence. With that in mind, the claim below implies by induction that Q is a $(1+\epsilon')^2$-cover.

Claim. The point q_i $(1+\epsilon')^2$-covers all of the points in P with their x-coordinate between $x(q_{i-1})/(1+\epsilon')$ and $x(q_i)/(1+\epsilon')$.

Proof. Suppose there exists some point p in the solution space, and $q_{i-1}, q_i \in Q$ such that $x(q_i)/(1+\epsilon') \leq x(p) \leq x(q_{i-1})/(1+\epsilon')$, and q_i does not $(1+\epsilon')^2$-cover p. In that case $y(p) < y(q_i)/(1+\epsilon')^2$. But then the y value of $\mathrm{ZAG}(\mathrm{q_{i-1}})(1+\epsilon')^2$ must be less than $y(p)$ by definition of $\mathrm{GAP}_{\epsilon'}$, and further, $\mathrm{y(ZIG(ZAG}(\mathrm{q_{i-1}}))) \leq \mathrm{y(ZIG}(\mathrm{q_{i-1}}))$. Therefore $y(q_i)/(1+\epsilon')^2 \leq y(p)$, a contradiction.

Thus Q forms an $(1+\epsilon')^2$-cover of P.

Lemma 2. *The size of Q returned by* ZIGZAG *above, is no more than 11 times the size of the smallest ϵ-Pareto set, P^*_ϵ.*

Proof. Sketch: Let P^*_ϵ be the smallest ϵ-Pareto set, and let p^* be a point in P^*_ϵ. We charge those points in Q that are $(1+\epsilon)$-covered by p^* to the point p^*. We prove that every point p^* is charged with at most 11 points of Q. We omit the details of the proof.

In the second phase we reduce the set Q to make it a small ϵ-Pareto set. Note that if R is any $(1+\epsilon')^2$-cover of the points in Q, then for any point in the original solution space, there is some point in R that $(1+\epsilon')^2(1+\epsilon')^2 = (1+\epsilon)$-covers it, in other words R is an ϵ-Pareto set. Further, since there is a total order on points in Q, we can compute the smallest $(1+\epsilon')^2$-cover by a simple greedy algorithm. At the end of phase 1 the points in Q are already sorted by their x value. Given q_1 we now find a point q_i with the smallest x coordinate such that $q_i/(1+\epsilon')^2 \leq q_1$ in all dimensions. Add q_i to R, remove points $q_1, \ldots, q_{i-1}$ and repeat until all of the points are covered.

Lemma 3. *The size of the smallest* $(1+\epsilon')^2$*-cover of* Q *is no more than* $3 \cdot |P_\epsilon^*|$.

Proof. Consider any point p^* in P_ϵ^* again, and let q be a point of Q that $(1+\epsilon')^2$ covers p^*. If we add q to our cover, the points in Q that are $(1+\epsilon)$ covered by p^* but are not $(1+\epsilon)$ covered by q are split into two disjoint groups: those above q and those below q, each of which can always be $(1+\epsilon')^2$-covered by a single point.

We now proceed to analyze the running time of the algorithm. Let k be the total number of points in the optimal ϵ-Pareto set, $k = |P_\epsilon^*|$. Recall that m denotes the number of bits in the objective functions. To avoid clutter in the expressions below, we will use ϵ in place of $\log(1+\epsilon)$, which is a valid approximation for small ϵ (for large ϵ simply drop this factor). In the end of the first phase we produce $O(k)$ points. To find each point, we called one execution of ZIG and one of ZAG. Both of these can be implemented as binary searches on $O(m/\epsilon)$ points. Therefore, the runtime of the first part is bounded by $O(k \log(m/\epsilon))$ $\mathrm{GAP}_{\epsilon'}$ calls. The greedy algorithm as described is linear, and its time is subsumed by that of the first phase. Therefore the overall runtime is $O(k \log(m/\epsilon))$ $\mathrm{GAP}_{\epsilon'}$ calls.

Theorem 2. *The* ZIGZAG *algorithm with the cleanup step as described above computes a 3-approximation to the smallest* ϵ*-Pareto set in time* $O(k \log(m/\epsilon))$ $\mathrm{GAP}_{\epsilon'}$ *calls.*

Suppose that we are allowed to return an ϵ'-Pareto set for a value $\epsilon' > \epsilon$ and let δ be such that $(1+\epsilon') = (1+\epsilon)(1+\delta)^4$. We can proceed in a way similar to above: first divide the grid with the coordinate ratio of $(1+\delta)$, and use ZIGZAG with GAP_δ. In the cleanup phase, we look for the smallest $(1+\delta)^2(1+\epsilon)$ cover of the points from the first phase.

Theorem 3. *For* $(1+\epsilon') = (1+\epsilon)(1+\delta)^4$ *we can find an* ϵ'*-Pareto set* R *such that* $|R| < |P_\epsilon^*|$ *in time* $O(k \log(m/\delta))$ GAP_δ *calls.*

Proof. Since a $(1+\epsilon)(1+\delta)^2$ cover of a $(1+\delta)^2$-Pareto set will form a $(1+\epsilon)(1+\delta)^4 = (1+\epsilon')$-Pareto set, the only step we need to prove is that the size of the smallest such cover is no bigger than the size of the smallest ϵ-Pareto set. We proceed the same as above, by picking a point $p^* \in P_\epsilon^*$. Then there's a point $q \in Q$ such that $q(1+\delta)^2$ covers p^*. That implies, that all of the points that are $(1+\epsilon)$-covered by p^* are $(1+\delta)^2(1+\epsilon)$-covered by q. Therefore the smallest $(1+\delta)^2(1+\epsilon)$ cover of all of the points in Q is of size no bigger than $|P_\epsilon^*|$

3.3 Computing the Best k Solutions

Now let's consider the dual problem: We want to compute a set of k solutions that collectively approximate as well as possible the Pareto curve. That is, we wish to find a set S of k solutions that minimizes the value of the ratio r such that S r-covers the whole set P of solutions. For $k = 1$, this solution is the "knee" of the Pareto set.

As shown at the beginnining of the section, finding the knee (and more generally the best k points) is NP-hard even in simple cases. However, we can approximate the optimal ratio r, within any degree of accuracy $1+\delta$. For the knee case it is easy to do this using $O(\log^2(m/\delta))$ GAP_δ calls by a simple binary search(we omit the details). For general k, we use our above algorithm for the $\epsilon' > \epsilon$ case to show the following (proof omitted).

Theorem 4. *We can approximate the smallest ratio $1+\epsilon$ for which the ϵ-Pareto set has at most k points to a factor of $1+\delta$ in time $O(k\log^2(m/\delta))$ GAP_δ calls.*

3.4 Applications

The results here can be applied to all of the problems which have the required *Gap* routine, e.g. the classes of problems shown in [PY1,PY2], including multiobjective flows and other convex problems, shortest path, spanning tree, matching and cost-time trade-offs in query evaluation.

In some cases, better bounds can be proved, by using a sharper routine than *Gap*. We discuss briefly the case of shortest paths, with two objectives, cost and length. A stronger variant of the *Gap* problem in this case is the well-studied Restricted Shortest Path (RSP) problem: given a bound b on the cost of the path, minimize the length of the path subject to the bound on the cost. There exists an FPTAS for RSP with running time $O(en/\epsilon)$, where n is the number of nodes and e the number of edges [ESZ].

The RSP routine can be used directly to implement both the ZIG and the ZAG steps in the algorithm. Hence we can compute the smallest ϵ-Pareto set for bicriteria shortest paths in time $O(\frac{enk}{\epsilon})$ where k is the size of the smallest ϵ-Pareto set.

4 Three Objectives

4.1 Lower Bound

Theorem 5. *Any generic algorithm computing the smallest ϵ-Pareto set for a problem with more than two objective functions cannot be c-competitive for any constant c.*

Proof. Just like in the case of 2 objectives we will exploit the fact that $\text{GAP}_\delta(b)$ is not uniquely defined for some points b. Again construct two sets, P and Q such that GAP_δ cannot distinguish between them, and the size of the optimal ϵ-Pareto set for P has one point, and for Q has arbitrarily many points. Consider a point $p = (p_x, p_y, p_z)$, and let $q_i = \left(p_x(1+\epsilon)^i, p_y(1+\epsilon)^{k-i}, \frac{p_z}{1+\epsilon}\right)$ for $i = 0 \ldots k$. Let $P = \{p, q_0, \ldots, q_k\}$. Clearly, $\{p\}$ is an ϵ-Pareto set for P. Let $r_i = \left(p_x(1+\epsilon)^i, p_y(1+\epsilon)^{k-i}, \frac{p_z-1}{(1+\epsilon)}\right)$. Notice that p does not $(1+\epsilon)$-cover any of the r_is and none of the q_is cover p. So if we let $Q = \{p, q_0, \ldots q_k, r_0, \ldots, r_k\}$, the smallest ϵ-Pareto set for Q will have $\Theta(k)$ points. But again GAP_δ cannot

distinguish between the two cases, since for all b where $\mathrm{GAP}_\delta(\mathrm{b})$ can return r_i it can either return q_i or return NO. Therefore, we cannot conclude if the size of the optimal solutoin is one point, or $\Theta(k)$ points for arbitrary k. Again, we can turn this into an NP-hardness proof by specifying a suitable 3-objective problem.

In order to beat the lower bound above, we are forced to search for algorithms which will return ϵ'-Pareto sets, for $\epsilon' > \epsilon$ when the original problem has 3 or more objectives.

4.2 Three Objectives Algorithm

We will present an algorithm that is 4-competitive and returns an ϵ'-Pareto set for $(1+\epsilon') > (1+\epsilon)^2$. Choose a suitable small $\delta > 0$ such that $(1+\epsilon') > (1+\epsilon)^2(1+\delta)^4$; it is convenient to pick a δ such that $(1+\epsilon)$ is a power of $(1+\delta)$. As before, we will be working with a geometric grid of the space of objective values (equivalently, an arithmetic grid in the log-log scale). We let the ratio of the grid in the x dimension be $(1+\delta)$ and in the y, z dimensions be $(1+\epsilon)(1+\delta)$. Let C be the set of all corner points of the grid where GAP_δ returns a solution (We will not be computing these points explicitly).

Assume for concreteness again that all objectives are to be minimized; the algorithm is similar in the other cases. We will outline first the algorithm, then prove its correctness, and finally sketch an efficient implementation.

The algorithm computes a set of corner points Q such that $\mathrm{GAP}_\delta(Q) = \{\mathrm{GAP}_\delta(q) \mid q \in Q\}$ is an ϵ'-Pareto set of size at most 4 times the size of the optimal ϵ-Pareto set P^*_ϵ. We say that a corner point $r \in C$ is *ineligible* at some time during the algorithm if there is a point $q \in Q$ such that $x(r) \leq x(q)(1+\epsilon)^2(1+\delta)^2$, $y(r) \leq y(q)(1+\epsilon)(1+\delta)$ and $z(r) \leq z(q)(1+\epsilon)(1+\delta)$; the conditions are asymmetric because of the asymmetry in the grid ratios. The corner point r is called *eligible* otherwise, and we let C/Q denote the set of eligible corner points. For a set S of points, we use $\min_x S$ to denote the subset of points of S that have minimum x coordinate, similarly define $\min_y S$ and $\min_z S$.

$Q = \emptyset$
While $C/Q \neq \emptyset$ do the following:
Find the point $p \leftarrow \min_y \min_x \min_z C/Q$.
$S(p) = \{s \in C : x(s) \leq x(p)(1+\epsilon)(1+\delta), y(s) \leq y(p), z(s) \leq z(p)(1+\epsilon)(1+\delta)\}$.
$T(p) = \{t \in C : x(t) \leq x(p), y(t) \leq y(p)(1+\epsilon)(1+\delta), z(t) \leq z(p)(1+\epsilon)(1+\delta)\}$.
Let $s(p) \in \min_y S(p)$ and $t(p) \in \min_x T(p)$ be minimal (undominated) points in the corresponding sets.
Update $Q \leftarrow Q \cup \{s(p), t(p)\}$.

To simplify notation, we blur below the distinction between the selected set Q of corner points and the corresponding set $\mathrm{GAP}_\delta(Q) = \{\mathrm{GAP}_\delta(\mathrm{q}) \mid \mathrm{q} \in \mathrm{Q}\}$ of feasible solution points derived from it; note that every point $q \in Q$ is dominated by $\mathrm{GAP}_\delta(q)$.

Theorem 6. *The set Q computed by the above algorithm forms a ϵ'-Pareto set, and $|Q| \leq 4|P^*_\epsilon|$.*

Proof. Sketch: Let P^*_ϵ be the optimal ϵ-Pareto set. We will charge each point of Q to a point of P^*_ϵ so that every point $p^* \in P^*_\epsilon$ is charged with at most 4 points of Q. The details of the proof are quite involved and are omitted from this extended abstract.

Observe that given p, it is easy to find $s(p)$ and $t(p)$ in $O(\log(m/\delta))$ GAP_δ calls using the binary search technique, as in the 2-objective case. The question remains of how to efficiently find $p \leftarrow \min_x \min_z C/Q$.

An obvious solution is to scan through the z values from smallest to largest, and at each z value use the 2 objective algorithm to find p. Ignore the point if it is already covered by another point in Q and continue. However, this involves both a linear scan through all z values (of cost at least $\Omega(m/\delta)$) and potentially many points p which are covered by others in Q.

Lemma 4. *We can compute $p \leftarrow \min_y \min_x \min_z C/Q$ using $O(\log(m/\delta))$ GAP_δ calls.*

Proof. Since we consider points in increasing z value, we only need to consider the x-y projection of the points in Q and maintain the frontier of points undominated in x and y. These points are sorted in increasing order by their x coordinate, as $q_1, \ldots, q_l$. The x-y projection of the corner points that are eligible (i.e. not covered by Q) is the region below a rectilinear curve that passes through a translation of the points q_j. The convex corners of the region are $c_j = \left(\frac{x(q_{j+1})}{(1+\epsilon)^2(1+\delta)^3}, \frac{y(q_j)}{(1+\epsilon)^2(1+\delta)^2}\right)$, $j = 0, \ldots, l$. (We let $y(c_0), x(c_l)$ be the maximum posible values of the objectives, and omit the points whose values are below the minimum.) Observe that every eligible point dominates one of the c_j's. For each j let $h_j = \text{minimum } z$ such that $\text{GAP}_\delta\left(\frac{x(q_{j+1})}{(1+\epsilon)^2(1+\delta)^3}, \frac{y(q_j)}{(1+\epsilon)^2(1+\delta)^2}, z\right)$ returns YES. We can compute h_j via a binary search in $O(\log(m/\delta))$ GAP_δ calls for each j. We maintain the h_j's in a priority queue H. When we add a new point to Q, we may eliminate some of the elements of Q (if they become dominated), and we will create at most two more intervals. The computation of two more h_j values can be done in the time allotted.

Now, instead of doing a linear scan through the z values, we can immediately jump to the next z value where we will be guaranteed to find a point in C/Q. Once we find the z value, we limit our search to an appropriate interval to seek the next point from C/Q. We can find the point using an algorithm similar to the ZIGZAG algorithm presented for the 2-d case.

Theorem 7. *The algorithm to compute the ϵ'-Pareto set Q can be implemented to run in time $O(|Q|\log(m/\delta))$ GAP_δ calls.*

5 d Objectives

We have shown that for $d \geq 3$ objectives we are forced to compute an $\epsilon' > \epsilon$-Pareto set, if we are to have a guarantee on its size. In fact, we can easily find a $\log n$-competitive algorithm for the problem. Let $(1+\epsilon') = (1+\epsilon)(1+\delta)^2$.

Theorem 8. *For any $\epsilon' > \epsilon$ we can compute an ϵ'-Pareto set Q such that $|Q| \leq |(\log(d \log(m/\delta))|P_\epsilon^*|$ using $O((m/\delta)^d)$ GAP calls.*

Proof. The algorithm will proceed in two stages. In the first stage, we will compute a δ-Pareto set, by using the original algorithm. Break up the solution space using a geometric grid of size $\sqrt{1+\delta}$ and call $\text{GAP}_{\sqrt{1+\delta}}$ on all of the corner points, while keeping an undominated subset R. Note that $|R| \leq O(m/\delta)^{d-1}$. Now we can phrase the problem as a set cover problem. Let the universe be all of the points in R, and for each $r \in R$ associate a set $S_r = \{$ the points that are $(1+\epsilon)(1+\delta)$-covered by $r\}$. The smallest set cover, will yield an $(1+\epsilon)(1+\delta)^2$-Pareto set, Q. Since we can compute a $\log n$ approximate for the Set Cover problem on a universe of size n, the result follows.

Unfortunately, the algorithm above is the best that we know of for $d > 3$. There are two aspects in which this algorithm is inferior to the ones we presented earlier (even for fixed d): The approximation ratio is not constant, and the running time grows with m rather than $\log m$.

We show that the ratio of above algorithm is probably close to the best possible in very high dimensions, even if the points are given explicitly as input.

Theorem 9. *The task of computing the smallest ϵ-Pareto set on d objectives is NP-hard to approximate to within $c \log d$ (for some $c < 1$) even if all the solution points are given explicitly.*

Proof. We will prove this via a gap-preserving reduction from *SetCover*. In a set cover instance we are given a universe of elements U and subsets $S_1, \ldots, S_l \subseteq U$. We are then asked to select a minimum number of subsets S_i such that their union is U. Our reduction is as follows: for each element $u_i \in U$ add a point p_i in the solution space, whose ith coordinate is $1/(1+\epsilon)$ and all other coordinates are at ∞. For each set S_j we add a point q_j such that the ith coordinate of q_j is 1 if $u_i \in S_j$ and $(1+\epsilon)^3$ otherwise. Finally, we add a point r with value $(1+\epsilon)$ in all dimensions.

Let P_ϵ be the smallest ϵ-Pareto set. Since r cannot be $(1+\epsilon)$-covered by any other points, r must be part of the final solution. Since every point p_i is $(1+\epsilon)$-covered in P_ϵ, the sets corresponding to the q_is must form a valid set cover. Finally it is easy to see that this approximation is gap preserving and the theorem follows.

Observe that the reduction above breaks down if we are allowed to relax the ϵ value, since for $(1+\epsilon') = (1+\epsilon)^2$ the ϵ'-Pareto set will always contain just the single point r. We show below that in high dimensions we cannot achieve

a constant factor approximation to the ϵ-Pareto set, even if we are allowed to relax ϵ by an arbitrary constant.

Theorem 10. *Let* $(1+\epsilon') < (1+\epsilon)^{\log^* d/3}$. *Even if all the solution points are given explicitly in the input, it is NP-hard to compute an* ϵ'*-Pareto set whose size is within a* $\log^* d$ *factor of the smallest* ϵ*-Pareto set.*

Proof. We will use a reduction from assymetric k-center along with a recent result by Chuzhoy et al. [CG+] to finish the proof. In the assymetric k-center problem we are given a set of points V with distances, $dist(u,v)$ that must satisfy the triangle inequality, but may be assymetric: i.e. $dist(u,v) \neq dist(v,u)$. We are asked to find a subset $U \subseteq V, |U| = k$, that minimizes $dist^* = \max_{v \in V} \min_{u \in U} dist(u,v)$.

Let us chose a value $dist'$ which will specify later, and encode the asymmetric k-center problem as follows. For each $v_i \in V$ create a point p_i such that, the ith coordinate of p_i is 1, and the jth coordinate is $(1+\epsilon)^{-\lceil dist_{ij}/dist' \rceil}$. Notice that if $dist' = dist^*$ then the smallest ϵ-Pareto set will contain precisely k points and correspond to the optimal solution. In a similar way if we can compute a $(1+\epsilon)^a$ Pareto set of size less than ck then we can approximate $dist^*$ to a factor of a while using less than ck centers. Thus, if we do a binary search on $dist'$ to find lowest distance that still uses fewer than ck centers, we can obtain an (a,c) approximation to the asymmetric k-center problem. However, this problem is hard to approximate to a factor of $\log^* n$ even when using $(\frac{1}{3}\log^* n)k$ centers.

References

[CJK] T. C. E. Cheng, A. Janiak, and M. Y. Kovalyov. Bicriterion Single Machine Scheduling with Resource Dependent Processing Times. *SIAM J. Optimization*, 8(2), pp. 617–630, 1998.

[CG+] J. Chuzhoy, S. Guha, E. Halperin, S. Khanna, G. Kortsartz, S. Naor Asymmetric k-center is $\log^* n$-hard to Approximate. To appear in *Proc. STOC 2004*.

[Cli] J. Climacao, Ed. *Multicriteria Analysis.* Springer-Verlag, 1997.

[Ehr] M. Ehrgott. *Multicriteria optimization*. Springer-Verlag, 2000.

[ESZ] F.Ergun, R.Sinha, and L.Zhang. An improved FPTAS for Restricted Shortest Path. *Information Processing Letters* 83(5):237-293. September 2002.

[GG+] S. Guha, D. Gunopoulos, N. Koudas, D. Srivastava, and M. Vlachos. Efficient Approximation of Optimization Queries Under Parametric Aggregation Constraints. *Proc. 29th VLDB,* 2003.

[Han] P. Hansen. Bicriterion Path Problems. *Proc. 3rd Conf. Multiple Criteria Decision Making Theory and Application,* pp. 109–127, Springer Verlag LNEMS 177, 1979.

[PY1] C.H.Papadimitriou, M.Yannakakis. On the Approximability of Trade-offs and Optimal Access of Web Sources. In *Proceedings 41st IEEE Symp. on Foundations of Computer Science*, 2000.

[PY2] C.H.Papadimitriou, M.Yannakakis. Multiobjective Query Optimization. In *Proceedings of PODS* 2001.

[RM+] R. Ravi, M.V. Marathe, S.S. Ravi, D.J. Rosenkrantz, and H.B. Hunt. Many Birds with One Stone: Multi-objective Approximation Algorithms. *Proc. 25th STOC*, pp. 438-447, 1993.

[Wa] A. Warburton. Approximation of Pareto Optima in Multiple-Objective Shortest Path Problems. *Operations Research*, 35, pp. 70-79, 1987.

On Randomization Versus Synchronization in Distributed Systems

Hagen Völzer

Institute for Theoretical Computer Science, University of Lübeck, Germany
Ratzeburger Allee 160, 23538 Lübeck, Germany
Ph/Fax: +49 451 500 5314/5301
voelzer@tcs.uni-luebeck.de

Abstract. We use techniques from concurrency theory to present two new impossibility results for asynchronous distributed systems with randomization. While these results are interesting by themselves, they also complete a picture of how particular liveness assumptions influence the solvability of distributed computation problems in the absence and presence of randomization.

1 Introduction

The benefit of randomization in distributed algorithms, i.e., agents flipping coins during the execution of their program, is well known. Randomized algorithms are often simpler and more efficient than their deterministic counterparts and they sometimes solve problems that deterministic algorithms provably cannot solve. Much less is known about the limitations of randomization with respect to the solvability of coordination and synchronization problems.

One of the most prominent examples showing the benefit of randomization is symmetry breaking. There, randomization allows us to construct symmetric algorithms that start in a symmetric state and lead to an asymmetric state with probability 1. Examples for symmetry breaking problems are leader election and mutual exclusion.

Hart, Sharir, and Pnueli [6] show however that it still depends on the particular problem whether a symmetric randomized solution exists. While there is a symmetric solution to the leader election problem, they show that every fully symmetric mutual exclusion *(mutex)* protocol where processors communicate via a shared variable with separate read/write operations deadlocks with positive probability. They conclude: "These phenomena call for further study to understand better the distinction between those concurrent problems that admit probabilistic solutions that are better than deterministic solutions, and those problems that do not benefit from the introduction of randomization."

We present two new results in this paper that show how the solution of mutex problems depends on the ability of the underlying model to synchronize independent resources while randomization does not influence the solvability of

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1214–1226, 2004.

these mutex problems. Moreover, the new results complete a picture on the expressiveness of liveness assumptions when combined with randomization, which is described below.

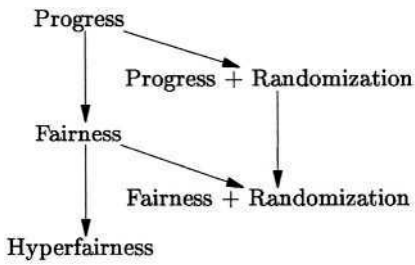


Fig. 1. The hierarchy of models

We distinguish three elementary, architecture independent liveness assumptions: *progress, fairness,* and *hyperfairness.* They roughly correspond to the concepts of weak fairness, strong fairness and hyperfairness in the literature, respectively. Together with known results, our results establish the picture shown in Fig. 1, where an arrow represents a strictly increasing ability to solve distributed computation problems.

Kindler and Walter [7] and, independently, Vogler [9], have shown that starvation-free mutual exclusion cannot be solved if only progress is assumed. All known mutex-solutions assume fairness at one place or another. As a new result, we show in this paper that mutex cannot be solved starvation-free with probability 1 by progress and randomization. This implies that fairness cannot be implemented with probability 1 by progress and randomization.

When we compare mutex with the classical message-passing consensus problem with crashing agents (as defined by Fischer et al. [5]) we can complete the picture. While mutex cannot be solved with progress and randomization, consensus can (Ben-Or's algorithm [2] does not need fairness) and, on the other hand, while consensus cannot be solved with fairness alone, mutex can (fairness is as strong as the liveness assumed by Fischer et al. [5]). Hence fairness and randomization are incomparable with respect to their expressive power and mutex and consensus are incomparable with respect to their solvability.

It is therefore interesting to investigate the power of the combination of fairness and randomization. Here we can obviously solve mutex as well as consensus. To show the limitation of this model, we consider a generalization of the mutex problem known as the dining philosophers problem [3]. We show that this problem cannot be solved with probability 1 if some crash-tolerance is additionally required—even if the starvation-freedom requirement is considerably weakened. Crash-tolerant generalized mutex can be solved through hyperfairness.

We use Petri nets and their partial-order semantics in this paper, the benefits of which will be discussed in the conclusion.

2 The Liveness Assumptions

This section introduces the used liveness assumptions by help of Petri nets. Fig. 2a shows an object x in state A that changes its state to B by the occurrence of transition t (t removes the token from A and puts a token into B). An object could be a processor, a shared register, or a message. Fig. 2b shows

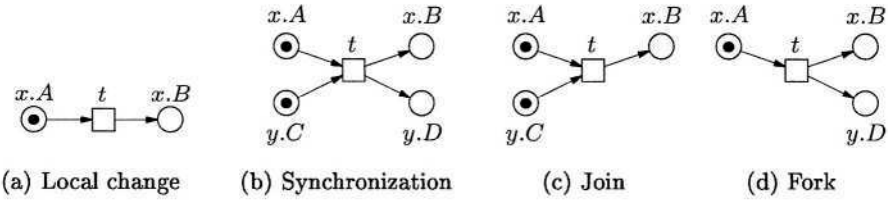


Fig. 2. Petri net modelling of state changes of objects

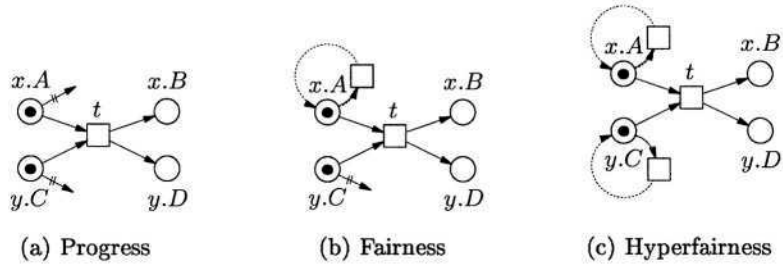


Fig. 3. Three liveness assumptions

a synchronization of the objects x and y. Both objects change their state simultaneously through the occurrence of t. Object x changes from state A to B and y changes from C to D. Object x could be a processor and y could be a shared register in this case. If object x is in state A we also say x is *ready* for t. Figs. 2c and d show a join and a fork, two special cases of the synchronization in Fig. 2b, which can represent a processor x receiving and x sending a message y, respectively.

We describe progress, fairness, and hyperfairness here for two-party synchronization. Generalization to n-party synchronization can be found in the remainder of this paper.

Progress means that both objects eventually synchronize in t if both wait for each other when ready for t, i.e., neither object engages in another transition departing from A and C, respectively. This is symbolically depicted by the negated arcs in Fig. 3a. Progress for t is the partial-order version[1] of *weak fairness* (also *justice*) for transition t. That is, t eventually occurs when continuously enabled. If x is a shared register, it means deadlock freedom of x—as long as there is some process that is ready to access x then some process will access x.

Fairness means that both objects eventually synchronize in t if one object waits for the other and the other object is always eventually ready, i.e., it also waits or it always returns to the state where it is ready for t (Fig. 3b). Fairness is the partial-order version of *strong fairness* (also *compassion*) for t, that is t occurs when always eventually enabled. If x is a shared register then fairness

[1] In a strict sense [11]; there is a subtle technical difference between progress and weak fairness, see e.g. [11] for details.

means wait-freedom (also starvation freedom) of x, that is, y will eventually access x when waiting to access x. For messages, it means that each message that is sent to a process x is eventually received by x even if another process independently sends infinitely many messages to x.

Hyperfairness means that both objects synchronize in t when both objects are always, independently of each other, eventually ready (Fig. 3c). This is a strong assumption, usually too strong to postulate as it assumes that both objects are eventually ready at the same time although they leave and enter their ready state independently of each other. Hyperfairness was introduced by Attie, Francez, and Grumberg [1] and our notion gives a new formalization for their concept. Here, hyperfairness occurs mainly for completing the picture.

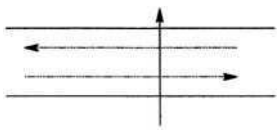


Fig. 4. Crossing a two-lane road

The relative power of the three assumptions is illustrated by the following example, which is a three-party synchronization. Consider a pedestrian who wants to cross a two-lane road (Fig. 4). To cross, the pedestrian needs two resources at the same time: a sufficiently wide space on the left lane and one on the right lane (assume that he cannot safely wait between the two lanes). Progress postulates that he will cross the road if both resources are permanently available, i.e., if no cars come. Fairness postulates that he will also eventually cross the road if one resource is permanently available and the other always eventually. This includes cases with infinitely many cars on one lane. Hyperfairness postulates that the pedestrian also eventually crosses if both resources are always, independently of each other, eventually available. This includes cases with infinitely many cars on both lanes.

3 Computational Model

We informally introduce various aspects of our computational model in this section. Formal definitions of all notions can be found in the appendix (in the full version).

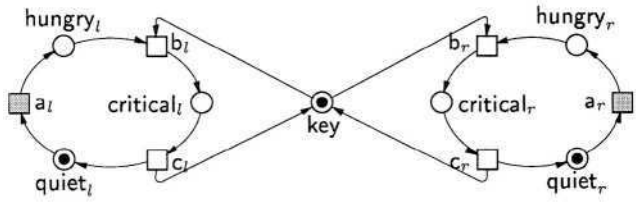


Fig. 5. Σ_1 – a mutex system

Figure 5 shows a simple mutex system Σ_1, modelled by a Petri net, with two agents l and r cycling through the three states *quiet, hungry,* and *critical* with a central key that is needed for entering the critical state. Transitions a_l and a_r are grey shaded to designate that no liveness is assumed for them, i.e., an agent may remain quiet forever. For all other transitions, progress is assumed. This still does not guarantee *starvation*

freedom of the mutex system, i.e., the requirement that each hungry agent eventually becomes critical, as one agent may remain hungry forever while the other is using the key infinitely often. However, fairness for b_l and b_r guarantees starvation-freedom.

Systems. A Petri net is represented by a set P of *places,* a set T of *transitions,* and a set $F \subseteq (P \times T) \cup (T \times P)$ of arcs. We graphically represent a place by a circle, a transition by a square, and an arc by an arrow between the corresponding elements. A *system* is a Petri net together with an initial marking (e.g. Σ_1 in Fig. 5). For a place or transition x, we denote the preset and the postset of x by ${}^\bullet x = \{y \mid (y,x) \in F\}$ and $x^\bullet = \{y \mid (x,y) \in F\}$, respectively. For a set X of places or transitions, let ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$. The classical firing rule defines the sequential behavior, i.e., sequential executions and computation trees. There is a natural prefix order on all computation trees of a system with a maximal computation tree that is unique up to isomorphism. A set C of transitions of a net such that $|C| > 1$ is called a *conflict* if $\bigcap_{t \in C} {}^\bullet t \neq \varnothing$.

Partial-order semantics. Figure 6 shows a *partial-order execution,* also called a *run,* of Σ_1 from Fig. 5. It is represented as a special acyclic and conflict-free net where a transition is now called *event*, which represents the occurrence of a transition of the system through a labelling, and a place is now called *condition,* which represents the occurrence of a token in a place. In contrast to a sequential execution, a partial-order execution does not represent a total order on the events but a partial order, called the *causal order.* While the event labelled with b_r is causally dependent on the event labelled with b_l, the events labelled with b_l and a_r are independent (not ordered). Causally independent events or conditions are also said to be *concurrent.* Each two different events or conditions are either causally dependent or concurrent in a partial-order execution. Each event of a partial-order execution has only finitely many causal predecessors. Note that, in contrast to sequential executions, there are infinite partial-order executions that can be extended, i.e., there are infinite executions that are proper prefixes of other executions. For example, in the run of Σ_1 where l becomes critical infinitely often while r remains quiet, the initial $quiet_r$-condition has no successor, hence the run can be extended by an occurrence of transition a_r.

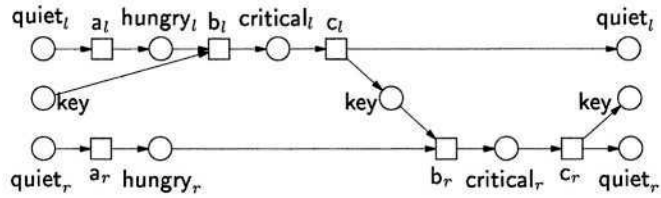


Fig. 6. A partial-order execution of Σ_1

A partial-order execution ρ represents a set of sequential executions, called *sequentializations* of ρ, which can be derived by arbitrarily increasing the causal order of ρ to a discrete total order. The partial-order counterpart of a computation tree is an *unfolding* [8]. Fig. 8 shows a system Σ_2 and Fig. 9 shows one of its

unfoldings (ignore the labels $\frac{1}{2}$ and % for now). Besides being causally dependent or concurrent, two events or conditions of an unfolding can also be *in (extended) conflict*, meaning that they cannot occur in the same execution together. In Fig. 9, both events labelled with e are in conflict with the event labelled with f. Two different events or conditions are either causally dependent, concurrent, or in conflict in an unfolding. There is a natural prefix-order on all unfoldings of a system with a maximal unfolding, which is unique up to isomorphism. For runs and unfoldings, "maximal" and "extension" are meant with respect to the prefix order in this paper. A partial-order execution is a conflict-free unfolding just as a sequential execution is a non-branching computation tree.

A *co-set* of an unfolding is a finite[2] set of pairwise concurrent conditions. A *cut* is a maximal co-set. Each cut of an unfolding of a system represents a reachable marking of that system through the labelling. A co-set is *reachable* from a cut if each condition of the co-set is causally dependent on or equal to some condition of the cut.

Randomized systems. Figure 7 shows an object flipping an n-sided coin. Each outcome of the coin flip is represented by a transition t_i which has a probability p_i associated with it such that $\sum_{i=1}^{n} p_i = 1$. Coin flips are internal events of objects, i.e., each transition t_i is *free,* that is $|{}^\bullet t_i| = 1$ (double slashed arcs in Fig. 7 and other figures below indicate the absence of arcs). The t_i are called *probabilistic transitions* and are graphically distinguished by the symbol %. A *randomized system* is a system that is equipped with such coin flips. It has, in general, *probabilistic conflicts,* which represent coin flips, as well as (purely) nondeterministic conflicts.

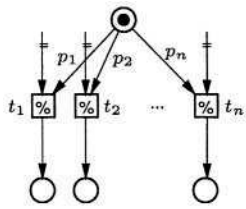


Fig. 7. An object flipping a coin

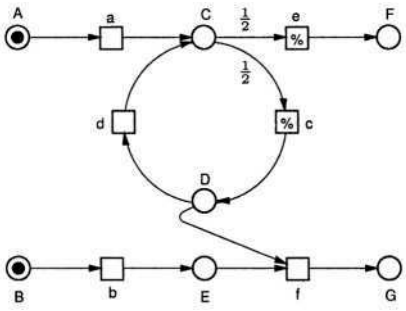


Fig. 8. A randomized system Σ_2

In the classical sequential semantics for randomized systems, there is a canonical probability space associated with each *probabilistic computation tree*, that is a computation tree in which each branching represents a coin flip, i.e., there is no (pure) nondeterminism in a probabilistic computation tree. The associated probability space measures properties of maximal sequential executions, i.e., paths, of that computation tree.

We use the partial-order counterpart of that semantics [10]. Two events of an

[2] We restrict our attention to finite co-sets and cuts because these correspond to reachable states of an unfolding.

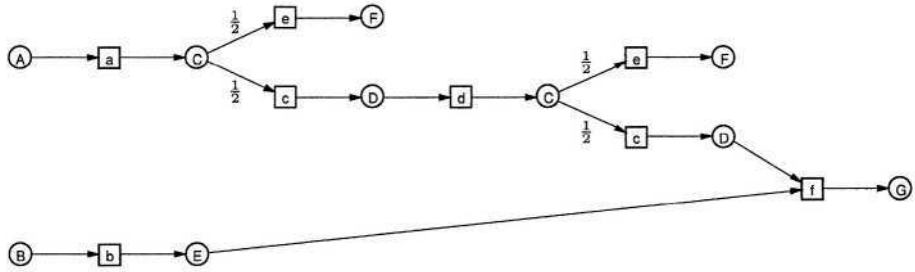


Fig. 9. A finite, maximal probabilistic unfolding of Σ_2

unfolding of a randomized system are said to be in *extended probabilistic conflict* if they causally depend on different outcomes of the same coin flip event. An unfolding is called *probabilistic* if each two events that are in extended conflict are also in extended probabilistic conflict. That means that each *reachable* conflict, that is, each conflict C such that ${}^\bullet C$ is a co-set, represents a coin flip, i.e., there is no pure nondeterminism in a probabilistic unfolding. There is a canonical probability space associated with each probabilistic unfolding, which measures properties of maximal runs of that unfolding. Fig. 8 shows a randomized system Σ_2 and Fig. 9 one of its maximal probabilistic unfoldings (Σ_2 has infinitely many, but only one of these is infinite). A maximal probabilistic unfolding can be constructed by starting with the initial marking and appending as long as possible to each co-set that enables a transition either a single non-probabilistic transition or a whole coin flip such that for each conflict C that is not part of a coin flip, ${}^\bullet C$ is not a co-set.

A *temporal property,* i.e., a set of runs, is said to be *1-valid* in a randomized system if it has probability 1 in the associated probability space of each *admissible* probabilistic unfolding where *admissible* will be defined by the liveness assumption of the particular model. The property "eventually F is marked" has probability $\frac{3}{4}$ in the probabilistic unfolding in Fig. 9. The property "eventually F or G is marked", which coincides with termination in Σ_2, is 1-valid in Σ_2 if admissibility is maximality. This partial-order semantics is more liberal than the classical sequential semantics, i.e., roughly speaking, if a property is 1-valid in all admissible computation trees then it is also 1-valid in all admissible probabilistic unfoldings but not necessarily vice versa [10]. A safety property that is 1-valid in a system is satisfied in every run of the system.

4 Progress

We present our first impossibility result in this section. We first formalize progress and define then which systems constitute a mutex solution.

A *progress system* consists of a system and a set of distinguished transitions $T^{\text{prog}} \subseteq T$, called *progress transitions.* A transition $t \in T \setminus T^{\text{prog}}$ is graphically distinguished by a grey shade. For a given coin flip of a randomized progress

system, we assume that either all or none of the transitions of the coin flip are progress transitions.

Let Q be a set of places of a system Σ and let ρ be a run of Σ. A co-set D of ρ is called a *Q-set* if it represents exactly Q through its labelling. Q is *persistent* in ρ if there is a Q-set D in ρ such that $D^\bullet = \varnothing$, i.e., it is a set of tokens that are never consumed in ρ. A run ρ *violates progress* with respect to t if ${}^\bullet t$ is persistent in ρ. A run ρ is *progressive* if there is no transition $t \in T^{\text{prog}}$ such that ρ violates progress with respect to t. An unfolding π is *progressive* if each run of π is progressive. Every (finite or infinite) unfolding can be extended to a progressive unfolding. Progress is maximality of partial-order executions and unfoldings defined with respect to single transitions.

We specify mutex in two steps (similar to Kindler and Walter's approach [7]). First we define *mutex structure* and then *mutex behaviour*. Although we consider only two agents in this section, we define mutex structure for a finite set A of agents to reuse the definition later.

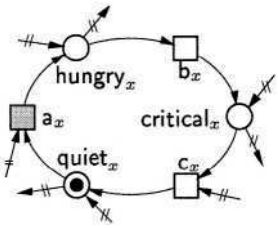


Fig. 10. Mutex structure Σ_x

Σ_x in Fig. 10 models a client x of a mutex algorithm and the double slashed arcs, indicating the absence of arcs, restrict the ways the client may interact with the mutex algorithm. A client may inform the mutex algorithm about its current state by outgoing arcs of a_x, b_x and c_x. The mutex algorithm may inform a client when it is allowed to enter the critical state by an ingoing arc to b_x. Note that a critical agent will always become quiet but a quiet agent may remain quiet forever, i.e., a mutex algorithm cannot rely on its clients eventually becoming hungry. Note also that a critical agent need not wait for a permission of the mutex algorithm to leave the critical state. This degree of clients' independence from the mutex algorithm is always assumed in the mutex problem. A progress system Σ has *mutex structure for A* if for each agent $x \in A$, (1) there is exactly one progress system Σ_x as in Fig. 10 that is a subsystem of Σ such that all Σ_x are pairwise disjoint and (2) the connection of Σ_x with the rest of Σ is restricted as Fig. 10 indicates. The formalization of this is straight-forward.

A randomized progress system Σ has *probabilistic mutex behaviour* for a set $\{l, r\}$ of two agents if the following two properties are 1-valid in each progressive probabilistic unfolding of Σ:

Both agents are never critical concurrently. (1)

Each hungry agent eventually becomes critical. (2)

Theorem 1. *There is no randomized progress system with mutex structure and probabilistic mutex behaviour for $\{l, r\}$.*

Proof: To derive a contradiction, let Σ be a randomized progress system with mutex structure and probabilistic mutex behavior. There is a progressive probabilistic unfolding π_1 where r is never hungry but l is always eventually hungry

with probability 1: start with the initial marking and let l become hungry and extend by progress, i.e., extend to a minimal progressive probabilistic unfolding. Let l become hungry everywhere (in every run) where possible and repeat infinitely often. It follows from the 1-validity of (2) that l is, with probability 1, infinitely often critical in π_1.

There is a progressive probabilistic unfolding π_2 that extends π_1 in which r becomes hungry: Let a_r occur and extend by progress. Each run of π_2 has a prefix that is a run of π_1. Note that r becomes hungry concurrently to all events in π_1 (cf. Fig. 11). The intuition for what follows is that r must get the permission to enter the critical section but that permission is already in permanent use with probability 1.

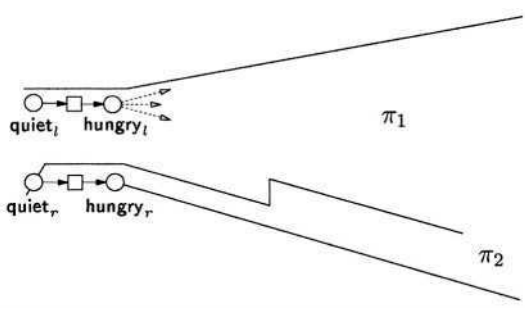


Fig. 11. Illustration for proof of Thm. 1

Claim: A run ρ of π_2 with infinitely many *critical*$_l$-conditions that are contained in π_1 has no *critical*$_r$-condition. *Proof:* By (1), a *critical*$_r$-condition cannot be concurrent to a *critical*$_l$-condition (remember that (1) is a safety property and hence its 1-validity is equivalent to its validity in all runs). Furthermore, it follows from the construction that a *critical*$_l$-condition that is contained in π_1 cannot be causally dependent on a *critical*$_r$-condition (which cannot belong to π_1). Therefore, a *critical*$_r$-condition of ρ must be causally dependent on all *critical*$_l$-conditions of ρ that are contained in π_1. This would contradict the fact that each condition has only finitely many causal predecessors, which proves the claim.

The set of maximal runs of π_1 that have infinitely many *critical*$_l$-conditions has probability 1 (in π_1). It follows that the set of maximal runs of π_2 that have infinitely many *critical*$_l$-conditions that are contained in π_1 has probability 1 (in π_2) since each maximal run of π_2 extends a maximal run of π_1. Therefore, the set of maximal runs of π_2 that have a *critical*$_r$-condition has probability 0. This contradicts (2) which completes the proof. □

The crucial point in the proof is that even an infinite run (likewise: unfolding) can be extended to a progressive run (progressive unfolding), because progress does not restrict conflict resolution at all.

It is evident from the proof of Thm. 1 that the theorem remains valid for mutex of more than two agents as well as for *n-mutex*, that is, if we replace (1) by "there are never more than n agents critical concurrently".

Thm. 1 implies that a naive idea to implement fairness with probability 1 by detecting conflicts and resolving them by flipping a coin cannot work.

5 Fairness

In this section, we formalize fairness and present our second impossibility result.

A *fairness system* consists of a progress system and a set of distinguished transitions $T^{\text{fair}} \subseteq T^{\text{prog}}$, called *fairness transitions*. For a randomized fairness system, we can assume $T^{\text{flip}} \cap T^{\text{fair}} = \varnothing$ without loss of generality[3]. Fairness uses a notion of concurrent conditions being available at the same time. Since there is no timing information in a partial-order execution, the conditions of a co-set D are only then available at the same time if they are either all persistent or they are all synchronized by the same event, i.e., if $e \in D^\bullet \Rightarrow D \subseteq {}^\bullet e$. Otherwise the execution may be timed in such a way that the conditions are not available at the same time (cf. also [11]).

A set Q of places of a system Σ is *strongly insistent* in a run ρ if for each cut C of ρ, there is a Q-set D that is reachable from C such that $e \in D^\bullet \Rightarrow D \subseteq {}^\bullet e$. A run ρ *violates fairness* with respect to a transition t of Σ if there is a set $Q \subseteq {}^\bullet t$ such that Q is persistent in ρ, ${}^\bullet t \setminus Q$ is strongly insistent in ρ, and t occurs only finitely many times in ρ. Fairness requires a set of objects to wait for the synchronization and the rest to be always eventually available together at the same time. If a run ρ violates fairness with respect to t then t is enabled infinitely often in each sequentialization of ρ. A run is *fair* if there is no transition $t \in T^{\text{fair}}$ such that ρ violates fairness with respect to t. A probabilistic unfolding π is *fair* if each run of π is fair. Each finite probabilistic unfolding can be extended to a progressive and fair probabilistic unfolding.

We consider now the dining philosophers problem [3], a generalization of Dijkstra's ring of five philosophers [4]. Let A be a finite set of agents and $N \subseteq A \times A$ be an irreflexive and symmetric relation. If $(x, y) \in N$ we call x and y *neighbours*. A system Σ has *generalized mutex behavior* for A and N if mutex is satisfied for each pair of neighbours, i.e.,

Neighbours are never critical concurrently, (3)

and the starvation freedom requirement (2) is satisfied in each admissible run.

Due to Thm. 1, fairness is needed for this problem and fairness is also sufficient [3]. However, consider now that agents may crash permanently. For our purposes, it suffices to allow that critical agents may crash, i.e., we do not require the transition c_x from Fig. 10 to be a progress transition anymore. We call the resulting notion of mutex structure a *crash-prone mutex structure*. We say that an agent x is *crashed* in a cut C of a run ρ if there is a $critical_x$-condition $b \in C$ such that $b^\bullet = \varnothing$. When an agent crashes—and crashes cannot be detected by other agents in an asynchronous system—we do not require neighbours to become critical again, i.e., we weaken the starvation-freedom (2) to

Each hungry agent eventually becomes critical unless itself
or one of its neighbours crashes. (4)

[3] Because a coin flip transition does not synchronize different objects and hence progress and fairness are equivalent for them.

In the case of a crashing agent, this still requires a hungry agent with at least distance 2 to the crashing agent in the neighbourhood graph to eventually become critical. A randomized fairness system Σ has *probabilistic crash-tolerant mutex behaviour* for A and N if (3) and (4) are 1-valid in each progressive and fair probabilistic unfolding of Σ.

Theorem 2. *Let A be a set of at least three agents. Then, there is a neighbourhood relation N on A such that there is no randomized fairness system with a crash-prone mutex structure for A and probabilistic crash-tolerant mutex behaviour for A and N.*

Fig. 12. Three agents in a neighbourhood relation

Proof. (sketch) Let A and N be as in Fig. 12. The proof is by contradiction. We show that a solution has a behaviour where agent b being hungry does not become critical with high probability. We present only the main idea here, the actual proof can be found in the full version of this paper. We will use a pattern of behaviour that is known as *conspiracy* [4,1]. A conspiracy occurs for example in Dijkstra's ring of five philosophers when the two neighbours of a particular philosopher eat alternatingly in such a fashion that the philosopher in the middle always eventually sees each of his two forks but never together at the same time. In our proof, agents a and c conspire against b by becoming, with high probability, alternatingly critical in such a way that there is always at least one of the two agents a and c in its critical state, hence b can never enter the critical state.

Starting in the initial marking, one can easily obtain a situation where a is critical and b and c are hungry. If we now freeze a, i.e., we pretend that a is crashed, then c will eventually become critical with high probability. Now, a can become quiet and then hungry again, resulting in a situation where a and b are hungry and c is critical. We can now extend in the same way with a and c in swapped roles. Iterating infinitely, we obtain a behaviour where, with high probability, always either a or c is critical, preventing b becoming critical with high probability. □

The crash-tolerant dining philosophers problem requires stronger synchronization than is provided by fairness, even in the presence of randomization. It requires synchronization of several independent objects that cannot wait for synchronization because of the required crash-tolerance. Such a guarantee of synchronization is provided by hyperfairness.

6 Hyperfairness

This section briefly introduces hyperfairness as a way to solve crash-tolerant generalized mutex. In contrast to fairness, hyperfairness does not *require* that all objects of a synchronization have to be ready at the same time—it *postulates* that all objects will eventually be ready at the same time provided that all objects return to their ready state independently of each other.

A set Q of places of a system Σ is *insistent* in a run ρ of Σ if for each cut C of ρ, there is a Q-set that is reachable from C. A run violates *hyperfairness* with respect to t if ${}^\bullet t$ is insistent in ρ and t occurs only finitely many times in ρ. A run violates hyperfairness with respect to t if and only if it has a sequentialization in which t is enabled infinitely often but occurs only finitely many times. Hyperfairness just excludes the *conspiracies* that we used in the proof of Thm 2.

Crash-tolerant generalized mutex can be solved through hyperfairness: We just have to place a key between each pair of neighbours as in Fig. 5. Also message-passing consensus can be solved with hyperfairness.

7 Conclusion

We presented two examples in this paper where the introduction of randomization did not help to solve the problems because it did not increase the ability of the model to synchronize independent objects. Note that our results establish that all depicted relationships in Fig. 1 are strict.

We obtained our results by using a model that is more abstract and a semantics that is more complex than those usually used for impossibility results in distributed computing. This is justified by the following benefits:

- Partial-order semantics simplifies the proof of Thm. 1.
- Thms. 1 and 2 are stronger than corresponding theorems in the classical sequential semantics, since the adversary associated with the partial-order semantics is strictly weaker than that associated with the classical sequential semantics [10].
- Our results are independent of the usual architecture assumptions such as message passing, read/write registers, etc. These can be modelled in our formalism.
- Our results are not based on the assumption that transitions and conditions are indivisible, since our semantics is robust under refinement of atomicity [11]. In classical models, it is often assumed that certain operations on shared objects are indivisible.

References

[1] P. C. Attie, N. Francez, and O. Grumberg. Fairness and hyperfairness in multiparty interactions. *Distributed Computing,* 6:245–254, 1993.

[2] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. 2nd PoDC,* pp. 27–30. ACM, 1983.

[3] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, 1988.

[4] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1:115–138, 1971.

[5] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. of the ACM,* 32(2):374–382, Apr. 1985.

[6] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM ToPLaS,* 5(3):356–380, July 1983.
[7] E. Kindler and R. Walter. Mutex needs fairness. *IPL,* 62:31–39, 1997.
[8] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science,* 13:85–108, 1981.
[9] W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In *Proceedings ICALP'97,* volume 1256 of *LNCS,* pages 538–548. Springer, 1997.
[10] H. Völzer. Randomized non-sequential processes. In *Proc. 12th CONCUR, LNCS* 2154, pp. 184–201. Springer, Aug. 2001.
[11] H. Völzer. Refinement-robust fairness. In *Proc. 13th CONCUR, LNCS* 2421, pp. 547–561. Springer, Aug. 2002.

A New Algorithm for Optimal Constraint Satisfaction and Its Implications

Ryan Williams*

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213
ryanw@cs.cmu.edu

Abstract. We present a novel method for exactly solving (in fact, counting solutions to) general constraint satisfaction optimization with at most two variables per constraint (*e.g.* MAX-2-CSP and MIN-2-CSP), which gives the first exponential improvement over the trivial algorithm. More precisely, the runtime bound has a constant factor improvement in the base of the exponent: the algorithm can count the number of optima in MAX-2-SAT and MAX-CUT instances in $O(m^3 2^{\omega n/3})$ time, where $\omega < 2.376$ is the matrix product exponent over a ring. When constraints have arbitrary weights, there is a $(1+\epsilon)$-approximation with roughly the same runtime, modulo polynomial factors. Our construction shows that improvement in the runtime exponent of either *k-clique* solution (even when $k = 3$) or *matrix multiplication over GF(2)* would improve the runtime exponent for solving 2-CSP optimization.

The overall approach also yields connections between the complexity of some (polynomial time) high dimensional search problems and some NP-hard problems. For example, if there are sufficiently faster algorithms for computing the diameter of n points in ℓ_1, then there is an $(2-\epsilon)^n$ algorithm for MAX-LIN. These results may be construed as either lower bounds on the high-dimensional problems, or hope that better algorithms exist for the corresponding hard problems.

1 Introduction

The extent to which NP-hard problems are indeed hard to solve remains largely undetermined. For some problems, it intuitively seems that the best one can do is examine every candidate solution, but this intuition has been shown to fail in many scenarios. The fledgling development of improved exponential algorithms in recent times suggests that for many hard problems, something substantially faster than brute-force search can be done, even in the worst case. However, some fundamental problems have persistently eluded researchers from better algorithms. One popular example in the literature is MAX-2-SAT.

There has been notable theoretical interest in discovering a procedure for MAX-2-SAT running in $O((2-\epsilon)^n)$ steps on all instances, for some $\epsilon > 0$. Unlike

* Supported by the NSF ALADDIN Center (NSF Grant No. CCR-0122581) and an NSF Graduate Research Fellowship.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1227–1237, 2004.

other problems such as Vertex Cover and k-SAT, where a sophisticated analysis of branch-and-bound techniques (or random choice of assignments) sufficed for improving the naïve bounds (*e.g.* [22,19]), MAX-SAT has been surprisingly difficult to attack. Previous work has only been able to show either a $(2-\epsilon)^m$ time bound, where m is the number of clauses, or an approximation scheme running in $(2-\epsilon)^n$ [10,9] (but $\epsilon \to 0$ as the quality of the approximation goes to 1). Of course, the number of clauses can in general be far higher than the number of variables, so the $(2-\epsilon)^m$ bounds only improve the trivial bound on some instances. The current best worst-case bound for MAX-2-SAT explicitly in terms of m is $\tilde{O}(2^{m/5})$ [1], by [8] (so for $m/n > 5$, this is no better than 2^n). This result followed a line of previous papers on bounds in terms of m [11,18, 7,5,3,16]; a similar line has formed for MAX-CUT [27,15]. In terms of partial answers to the problem, [20] showed that when every variable occurs at most three times, MAX-2-SAT remains NP-complete, but has an $O(n3^{n/2})$ algorithm. While definite stepping stones, these results still appeared distant from an improved exponential algorithm. As a result, several researchers (*e.g.* [20,1,9,29]) explicitly proposed a $(2-\epsilon)^n$ algorithm for MAX-2-SAT (and/or MAX-CUT) as a benchmark open problem in the area.

1.1 Outline of Our Approach: Split and List

Most exact algorithms for NP-hard problems in the literature involve either a case analysis of a branch-and-bound strategy [8], repeated random choice of assignments [19], or local search [24]. Our design is a major departure from these approaches, resembling earlier algorithms from the 70's [13,25]. We *split* the set of n variables into k partitions (for $k \geq 3$) of (roughly) equal size, and *list* the $2^{n/k}$ variable assignments for each partition. From these $k2^{n/k}$ assignments, we build a graph with weights on its nodes and edges, arguing that a optimum weight k-clique in the graph corresponds to a optimum solution to the original instance. The weights are eliminated using a polynomial reduction, and a fast k-clique algorithm on undirected graphs yields the improvement over 2^n. To get a $(1+\epsilon)$-approximation when constraints have arbitrary weights, we can adapt results concerning approximate all pairs shortest paths [30] for our purposes.

Finally, we investigate the possibility of efficient split-and-list algorithms for more general problems such as SAT and MAX-LIN-2 (satisfying a maximum number of linear equations in 0-1 variables). In particular, we will demonstrate some connections between this question and problems in high dimensional geometry. For example, if a furthest pair out of n d-dimensional points in ℓ_1 norm can be found faster than its known solutions (say, in $O(poly(d) \cdot n^{2-\epsilon})$ time), then there exists a $(2-\epsilon)^n$ split-and-list algorithm for MAX-LIN-2.

1.2 Notation

Let $V = \{x_1, \ldots, x_n\}$ be a set of variables over (finite) domains $D_1, \ldots, D_n$, respectively. A *k-constraint* on V is defined as a function $c : D_1 \times \cdots \times D_n \to$

[1] The $\tilde{O}$ suppresses polynomial factors.

$\{0,1\}$, where c only depends on k variables of V (one might say c is a *k-junta*). For a k-constraint c, define $\mathsf{vars}(c) \subseteq V$ to be this k-set of variables. Partial assignments a to variables of V are given by a sequence of assignments $x_{i_1} := v_2, x_{i_2} := v_2, \ldots, x_{i_k} := v_k$, where $i_j \in [n]$ and $v_{i_j} \in D_{i_j}$. A partial assignment a *satisfies* a constraint c if $\mathsf{vars}(c)$ is a subset of the variables appearing in a, and $c(a) = 1$ (the restriction of c to the variables in a evaluates to 1, on the variable assignments given by a).

Given a set S, $S^{m \times n}$ is the set of $m \times n$ matrices with entries taken from S.

Throughout, ω refers to the smallest real number such that for all $\epsilon > 0$, matrix multiplication over a ring can be performed in $O(n^{\omega+\epsilon})$ time. We will discuss three types of matrix product in the paper; unless otherwise specified, the default is matrix product over the ring currently under discussion. The other two are the distance product $(\otimes_d)$ on $\mathbb{Z} \cup \{-\infty, \infty\}$, and Boolean matrix product $(\otimes_b)$ on 0-1 matrices. Let $A, B \in (\mathbb{Z} \cup \{-\infty, \infty\})^{n \times n}$. $A \otimes_d B$ is matrix product over the (min, +)-semiring; that is, the usual + in matrix product is replaced with the min operator, and $\times$ is replaced with addition. When A and B are 0-1 matrices, the Boolean product $\otimes_b$ replaces + with $\vee$ (OR) and $\times$ with $\wedge$(AND).

2 Fast k-Clique Detecting and Counting

We briefly review an algorithm [17] for detecting if a graph has a k-clique in less than n^k steps.

Theorem 1. *([17]) Let $r \in \mathbb{Z}^+$. Then $3r$-clique on undirected graphs is solvable in $O(n^{\omega r})$ time.*

Proof. First consider $k = 3$. Given $G = (V, E)$ with $n = |V|$, let $A(G)$ be its adjacency matrix. Recall that $tr(M)$, the trace of a matrix M, is the sum of the diagonal entries. $tr(A(G)^3)$ is computable in two matrix multiplications, and it is easy to see that $tr(A(G)^3)$ is non-zero if and only if there is a triangle in G. For $3r$-cliques when $r > 1$, build a graph $G_r = (V_r, E_r)$ where V_r is the collection of all r-cliques in G, and $E_r = \{ \{c_1, c_2\} : c_1, c_2 \in V_r, c_1 \cup c_2$ is a $2r$-clique in $G\}$. Observe that each triangle in G_r corresponds to a unique $3r$-clique in G. Therefore $tr(A(G_r)^3) \neq 0$ if and only if there is a $3r$-clique in G, which is determined in $O(n^{\omega r})$ time. Finding an explicit $3r$-clique given that one exists may be done by using an $O(n^\omega)$ algorithm for finding witnesses to Boolean matrix product [2]; details omitted. □

In fact, the above approach may be used to *count* the number of k-cliques as well. Let $C_k(G)$ be the set of k-cliques in G, and G_r be as defined in the previous proof.

Proposition 1. $tr(A(G_r)^3) = 6|C_{3r}(G)|$.

Proof. In $tr(A(G)^3)$, each triangle $\{v_i, v_j, v_k\}$ is counted once for each vertex v (say, v_i) in the triangle, times the two paths traversing the triangle starting from that v (for v_i, they are $v_i \to v_j \to v_k \to v_i$ and $v_i \to v_k \to v_j \to v_i$). Similar reasoning shows that each $3r$-clique is counted six times in $tr(A(G_r)^3)$. □

3 Algorithm for 2-CSP Optimization

Let us explicitly define the problem we will tackle, in its full generality.

Problem COUNT-2-CSP:
Input: A set of m 1-constraints and 2-constraints C on n variables $x_1, \ldots, x_n$ with domains of size $d_1, \ldots, d_n$ (respectively), and a parameter $N \in \{0, 1, \ldots, m\}$.
Output: The number A of variable assignments ($0 \leq A \leq \prod_{i=1}^{n} d_i$) such that exactly N constraints of C are satisfied.

Let $\kappa(k)$ be such that the number of k-cliques on n-node undirected graphs can be found in $O(n^{\kappa(k)})$ time. (One may think of $\kappa(k)$ as the "k-clique exponent", analogous to the matrix multiplication exponent ω. But note also that k and $\kappa(k)$ may be functions of n.) For simplicity, let us assume that $|d_i|$ is the same for all $i = 1, \ldots, n$, and is equal to d.

Theorem 2. *Let $k(n) \geq 3$ be monotone non-decreasing and time-constructible. Then* COUNT-2-CSP *is in $O\left(\binom{N+\binom{k(n)}{2}-1}{\binom{k(n)}{2}-1}[k(n)d^{n/k(n)}]^{\kappa(k(n))}\right)$ time, where m is the number of constraints, n is the number of variables, and d is the domain size.*

Corollary 1. *The number of optima for* MAX-2-SAT *and* MAX-CUT *instances can be determined in $O(m^3 1.732^n)$ time, and an optimal assignment can be found in $O(nm^3 1.732^n)$ time.*

Proof of Corollary 1. Set $d = 2$ and $k = 3$. Search for the largest $N \in [m]$ such that the number of assignments a satisfying N constraints is non-zero and return a. This incurs an extra $O(m)$ factor. An explicit assignment can be found using self-reducibility, increasing the runtime by an $O(n)$ factor. □

Proof of Theorem 2. We reduce the problem to counting k-cliques in a large graph. Assume w.l.o.g. that n is divisible by k. Let C be a given instance. Arbitrarily partition the n variables of C into sets $P_1, P_2, \ldots, P_k$ with n/k variables each. For each P_i, make a list L_i of all $d^{n/k}$ assignments to the variables of P_i.

Step 1: Delegating responsibility.
Certain partitions (or pairs of partitions) will be responsible for satisfying certain constraints of C. Let $[k] = \{1, \ldots, k\}$, and $\binom{k}{2}$ denote the collection of 2-sets from $[k]$. We define a *responsibility map* $r : C \to ([k] \cup \binom{k}{2})$ from constraints to partitions and 2-sets of partitions: $r(c) := i \in [k]$, if $\mathsf{vars}(c) \subseteq P_i$, and $r(c) := \{i, j\} \in \binom{k}{2}$, if $\mathsf{vars}(c) \cap P_i \neq \varnothing$ and $\mathsf{vars}(c) \cap P_j \neq \varnothing$. Observe that r is well-defined assuming c is dependent on at most two variables ($|\mathsf{vars}(c)| \leq 2$).

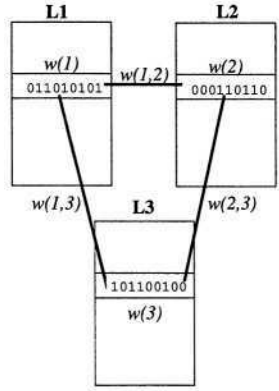
Step 2: Weighting accordingly.
Next, we will consider the $L_1, \ldots, L_k$ as a weighted k-partite complete graph $G = (V, E)$, having $d^{n/k}$ nodes per partition. For a vertex $v \in L_i$, let a_v denote

the partial assignment to which v refers. For a partial assignment a, $c(a)$ is the value of c when a is assigned to its variables. The weight function w for G is on nodes and edges of G, and is defined:

- $w(v) := |\{r(c) = i \ : \ c \in C, v \in L_i, c(a_v) = 1\}|$,
- $w(\{u,v\}) := |\{r(c) = \{i,j\} \ : \ c \in C, u \in L_i, v \in L_j, c(a_u \ a_v) = 1\}|$.

(Assuming the variables of $\mathbf{vars}(c)$ are assigned in a, $c(a)$ is well-defined; this is the case, by definition of r.) In general, $w(t)$ tells the number of $c \in C$ for which (a) t is in a partition/pair of partitions responsible for c, and (b) the partial assignment that t denotes satisfies c.[2]

Let $K = \{v_1, \ldots, v_k\}$ be a k-clique in G. Define $w(K) := \sum_{i=1}^{k} w(v_k) + \sum_{\{i,j\} \in \binom{k}{2}} w(\{v_i, v_j\})$, *i.e.* the weight of all nodes and edges involved in K. When $k = 3$ and $d = 2$, G resembles the picture below, for $1 \in L_1$, $2 \in L_2$, $3 \in L_3$.



Claim. The number of k-cliques of weight N in G is equal to the number of assignments satisfying exactly N constraints in C.

Proof. Let a be an assignment to all n variables of C, and suppose a satisfies exactly N constraints of C. Clearly, there exist unique $v_i \in L_i$ for $i = 1, \ldots, k$ such that $a = a_{v_1} a_{v_2} \cdots a_{v_k}$, *i.e.* a corresponds to a unique clique $K_a = \{v_1, \ldots, v_k\}$ in G. We have that $w(K_a)$ equals $\sum_{i=1}^{k} |\{r(c) = i : v \in L_i, c(a_v) = 1\}| + \sum_{\{i,j\} \in \binom{k}{2}} |\{r(c) = \{i,j\} : u \in L_i, v \in L_j, c(a_u \ a_v) = 1\}|$. That is, $w(K_a)$ counts the number of $c \in C$ that are satisfied by a, such that either $r(c) = i \in [k]$ for some i, or $r(c) = \{i,j\} \in \binom{k}{2}$ for some i, j. But as we argued above, $r(c)$ is well-defined over all $c \in C$, therefore $w(K_a)$ is precisely the number of constraints satisfied by a. As there is a 1-to-1 correspondence between k-cliques in G and assignments to C, and as k-cliques with N weight correspond to assignments satisfying N constraints, the claim is proven. □

[2] **Example.** For MAX-CUT, the formulation is especially simple: the $v \in L_i$ denote all $2^{n/k}$ possible cuts with a distinct "left" and "right" side, on the subgraph of n/k vertices P_i, $w(v)$ is the number of edges crossing the "sub-cut" defined by v, and $w(\{u,v\})$ is the number of edges crossing from one side (say, "left") of u to the opposite side (say, "right") of v.

Step 3: Reduction from weighted to unweighted graphs.
There is a slight problem: we want to count k-cliques in the above, but the above method for counting only works on unweighted graphs. We can remove this difficulty and tack on a multiplicative factor that is polynomial in $N \leq m$ but exponential in k. Consider the $\binom{k}{2}$-tuples $(i_{1,2}, i_{1,3}, \ldots, i_{k-1,k})$ where $i_{j,l} \in [N]$, and $i_{1,2} + i_{1,3} + \cdots + i_{k-1,k} = N$. For each tuple, construct a unweighted graph $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ where edges are placed according to the rules:

- If $j = 1$ and $l = 2$, put $\{v_1, v_2\}$ in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ iff $w(v_1) + w(v_2) + w(\{v_1, v_2\}) = i_{1,2}$,
- If $j = 1$ and $l > 2$, put $\{v_1, v_l\}$ in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ iff $w(v_l) + w(\{v_j, v_l\}) = i_{1,l}$,
- If $j > 1$, put $\{v_j, v_l\}$ in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ iff $w(\{v_j, v_l\}) = i_{j,l}$.

(Note $w(v_j), w(\{v_j, v_l\}) \in [m]$;this bounds the possible $i_{j,l}$values.) Then, for each k-clique $K = \{v_1, \ldots, v_k\}$ in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ (it takes $O([kd^{n/k}]^{\kappa(k)})$ time to count them), a short verification shows that N equals $w(K)$. That is, each k-clique counted in $G_{(i_{1,2},i_{1,3},\ldots,i_{k-1,k})}$ is a k-clique of weight N in G. Moreover, each distinct $G_{(i_{1,2},\ldots,i_{k-1,k})}$ represents a distinct collection of weight-N cliques in G. Hence the total sum of k-clique counts over all $G_{(i_{1,2},\ldots,i_{k-1,k})}$ is the number of weight-N k-cliques in G. The possible $\binom{k}{2}$-tuples correspond to all non-negative integral solutions to $x_1 + x_2 + \cdots + x_{\binom{k}{2}} = N$, which is $\binom{N+\binom{k}{2}-1}{\binom{k}{2}-1}$. A list of these solutions may be formed in such a way that each solution appears exactly once, with $O(1)$ (amortized) time to generate each one [23]. □

4 General Remarks

Out of space considerations, we have omitted proofs in the following subsections. For details, see [28].

4.1 On Triangles and Matrix Multiplication

The current $O(n^{3-\epsilon})$ matrix multiplication algorithms only begin to outperform Gaussian elimination (in practice) for very large cases. This coincides nicely with the fact that the size of our matrices are exponential in n. Still, it would be quite desirable to find a more *practical* algorithm for detecting triangles in $O(n^{3-\epsilon})$ time, but this has been an open problem since the 70's [14]. We can in fact show that if one only wishes to detect a k-clique quickly, it suffices to matrix multiply quickly *over GF(2).* To us, this gives some hope that triangles can be found more quickly, as GF(2) is the "simplest" possible field and matrix multiplication could potentially be easier over it.

Theorem 3. *If $n \times n$ matrices can be multiplied over GF(2) in $O(n^c)$ time, then there is a randomized algorithm for detecting if a graph has a triangle, running in $O(n^c \log n)$ time and succeeding with high probability.*

4.2 The Arbitrary Weight Case

We may also employ our main algorithm to get a $(1+\epsilon)$-approximation to MAX-2-CSP and MIN-2-CSP with arbitrary weights on the constraints. The approximation will have similar runtime to the exact algorithm in the unweighted case. Formally, the arbitrary weight problem is:

Problem OPT-WEIGHT-2-CSP:
Input: A 2-CSP instance with $C = \{c_1, \ldots, c_m\}$, n variables, and weight function $w : C \to \mathbb{Z}^+$.
Output: An assignment a such that $\sum_{i \in \{j \,:\, c_j(a)=1\}} w(c_i)$ is minimal/maximal.

If the constraints have weights describable in $O(\log m)$ bits, we could simply modify the above algorithm (where the weight of an assignment node is now the sum of weights of clauses) and get runtime $O(poly(m) \cdot 1.732^n)$. When weights are independent of m and n, it is possible to use an approximate all-pairs shortest paths algorithm of [30] to get a $(1+\epsilon)$-approximation in roughly $O(nm^3 1.732^n)$ time (setting $k = 3$ in Theorem 2). Recall $\otimes_d$ (defined earlier) is the distance product on matrices. If A is the adjacency matrix of a weighted (on edges) graph G, then $\min_{i=1}^{n}(A \otimes_d A \otimes_d A)[i,i]$ gives the length of a smallest triangle in G (and is 0 if there is no triangle in G). Say that C is an a-approximation to D iff for all i, j, $C[i,j] \leq D[i,j] \leq aC[i,j]$. Then the following theorem implies an efficient $(1+\epsilon)$-approximation to OPT-WEIGHT-2-CSP.

Theorem 4. *([30]) Let $A, B \in (\mathbb{Z} \cup \{-\infty, \infty\})^{n \times n}$. $A \otimes_d B$ has a $(1+\epsilon)$-approximation computable in $O((n^\omega/\epsilon)\log(W/\epsilon))$ time, where*
$W = \max\{A[i,j], B[i,j] \;:\; i,j \in [n]\}$.

5 Further Directions

Is it possible to solve general problems like SAT much faster than the trivial algorithm, using a "split-and-list" method akin to the above? Depending on one's outlook, the results below may be viewed as lower bounds on solving various high-dimensional problems, or they may be promising signs that much better algorithms exist for general NP-complete problems, using split-and-list methods.

5.1 Cooperative Subset Queries and SAT

Usually in query problems, one considers a database D of objects, and an adversarial list of queries $q_1, \ldots, q_k$ about D. Such a model often leads to very difficult problems, in particular the *subset query problem:* given a database D of sets $S_1, \ldots, S_k$ over a universe U, build a space-efficient data structure to answer (time-efficiently) queries of the form "is $q \in 2^U$ a subset of some $S_j \in D$?". This problem is a special case of the *partial matching problem;* that is, supporting queries with wildcards (*e.g.* "S**RCH") in a database of strings. Non-trivial algorithms exist [21,4], but all known solutions for the problem require either

$\Omega(|D|)$ query time (search the whole database) or $2^{\Omega(|U|)}$ space (store all possible subsets) in general. Our *cooperative* version of the subset query problem is the following:

Given two databases D_1 and D_2 of subsets over U, find $s_1 \in D_1$ and $s_2 \in D_2$ such that $s_1 \subseteq s_2$.

That is, we merely want to determine if one of the given queries has a *yes* answer. Intuitively, the cooperative version ought to be significantly easier than the general one. Of course, it admits a trivial $O(|U| \cdot |D_1| \cdot |D_2|)$ time algorithm, but if this solution can be significantly improved, then SAT in conjunctive normal form can be solved faster than the 2^n bound. Note the current best SAT algorithm is randomized, and runs in $2^{n-\Omega(n/\log n)}$ time [26].

Theorem 5. *Let f be time constructible. If the cooperative subset query problem with $d = |U|$ and $k = \max\{|D_1|, |D_2|\}$ is solvable in $\tilde{O}(f(d)k^{2-\epsilon})$ time, then CNF-SAT is in $\tilde{O}(f(m)2^{(1-\epsilon/2)n})$ time, where m is the number of clauses and n is the number of variables.*

Proof. (Sketch) Recall CNF-SAT is a constraint satisfaction problem where constraints are arbitrary *clauses, i.e.* OR's of negated and non-negated variables. Suppose the clauses are indexed $c_1, \ldots, c_m$. Partition variables into two sets P_1, P_2 of size $n/2$ each, making lists L_1, L_2 of the $2^{n/2}$ assignments. Associate with each $p \in L_1$ a set S_p, defined by $c_j \in S_p$ iff p does not satisfy c_j. For $p' \in L_2$, make a set $S_{p'}$, defined by $c_j \in S_{p'}$ iff p' satisfies c_j. Now determine if there is $S_p \in L_1$ and $S_{p'} \in L_2$ whereby $S_p \subseteq S_{p'}$. Then the set of clauses is satisfiable iff the cooperative subset query instance has a "yes" answer. □

5.2 Furthest Pair of Points in ℓ_1 and MAX-LIN

Recall that the ℓ_1 distance between two d-dimensional points $(x_1, \ldots, x_d)$ and $(y_1, \ldots, y_d)$ is $|x - y|_1 = \sum_{i=1}^{d} |x_i - y_i|$. A classic high dimensional geometry problem is ℓ_1-Furthest-Pair, or ℓ_1-Diameter: given a set $S \subseteq \mathbb{R}^d$ of n points, find a pair $x, y \in S$ for which $|x-y|_1$ is maximized. (It may be seen as a cooperative version of a problem where, given a query, one reports points furthest from it.) Beyond the naïve $O(dn^2)$ solution, an isometric embedding from ℓ_1 in d dimensions to ℓ_∞ in 2^d dimensions yields a $O(2^d n)$ time algorithm. We will prove if ℓ_1-Furthest-Pair can be solved in (for example) $O(2^{o(d)}n^{2-\epsilon})$, then there is a better algorithm for MAX-LIN-2. Recall the MAX-LIN-2 problem is, given a system of m linear equations over n variables in GF(2), to find a variable assignment that maximizes the number of equations satisfied.

Theorem 6. *If ℓ_1-Furthest-Pair (of n points in d-dimensions) is in $O(f(d)n^{2-\epsilon})$ time, then MAX-LIN-2 is in $O(f(m+1)2^{(1-\epsilon/2)n})$ time.*

Proof. Rewrite each GF(2) linear equation as a constraint: an equation $\sum_i a_i x_i = d$ becomes $c(x_1, \ldots, x_n) = \sum_i a_i x_i + d + 1$. Let $c_1, \ldots, c_m$ be the resulting constraints. As before, split the variables into two $n/2$ sets P_1, P_2,

and have two lists of assignments L_1 and L_2. For each c_j, let $c_j|_{P_i}$ be the *restriction* of c_j to the variables of P_i. In the case where $+1$ appears in c_j, add $+1$ to $c_j|_{P_1}$. For example, suppose we had the partitions $P_1 = \{x_1, x_2\}$ and $P_2 = \{x_3, x_4\}$; if $c = \sum_{i=1}^{4} x_i + 1$, then $c|_{P_1} = x_1 + x_2 + 1$, $c|_{P_2} = x_3 + x_4$. Clearly, $c_j(x_1, \ldots, x_n) = c_j|_{P_1}(x_1, \ldots, x_{n/2}) + c_j|_{P_2}(x_{n/2+1}, \ldots, x_n)$. For $i \in \{1, 2\}$ and for all assignments $a \in L_i$, make an $(m+1)$-dimensional point

$$v_a = (\ c_1|_{P_i}(a),\ \ldots,\ c_m|_{P_i}(a), 2m \cdot i\ )$$

Let v_a and $v_{a'}$ be a furthest pair of points out of these, with respect to ℓ_1 distance.

Claim: The assignment $(x_1, \ldots, x_n) = (a, a')$ satisfies a maximum number of constraints in the original instance.

Proof. (Sketch) Use the fact that addition in GF(2) is subtraction in GF(2). The last component of v_a ensures that the furthest pair of points consists of one point from L_1 and one from L_2. □

6 Conclusion

We have presented the first improved exponential algorithm (in n) for solving and counting solutions to 2-constraint optimization. Our techniques are general enough to be possibly employed on a variety of problems. We have also established interesting connections between the complexity of some efficiently solvable problems and some hard problems (matrix multiplication and counting 2-CSP optima, furthest pair of points and MAX-LIN-2, subset queries and SAT). Several pointed questions are left open by our work.

- How does one extend our results for k-CSPs, when $k \geq 3$? A straightforward generalization to 3-CSPs results in a weighted hypergraph of edges and 3-edges. It is conjectured that matrix multiplication can be done in $O(n^{2+o(1)})$ time, and in our investigation of 3-CSPs, it appears a $2^{3n/4}$ bound might be possible. We therefore conjecture that for all $k \geq 2$, MAX-k-SAT is in $\tilde{O}(2^{n(1-\frac{1}{k+1})})$ time.
- Are there fast algorithms for 2-CSP optimization using only polynomial space?
- Is there a randomized k-clique detection algorithm running in $O(n^{3-\epsilon})$? (Is there merely a good one that doesn't use matrix multiplication?)
- What other problems are amenable to the split-and-list approach?

Acknowledgements. I am indebted to my advisor Manuel Blum, to Nikhil Bansal, and to Virginia Vassilevska, for enlightening conversations on this work.

References

1. J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: a parameterized point of view. *Discrete Mathematics* 229:3–27, Elsevier, 2001.
2. N. Alon and M. Naor. Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions. *Algorithmica* 16:434–449, 1996.

3. N. Bansal and V. Raman. Upper bounds for Max-Sat: Further Improved. In *Proceedings of ISAAC,* Springer LNCS 1741:247–258, 1999.
4. M. Charikar, P. Indyk, and R. Panigrahy. New Algorithms for Subset Query, Partial Match, Orthogonal Range Searching, and Related Problems. In *Proceedings of ICALP,* Springer LNCS 2380:451–462, 2002.
5. J. Chen and I. Kanj. *Improved Exact Algorithms for MAX-SAT.* In *Proceedings of LATIN,* Springer LNCS 2286:341–355, 2002.
6. D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. *JSC* 9(3):251–280, 1990.
7. J. Gramm and R. Niedermeier. Faster exact solutions for Max2Sat. In *Proceedings of CIAC,* Springer LNCS 1767:174–186, 2000.
8. J. Gramm, E.A. Hirsch, R. Niedermeier and P. Rossmanith. Worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. *Discrete Applied Mathematics* 130(2):139–155, 2003.
9. E. Dantsin, M. Gavrilovich, E. A. Hirsch, and B. Konev. MAX-SAT approximation beyond the limits of polynomial-time approximation. *Annals of Pure and Applied Logic* 113(1-3): 81–94, 2001.
10. E. A. Hirsch, Worst-case study of local search for MAX-k-SAT. *Discrete Applied Mathematics* 130(2): 173-184, 2003.
11. E. A. Hirsch. A $2^{m/4}$-time Algorithm for MAX-2-SAT: Corrected Version. *Electronic Colloquium on Computational Complexity* Report TR99-036, 2000.
12. T. Hofmeister, U. Schöning, R. Schuler, O. Watanabe. A probabilistic 3-SAT algorithm further improved. In *Proceedings of STACS,* Springer LNCS 2285:192–202, 2002.
13. E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *JACM* 21: 277–292, 1974.
14. A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing,* 7(4): 413–423, 1978.
15. A. S. Kulikov and S. S. Fedin. A $2^{|E|/4}$-time Algorithm for MAX-CUT. *Zapiski nauchnyh seminarov POMI* No.293, 129–138, 2002.
16. M. Mahajan and V. Raman. Parameterizing above Guaranteed Values: MAXSAT and MAXCUT. *J. Algorithms* 31(2): 335-354, 1999.
17. J. Nesetril and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae,* 26(2): 415–419, 1985.
18. R. Niedermeier and P. Rossmanith. New upper bounds for maximum satisfiability. *J. Algorithms* 26:63–88, 2000.
19. R. Paturi, P. Pudlak, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the 39th IEEE FOCS,* 628-637, 1998.
20. V. Raman, B. Ravikumar, and S. Srinivasa Rao. A Simplified NP-Complete MAXSAT problem. *Information Processing Letters* 65:1–6, 1998.
21. R. Rivest. Partial match retrieval algorithms. SIAM J. on Computing, 5:19–50, 1976.
22. M. Robson. Algorithms for maximum independent sets. *J. Algorithms,* 7(3):425–440, 1986.
23. F. Ruskey. Simple combinatorial Gray codes constructed by reversing sublists. In *Proceedings of International Symposium on Algorithms and Computation,* Springer LNCS 762:201–208, 1993.
24. U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th IEEE FOCS,* 410–414, 1999.
25. R. Schroeppel and A. Shamir. A $T=O(2^{n/2})$, $S=O(2^{n/4})$ Algorithm for Certain NP-Complete Problems. *SIAM J. Comput.* 10(3): 456–464, 1981.

26. R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. Accepted in *J. Algorithms,* 2003.
27. A. Scott and G. Sorkin. Faster Algorithms for MAX CUT and MAX CSP, with Polynomial Expected Time for Sparse Instances. In *Proceedings of RANDOM-APPROX 2003,* Springer LNCS 2764:382–395, 2003.
28. R. Williams. A new algorithm for optimal constraint satisfaction and its implications. *Electronic Colloquium on Computational Complexity,* Report TR04-032, 2004.
29. G.J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization - Eureka! You shrink!*, Springer LNCS 2570:185–207, 2003.
30. U. Zwick. All Pairs Shortest Paths using bridging sets and rectangular matrix multiplication. *JACM* 49(3):289–317, May 2002.

On the Power of Ambainis's Lower Bounds*

Shengyu Zhang

Computer Science Department,
Princeton University,
Princeton, NJ 08544, USA.
szhang@cs.princeton.edu

Abstract. The polynomial method and the Ambainis's lower bound (or *Alb,* for short) method are two main quantum lower bound techniques. While recently Ambainis showed that the polynomial method is not tight, the present paper aims at studying the power and limitation of *Alb*'s. We first use known *Alb*'s to derive $\Omega(n^{1.5})$ lower bounds for BIPARTITENESS, BIPARTITENESS MATCHING and GRAPH MATCHING, in which the lower bound for BIPARTITENESS improves the previous $\Omega(n)$ one. We then show that all the three known Ambainis's lower bounds have a limitation $\sqrt{N \cdot \min\{C_0(f), C_1(f)\}}$, where $C_0(f)$ and $C_1(f)$ are the 0- and 1-certificate complexity, respectively. This implies that for many problems such as TRIANGLE, k-CLIQUE, BIPARTITENESS and BIPARTITE/GRAPH MATCHING which draw wide interest and whose quantum query complexities are still open, the best known lower bounds cannot be further improved by using Ambainis's techniques. Another consequence is that all the Ambainis's lower bounds are not tight. For total functions, this upper bound for *Alb*'s can be further improved to $\min\{\sqrt{C_0(f)C_1(f)}, \sqrt{N \cdot CI(f)}\}$, where $CI(f)$ is the size of max intersection of a 0-and a 1-certificate set. Again this implies that *Alb*'s cannot improve the best known lower bound for some specific problems such as AND-OR TREE, whose precise quantum query complexity is still open. Finally, we generalize the three known *Alb*'s and give a new *Alb* style lower bound method, which may be easier to use for some problems.

1 Introduction

Quantum computing has received a great deal of attention in the last decade because of the potentially high speedup over the classical computation. Among others, query model is extensively studied, partly because it is a natural quantum analog of classical decision tree complexity, and partly because many known quantum algorithms fall into this framework [19,13,14,16,20,29]. In the query model, the input is accessed by querying an oracle, and the goal is to minimize the number of queries made. We are most interested in double-side bounded-error computation, where the output is correct with probability at least 2/3 for all inputs. We use $Q_2(f)$ to denote minimal number of queries for computing f

* This research was supported in part by NSF grant CCR-0310466.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1238–1250, 2004.

with double sided bound-error. For more details on quantum query model, we refer to [6,15] as excellent surveys.

Two main lower bound techniques for $Q_2(f)$ are the polynomial method by Beals, Buhrman, Cleve, Mosca and de Wolf [11] and Ambainis's lower bounds [5,4], the latter of which is also called quantum adversary method. Many lower bounds have recently been proven by applying the polynomial method [1,11,23, 25,28] and Ambainis's lower bounds[2,5,4,18,32]. Recently, Aaronson even uses Ambainis's lower bound technique to achieve lower bounds for some classical algorithms [2]. Given the usefulness of the two methods, it is interesting to know how tight they are. In a recent work [4], Ambainis proves that polynomial method is not tight, by showing a function with polynomial degree M and quantum query complexity $\Omega(M^{1.321\cdots})$. So a natural question is the power of Ambainis's lower bounds. We show that all known Ambainis's lower bounds are not tight either, among other results.

There are several known versions of Ambainis's lower bounds, among which the three Ambainis's theorems are widely used partly because they have simple forms and are thus easy to use. The first two *Alb*'s are given in [5] as follows.

Theorem 1 (Ambainis, [5]). *Let* $f : \{0,1\}^N \to \{0,1\}$ *be a function and X, Y be two sets of inputs* s.t. $f(x) \neq f(y)$ *if* $x \in X$ *and* $y \in Y$. *Let* $R \subseteq X \times Y$ *be a relation* s.t.

1. $\forall x \in X$, *there are at least* m *different* $y \in Y$ s.t. $(x,y) \in R$.
2. $\forall y \in Y$, *there are at least* m' *different* $x \in X$ s.t. $(x,y) \in R$.
3. $\forall x \in X, \forall i \in [N]$, *there are at most* l *different* $y \in Y$ s.t. $(x,y) \in R$, $x_i \neq y_i$.
4. $\forall y \in Y, \forall i \in [N]$, *there are at most* l' *different* $x \in X$ s.t. $(x,y) \in R$, $x_i \neq y_i$.

Then $Q_2(f) = \Omega(\sqrt{\frac{mm'}{ll'}})$.

Theorem 2 (Ambainis, [5]). *Let* $f : I^N \to \{0,1\}$ *be a Boolean function where I is a finite set, and X, Y be two sets of inputs* s.t. $f(x) \neq f(y)$ *if* $x \in X$ *and* $y \in Y$. *Let* $R \subseteq X \times Y$ *satisfy 1 and 2 in Theorem 1. Denote*

$$l_{x,i} = |\{y : (x,y) \in R, x_i \neq y_i\}|, \qquad l_{y,i} = |\{x : (x,y) \in R, x_i \neq y_i\}|$$

$$l_{max} = \max_{x,y,i:(x,y)\in R, i\in[N], x_i\neq y_i} l_{x,i} l_{y,i}.$$

Then $Q_2(f) = \Omega(\sqrt{\frac{mm'}{l_{max}}})$.

Obviously, Theorem 2 generalizes Theorem 1. In [4], Ambainis gives another (weighted) approach to generalize Theorem 1. We restate it in a form similar to Theorem 1.

Definition 1. *Let* $f : I^N \to \{0,1\}$ *be a Boolean function where I is a finite set. Let X, Y be two sets of inputs* s.t. $f(x) \neq f(y)$ *if* $x \in X$ *and* $y \in Y$. *Let* $R \subseteq X \times Y$ *be a relation. A weight scheme for X, Y, R consists three weight functions* $w(x,y) > 0$, $u(x,y,i) > 0$ *and* $v(x,y,i) > 0$ *satisfying*

$$u(x,y,i)v(x,y,i) \geq w^2(x,y) \tag{1}$$

for all $(x,y) \in R$ *and* $i \in [N]$ *with* $x_i \neq y_i$. *We further denote*

$$w_x = \sum_{y:(x,y)\in R} w(x,y), \qquad w_y = \sum_{x:(x,y)\in R} w(x,y)$$

$$u_{x,i} = \sum_{y:(x,y)\in R, x_i \neq y_i} u(x,y,i), \quad v_{y,i} = \sum_{x:(x,y)\in R, x_i \neq y_i} v(x,y,i).$$

Theorem 3 (Ambainis, [4]). *Let* $f : I^N \to \{0,1\}$ *where I is a finite set, and* $X \subseteq f^{-1}(0)$, $Y \subseteq f^{-1}(1)$ *and* $R \subseteq X \times Y$. *Let* w, u, v *be a weight scheme for X, Y, R. Then*

$$Q_2(f) = \Omega\left(\sqrt{\min_{x\in X, i\in[N]} \frac{w_x}{u_{x,i}} \cdot \min_{y\in Y, j\in[N]} \frac{w_y}{v_{y,j}}}\right)$$

Denote by $Alb_1(f)$, $Alb_2(f)$ and $Alb_3(f)$ the best lower bound for function f achieved by Theorem 1, 2 and 3, respectively[1]. Note that in the three *Alb*'s, there are many parameters (X, Y, R, u, v, w) to be set. By setting these parameters in an appropriate way, one can get good lower bounds for many problems. In particular, we consider the following three graph properties[2].

1. BIPARTITENESS: *Given an undirected graph, decide whether it is bipartite.*
2. GRAPH MATCHING: *Given an undirected graph, decide whether it has a perfect matching.*
3. BIPARTITE MATCHING: *Given an undirected bipartite graph, decide whether it has a perfect matching.*

We show by using Alb_2 that all these three graph properties have a $\Omega(n^{1.5})$ lower bound, where n is the number of vertices. For BIPARTITENESS, this improves the previous result of $\Omega(n)$ lower bound [21,17].

Since Alb_2 and Alb_3 generalizes Alb_1 in different ways, it is interesting to compare the their powers. It turns out that $Alb_2(f) \leq Alb_3(f)$.

However, even Alb_3 has a limitation: we show that $Alb_3(f)$ is no more than $\sqrt{N \cdot C_-(f)}$ where $C_-(f) = \min\{C_0(f), C_1(f)\}$ with $C_0(f)$ and $C_1(f)$ being the 0-and 1-certificate complexity of f, respectively. This has two immediate consequences. First, it gives a negative answer to the open problem whether Alb_2 or Alb_3 is tight, because for ELEMENT DISTINCTNESS, we know that $Q_2(f) = \Omega(N^{2/3})$ by Shi's result in [28], but $\sqrt{N \cdot C_-(f)}$ is only $\sqrt{2N}$.

Second, for some problems whose precise quantum query complexities are still unknown, our theorem implies that the best known lower bound cannot be further improved by using Ambainis's lower bound techniques, no matter how we choose the parameters in the *Alb* theorems. For example TRIANGLE/k-CLIQUE (k is constant) are the problems to decide whether an n-node graph contains a

[1] To make the later results more precise, we actually use $Alb_i(f)$ to denote the value inside the $\Omega()$ notation. For example, $Alb_1(f) = \max_{(X,Y,R)} \sqrt{\frac{mm'}{ll'}}$.

[2] In this paper, all the graph property problems are given by adjacency matrix input.

triangle/k-node clique. It is easy to get a $\Omega(n)$ lower bound for both of them. By our theorem, however, we know that this is the best possible by using Ambainis's lower bound techniques. Also the $\Omega(n^{1.5})$ lower bound for BIPARTITENESS, BIPARTITE MATCHING and GRAPH MATCHING cannot be further improved by *Alb*'s either, because $C_1(f) = O(n)$ for all of them.

If f is a total function, the above upper bound of *Alb*'s can be further tightened in two ways. The first one is $Alb_3(f) \leq \sqrt{N \cdot CI(f)}$, where $CI(f)$ is the the size of the largest intersection of a 0-certificate set and a 1-certificate set, so $CI(f) \leq C_-(f)\}$. The second approach leads to another result $Alb_3(f) \leq \sqrt{C_0(f)C_1(f)}$. Both the results imply that for AND-OR TREE, a problem whose quantum query complexity is still open [4], the current best $\Omega(\sqrt{N})$ lower bound [9] cannot be further improved by using Ambainis's lower bounds. The second result also give an positive answer to the open question whether $Alb_3(f) = O(\sqrt{C_0(f)C_1(f)})$.

Finally, it is natural to consider combining the different approaches that Alb_2 and Alb_3 use to generalize Alb_1, and get a further generalized one. Based on this idea, we give a new and more general lower bound theorem, which we call Alb_4. Compared with Alb_3, this may be easier to use.

Related Work

In the open problems part of [4], Ambainis mentions the $\sqrt{C_0(f)C_1(f)}$ limitation of Alb_1, and asks for new quantum lower bound techniques higher than $\sqrt{C_0(f)C_1(f)}$. However, it is not shown in [4] whether Alb_2 and Alb_3 are also bounded by the $\sqrt{C_0(f)C_1(f)}$ limitation for total function f, and actually even whether $Alb_2(f) = O(\sqrt{C_0(f)C_1(f)})$ was still open at the time, according to a private communication between Ambainis and us.

Szegedy independently shows in [30] that the quantum adversary method proposed in [10] cannot prove lower bounds higher than $\sqrt{N \cdot C_-(f)}$. Recently Szegedy also shows that $Alb_3(f) \leq \sqrt{N \cdot C_-(f)}$ and $Alb_3(f) \leq \sqrt{C_0(f)C_1(f)}$ for total functions in a different way [31]. In that paper, he also mentions (without proof) a very nice result that Alb_3 by Ambainis [4], Alb_4 in the present paper [33], and another quantum adversary method proposed in [10] are equivalent.

The theorem $Alb_3(f) \leq \sqrt{N \cdot C_-(f)}$ is also discovered by Laplante and Magniez by using Kolmogorov complexity in [21]. And the $\Omega(n^{1.5})$ lower bound for Matching is independently obtained by Berzina, Dubrovsky, Freivalds, Lace and Scegulnaja in [12].

2 Old Ambainis's Lower Bounds

In this section we first use Alb_2 to derive $\Omega(n^{1.5})$ lower bounds for BIPARTITENESS, BIPARTITE MATCHING and GRAPH MATCHING, then show that Alb_3 has actually at least the same power as Alb_2.

Theorem 4. *All the three graph properties* BIPARTITENESS, BIPARTITE MATCHING *and* GRAPH MATCHING *have* $Q_2(f) = \Omega(n^{1.5})$.

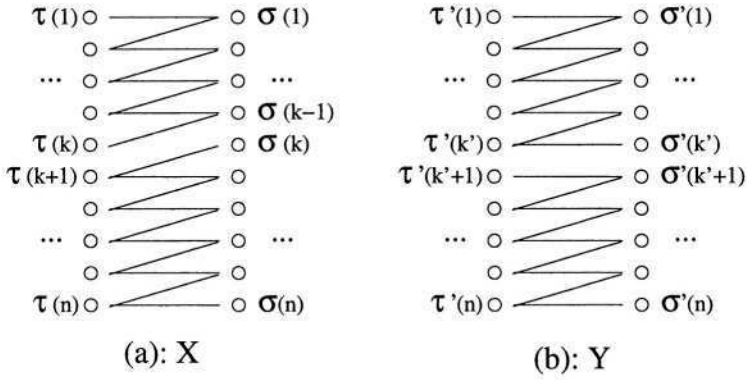


Fig. 1. X and Y

Proof. 1. BIPARTITENESS. Without loss of generality, we assume n is even, because otherwise we can use the following argument on arbitrary $n-1$ (out of total n) nodes and leave the n^{th} node isolated. Let *X, Y, R* to be exactly the same as the proof of $\Omega(n^{1.5})$ lower bound of GRAPH CONNECTIVITY by Durr, Mhalla and Lei [18]. Note that a graph is bipartite if and only if it contains no cycle with odd length, thus the graphs in X are all bipartite and the graph in Y are not. Then use the same arguments we as in [18] we can get $Alb_2(\text{BIPARTITENESS}) = \Omega(n^{1.5})$.

2. BIPARTITE MATCHING. Let X be the set of the bipartite graphs like Figure 1(a) where τ and σ are two permutations of $\{1,...,n\}$, and $\frac{n}{3} \le k \le \frac{2n}{3}$. Let Y be the set of the bipartite graphs like Figure 1(b), where τ' and σ' are two permutations of $\{1,...,n\}$, and also $\frac{n}{3} \le k' \le \frac{2n}{3}$. It is easy to see that all graphs in X have no perfect matching, while all graphs in Y have a perfect matching.

Let R be the set of all pairs of $(x,y) \in X \times Y$ as in Figure 2, where graph y is obtained from x by choosing two horizontal edges $(\tau(i),\sigma(i))$, $(\tau(j),\sigma(j))$, removing them, and adding two edges $(\tau(i),\sigma(j))$, $(\tau(j),\sigma(i))$.

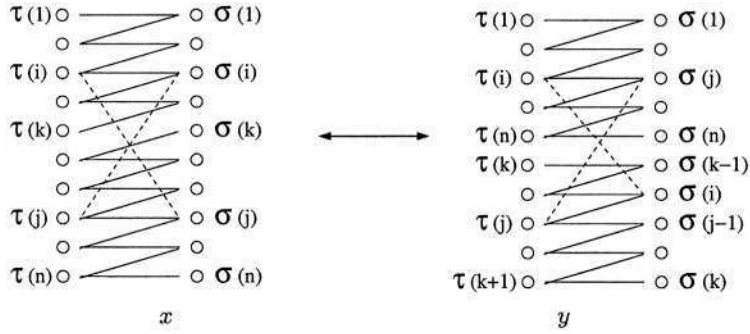


Fig. 2. $R \subseteq X \times Y$

Now it is not hard to calculate the m, m', l_{max} in Alb_2. For example, to get m we study x in two cases. When $\frac{n}{3} \leq k \leq \frac{n}{2}$, any edge $(\tau(i), \sigma(i))$ where $i \in [k - n/3, k]$ has at least $n/6$ choices for edge $(\tau(j), \sigma(j))$ because the only requirement for choosing is that $k' \in [n/3, 2n/3]$ and $k' = i+n-j$. The case when $\frac{n}{2} \leq k \leq \frac{2n}{3}$ can be handled symmetrically. Thus $m = \Theta(n^2)$. Same argument yields $m' = \Theta(n^2)$. Finally, for l_{max}, we note that if the edge $e = (\tau(i), \sigma(i))$ for some i, then $l_{x,e} = O(n)$ and $l_{y,e} = 1$; if the edge $e = (\tau(i), \sigma(j))$ for some i, j, then $l_{x,e} = 1$ and $l_{y,e} = O(n)$. For all other edges e, $l_{x,e} = l_{y,e} = 0$. Putting all cases together, we have $l_{max} = O(n)$. Thus by Theorem 2, we know that $Alb_2(\text{BIPARTITE MATCHING}) = \Omega(n^{1.5})$.

3. GRAPH MATCHING. This can be easily shown either by using the same (X, Y, R) as the proof for BIPARTITENESS, because a cycle with odd length has no matching, or by noting that BIPARTITE MATCHING is a special case of GRAPH MATCHING. □

It is interesting to note that we can also prove the above theorem by Alb_3. For example, for BIPARTITE MATCHING, we choose X, Y, R in the same way, and let $w(x, y) = 1$ for all $(x, y) \in R$. Let $u(x, y, e) = 1/\sqrt{n}$ if e is a horizonal edge $(\tau(i), \sigma(i))$ in x, and $u(x, y, e) = \sqrt{n}$ if $e = (\tau(i), \sigma(j))$ or $e = (\tau(j), \sigma(i))$ in x. Thus $u_{x,e} = \Theta(\sqrt{n})$ for all edge e, it is the same for $v_{y,e}$, thus $w_x/u_{x,e} = \Theta(n^{1.5}), w_y/v_{y,e} = \Theta(n^{1.5})$, and $Q_2(f) = \Omega(n^{1.5})$ by Alb_3.

This is not coincidence. It turns out that we can always show a lower bound by Alb_3 provided that it can be shown by Alb_2.

Theorem 5. $Alb_2(f) \leq Alb_3(f)$.

Proof. For any X, Y, R in Theorem 2, we set the weight functions in Theorem 3 as follows. Let $w(x, y) = 1$, $u(x, y, i) = \sqrt{l_{max}}/l_{x,i}$ and $v(x, y, i) = \sqrt{l_{max}}/l_{y,i}$. It is easy to check (1) in Definition 1, and calculate $u_{x,i} = \sqrt{l_{max}}, v_{y,i} = \sqrt{l_{max}}$. Thus, by denoting $m_x = |\{y : (x, y) \in R\}|$ and $m_y = |\{x : (x, y) \in R\}|$, we have

$$\min_{x,i} \frac{w_x}{u_{x,i}} \min_{y,i} \frac{w_y}{v_{y,i}} = \min_{x,i} \frac{m_x}{\sqrt{l_{max}}} \min_{y,i} \frac{m_y}{\sqrt{l_{max}}} = \frac{m}{\sqrt{l_{max}}} \frac{m'}{\sqrt{l_{max}}} = \frac{mm'}{l_{max}}.$$

□

3 Limitations of Ambainis's Lower Bounds

In this section, we show some bounds for the *Alb*'s in terms of certificate complexity. We consider Boolean functions.

Definition 2. *For an N-ary Boolean function $f : I^N \to \{0, 1\}$ and an input $x \in I^N$, a certificate set CS_x of f on x is a set of indices such that $f(x) = f(y)$ whenever $y_i = x_i$ for all $i \in CS_x$. The certificate complexity $C(f, x)$ of f on x is the size of a smallest certificate set of f on x. The b-certificate complexity of f is $C_b(f) = \max_{x:f(x)=b} C(f, x)$. The certificate complexity of f is $C(f) = \max\{C_0(f), C_1(f)\}$. We further denote $C_-(f) = \min\{C_0(f), C_1(f)\}$.*

3.1 A General Limitation for Ambainis's Lower Bounds

In this subsection, we give an upper bound for $Alb_3(f)$, which implies a limitation of all the three known Ambainis's lower bound techniques.

Theorem 6. *$Alb_3(f) \leq \sqrt{N \cdot C_-(f)}$, for any N-ary Boolean function f.*

Proof. Actually we prove a stronger result: for any (X, Y, R, u, v, w) as in Theorem 3,

$$\min_{(x,y)\in R, i\in[N]} \frac{w_x w_y}{u_{x,i} v_{y,i}} \leq NC_-(f). \tag{2}$$

With out loss of generality, we assume that $C_-(f) = C_0(f)$, and $X \subseteq f^{-1}(0)$ and $Y \subseteq f^{-1}(1)$. We can actually further assume that $R = X \times Y$, because otherwise we just let $R' = X \times Y$, and set new weight functions $u'(x,y,i), v'(x,y,i), w'(x,y,i)$ being the same as $u(x,y,i), v(x,y,i), w(x,y,i)$ if $(x,y) \in R$ and 0 otherwise. Then it is easy to see that it satisfies (1) so it is also a weight scheme. And for these new weight functions, we have $u'_{x,i} = \sum_{y:(x,y)\in R', x_i \neq y_i} u'(x,y,i) = \sum_{y:(x,y)\in R, x_i \neq y_i} u(x,y,i) = u_{x,i}$ and similarly $v'_{y,i} = v_{y,i}$ and $w'_x = w_x, w'_y = w_y$.[3] It follows that $\frac{w_x w_y}{u_{x,i} v_{y,i}} = \frac{w'_x w'_y}{u'_{x,i} v'_{y,i}}$, thus we can use (X', Y', R', u', v', w') to derive the same lower bound as we use (X, Y, R, u, v, w).

So we now suppose $R = X \times Y$ and prove that $\exists x \in X, y \in Y, i \in [N]$, *s.t.*

$$w_x w_y \leq N \cdot C_0(f)\, u_{x,i} v_{y,i},$$

Suppose the claim is not true. Then for all $x \in X, y \in Y, i \in [N]$, we have

$$w_x w_y > N \cdot C_0(f)\, u_{x,i} v_{y,i}. \tag{3}$$

We first fix i for the moment. And for each $x \in X$, we fix a smallest certificate set CS_x of f on x. Clearly $|CS_x| \leq C_0(f)$. We sum (3) over $\{x \in X : i \in CS_x\}$ and $\{y \in Y\}$. Then we get

$$\Big(\sum_{x\in X:\ i\in CS_x} w_x\Big)\Big(\sum_{y\in Y} w_y\Big) > N \cdot C_0(f)\Big(\sum_{x\in X:\ i\in CS_x} u_{x,i}\Big)\Big(\sum_{y\in Y} v_{y,i}\Big). \tag{4}$$

Note that $\sum_{y\in Y} w_y = \sum_{x\in X, y\in Y} w(x,y) = \sum_{x\in X} w_x$, and that $\sum_{y\in Y} v_{y,i} = \sum_{x\in X, y\in Y: x_i \neq y_i} v(x,y,i) = \sum_{x\in X} v_{x,i}$ where $v_{x,i} = \sum_{y\in Y: x_i \neq y_i} v(x,y,i)$. Inequality (4) now turns to

$$\begin{aligned}\Big(\sum_{x\in X: i\in CS_x} w_x\Big)\Big(\sum_{x\in X} w_x\Big) &> N \cdot C_0(f)\Big(\sum_{x\in X:\ i\in CS_x} u_{x,i}\Big)\Big(\sum_{x\in X} v_{x,i}\Big)\\ &\geq N \cdot C_0(f)\Big(\sum_{x\in X:\ i\in CS_x} u_{x,i}\Big)\Big(\sum_{x\in X:\ i\in CS_x} v_{x,i}\Big)\\ &\geq N \cdot C_0(f)\Big(\sum_{x\in X:\ i\in CS_x} \sqrt{u_{x,i} v_{x,i}}\Big)^2\end{aligned}$$

[3] Note that the function values of u', v', w' are zero when $(x,y) \neq R$, which does not conform to the definition of weight scheme. But actually Theorem 3 also holds for $u \geq 0, v \geq 0, w \geq 0$ as long as $u_{x,i}, v_{y,i}, w_x, w_y$ are all strictly positive for any x, y, i. This can be seen from the proof of Alb_4 in Section 4.

by Cauchy-Schwartz Inequality. We further note that

$$\begin{aligned} u_{x,i}v_{x,i} &= (\sum_{y\in Y:x_i\neq y_i} u(x,y,i))(\sum_{y\in Y:x_i\neq y_i} v(x,y,i)) \\ &\geq (\sum_{y\in Y:x_i\neq y_i} \sqrt{u(x,y,i)v(x,y,i)})^2 \\ &\geq (\sum_{y\in Y:x_i\neq y_i} w(x,y))^2 \\ &= (w_{x,i})^2 \end{aligned}$$

where we define $w_{x,i} = \sum_{y\in Y:x_i\neq y_i} w(x,y)$. Thus

$$(\sum_{x\in X:\ i\in CS_x} w_x)(\sum_{x\in X} w_x) > N \cdot C_0(f)(\sum_{x\in X:\ i\in CS_x} w_{x,i})^2 \qquad (5)$$

Now we sum (5) over $i = 1, ..., N$, and note that

$$\sum_i \sum_{x\in X:\ i\in CS_x} w_x = \sum_{x\in X} \sum_{i:i\in CS_x} w_x \leq C_0(f) \sum_{x\in X} w_x$$

because $|CS_x| \leq C_0(f)$ for each x. We have

$$(\sum_{x\in X} w_x)^2 > N \sum_{i=1}^{N} (\sum_{x\in X:\ i\in CS_x} w_{x,i})^2$$

By the arithmetic-square average inequality (or by Cauchy-Schwartz Inequality) $N(a_1^2 + ... + a_N^2) \geq (a_1 + ... + a_N)^2$, we have

$$\begin{aligned} (\sum_{x\in X} w_x)^2 &> (\sum_{x\in X, i\in[N]:i\in CS_x} w_{x,i})^2 = (\sum_{x\in X, i\in[N], y\in Y:i\in CS_x, x_i\neq y_i} w(x,y))^2 \\ &= (\sum_{x\in X, y\in Y} \sum_{i\in[N]:i\in CS_x, x_i\neq y_i} w(x,y))^2 \end{aligned}$$

But by the definition of certificate, we know that for any x and y there is at least one index $i \in CS_x$ *s.t.* $x_i \neq y_i$. Therefore, we derive an inequality

$$(\sum_{x\in X} w_x)^2 > (\sum_{x\in X, y\in Y} w(x,y))^2 = (\sum_{x\in X} w_x)^2$$

which is a contradiction, as desired. □

We add some comments about this upper bound of Alb_3. First, this bound looks weak at first glance because the $\sqrt{N}$ factor seems too large. But in fact it is necessary. Consider the problem of INVERT A PERMUTATION[5][4],where $C_0(f) = C_1(f) = 1$ but even the $Alb_2(f) = \Omega(\sqrt{N})$.

Second, the quantum query complexity of ELEMENT DISTINCTNESS is known to be $\Theta(N^{2/3})$. The lower bound part is obtained by Shi [28] (for large range) and Ambainis [7] (for small range); the upper bound part is obtained by Ambainis [8]. Observe that $C_1(f) = 2$ thus $\sqrt{NC_1(f)} = \Theta(N)$, we derive the following interesting corollary from the above theorem.

Corollary 1. *Alb_3 is not tight.*

[4] The original problem is not a Boolean function, but we can define a Boolean-valued version of it. Instead of finding the position i with $x_i = 1$, we are to decide whether i is odd or even. The original proof of the $\Omega(\sqrt{N})$ lower bound still holds.

3.2 Two Better Upper Bounds for Total Functions

It turns out that if the function is total, then the upper bound can be further tightened. We introduce a new measure which basically characterizes the size of intersection of a 0- and 1-certificate sets.

Definition 3. *For any function f, if there is a certificate set assignment $CS : \{0,1\}^N \to 2^{[N]}$ such that for any inputs x, y with $f(x) \neq f(y)$, $|CS_x \cap CS_y| \leq k$, then k is called a candidate certificate intersection complexity of f. The minimal candidate certificate intersection complexity of f is called the certificate intersection complexity of f, denoted by $CI(f)$. In other words, $CI(f) = \min_{CS} \max_{x,y:f(x)\neq f(y)} |CS_x \cap CS_y|$.*

Now we give the following theorem which improves Theorem 6 for total functions. Note that $CI(f) \leq C_-(f)$ by the definition of $CI(f)$.

Theorem 7. *$Alb_3(f) \leq \sqrt{N \cdot CI(f)}$, for any N-ary total Boolean function f.*

Proof. Again, we prove a stronger result that for any (X, Y, R, u, v, w) in Theorem 3,

$$\min_{(x,y)\in R, i\in[N]} \frac{w_x w_y}{u_{x,i} v_{y,i}} \leq N \cdot CI(f).$$

Similar to the proof for Theorem 6, we assume without loss of generality that $R = X \times Y$ and for all $x \in X, y \in Y$, we have

$$w_x w_y > N \cdot CI(f)\ u_{x,i} v_{y,i}. \tag{6}$$

We shall show a contradiction as follows. Fix i and sum (6) over $\{x \in X : i \in CS_x\}$ and $\{y \in Y : i \in CS_y\}$, we get

$$\begin{aligned}
&\textstyle\sum_{x\in X, y\in Y:\ i\in CS_x\cap CS_y} w_x w_y \\
&> N \cdot CI(f)\ (\textstyle\sum_{x\in X:\ i\in CS_x} u_{x,i})(\sum_{y\in Y:\ i\in CS_y} v_{y,i}) \\
&= N \cdot CI(f)\ (\textstyle\sum_{x\in X, y\in Y:\ i\in CS_x, x_i\neq y_i} u(x,y,i))\cdot(\sum_{x\in X, y\in Y: i\in CS_y, x_i\neq y_i} v(x,y,i)) \\
&\geq N \cdot CI(f)\ (\textstyle\sum_{x\in X, y\in Y:\ i\in CS_x\cap CS_y, x_i\neq y_i} u(x,y,i)) \\
&\qquad\qquad \cdot(\textstyle\sum_{x\in X, y\in Y:\ i\in CS_x\cap CS_y, x_i\neq y_i} v(x,y,i)) \\
&\geq N \cdot CI(f)\ (\textstyle\sum_{x\in X, y\in Y:\ i\in CS_x\cap CS_y, x_i\neq y_i} \sqrt{u(x,y,i)v(x,y,i)})^2 \\
&\geq N \cdot CI(f)\ (\textstyle\sum_{x\in X, y\in Y:\ i\in CS_x\cap CS_y, x_i\neq y_i} w(x,y))^2
\end{aligned}$$

Now sum over $i = 1, ..., N$, we get

$$\begin{aligned}
&\textstyle\sum_{x\in X, y\in Y, i\in[N]:\ i\in CS_x\cap CS_y} w_x w_y \\
&> N \cdot CI(f)\ \textstyle\sum_{i=1}^{N} (\sum_{x\in X, y\in Y:\ i\in CS_x\cap CS_y, x_i\neq y_i} w(x,y))^2 \\
&\geq CI(f)\ (\textstyle\sum_{x\in X, y\in Y, i\in[N]:\ i\in CS_x\cap CS_y, x_i\neq y_i} w(x,y))^2
\end{aligned}$$

Note that for total function f, if $f(x) \neq f(y)$, there is at least one position $i \in CS_x \cap CS_y$ *s.t.* $x_i \neq y_i$. Thus

$$\sum_{x\in X, y\in Y, i\in[N]:\ i\in CS_x\cap CS_y, x_i\neq y_i} w(x,y) \geq \sum_{x\in X, y\in Y} w(x,y)$$

On the other hand, by the definition of $CI(f)$, we have

$$\sum_{x\in X,y\in Y,i\in[N]:i\in CS_x\cap CS_y} w_x w_y \le CI(f)\sum_{x\in X,y\in Y} w_x w_y = CI(f)\left(\sum_{x\in X,y\in Y} w(x,y)\right)^2$$

Therefore we get a contradiction

$$CI(f)(\sum_{x\in X,y\in Y} w(x,y))^2 > CI(f)(\sum_{x\in X,y\in Y} w(x,y))^2.$$

as desired. □

AND-OR TREE is a famous problem in both classical and quantum computation. In the problem, there is a complete binary tree with height $2n$. Any node in odd levels is labelled with AND and any node in even levels is labelled with OR. The $N = 4^n$ leaves are the input variables, and the value of the function is the value that we get at the root, with value of each internal node calculated from the values of its two children in the common AND/OR interpretation. The classical randomized decision tree complexity for AND-OR TREE is known to be $\Theta((\frac{1+\sqrt{33}}{4})^n) = \Theta(N^{0.753\ldots})$ by Saks and Wigderson in [26] and Santha in [27]. The best known quantum lower bound is $\Omega(\sqrt{N})$ by Barnum and Saks in [9] and best quantum upper bound is no more than the best classical randomized one. Note that $C_-(\text{AND-OR TREE}) = 2^n = \sqrt{N}$ and thus $\sqrt{NC_-(f)} = N^{3/4}$. So if we only use Theorem 6, it seems that we still have chances to improve the known $\Omega(\sqrt{N})$ lower bound by Alb_3. But by Theorem 7 we know that actually it is impossible, because it is not hard to show $CI(f) = 1$ for AND-OR TREE. We leave the proof of this fact in the full version of the paper (and a preliminary report [33]), and only state the consequence as follows.

Corollary 2. $Alb_3(\text{AND-OR TREE}) \le \sqrt{N}$.

We can tighten the $\sqrt{N \cdot C_-(f)}$ upper bound in another way and get the following result which also implies Corollary 2.

Theorem 8. $Alb_3(f) \le \sqrt{C_0(f)C_1(f)}$, *for any total Boolean function* f.

Proof. For any (X, Y, R, u, v, w) in Theorem 3, we assume without loss of generality that $X \subseteq f^{-1}(0), Y \subseteq f^{-1}(1)$ and $R = X \times Y$. We are to prove $\exists x, y, i, j$ *s.t.* $w_x w_y \le C_0(f)C_1(f)u_{x,i}v_{y,j}$. Suppose this is not true, *i.e.* for all $x \in X, y \in Y, i, j \in [N]$, $w_x w_y > C_0(f)C_1(f)u_{x,i}v_{y,j}$. First fix x, y and sum over $i \in CS_x$ and $j \in CS_y$. Since $|CS_x| \le C_0(f), |CS_y| \le C_1(f)$, we have

$$w_x w_y > \sum_{i\in CS_x} u_{x,i} \sum_{j\in CS_y} v_{y,j}$$

Now we sum over $x \in X$ and $y \in Y$,

$$\begin{aligned}(\textstyle\sum_{x\in X} w_x)(\sum_{y\in Y} w_y) &> (\textstyle\sum_{x\in X,i\in CS_x} u_{x,i})(\sum_{y\in Y,j\in CS_y} v_{y,j}) \\ &= (\textstyle\sum_{x\in X,y\in Y,i\in[N]:x_i\neq y_i,i\in CS_x} u(x,y,i)) \\ &\quad \cdot(\textstyle\sum_{x\in X,y\in Y,j\in[N]:x_j\neq y_j,j\in CS_y} v(x,y,j))\end{aligned}$$

Since f is total, there is at least one $i_0 \in CS_x \cap CS_y$ *s.t.* $x_{i_0} \neq y_{i_0}$. Thus

$$\begin{aligned}(\sum_{x\in X} w_x)(\sum_{y\in Y} w_y) &> (\sum_{x\in X,y\in Y} u(x,y,i_0))(\sum_{x\in X,y\in Y} v(x,y,i_0)) \\ &\geq \sum_{x\in X,y\in Y} u(x,y,i_0)v(x,y,i_0) \\ &\geq (\sum_{x\in X,y\in Y} \sqrt{u(x,y,i_0)v(x,y,i_0)})^2 \\ &\geq (\sum_{x\in X,y\in Y} w(x,y))^2 \\ &= (\sum_{x\in X} w_x)(\sum_{y\in Y} w_y)\end{aligned}$$

which is a contradiction. □

Finally, we remark that even these two improved upper bounds of $Alb_3(f)$ are not always tight. For example, Yao and Zhang prove in [32] that graph property SCORPION, directed graph property SINK and a circular function all have $Q_2(f) = \tilde{\Theta}(\sqrt{n})$, but both $\sqrt{C_0(f)C_1(f)}$ and $\sqrt{N \cdot CI(f)}$ are $\Theta(n)$.

4 A Further Generalized Ambainis's Lower Bound

While Alb_2 and Alb_3 use different ideas to generalize Alb_1, it is natural to combine both and get a further generalization. The following theorem is a result in this direction. This theorem to Theorem 3 is as Theorem 2 to Theorem 1. We omit the proof for the space reason and leave it in the full version of the paper (and a preliminary report [33]).

Theorem 9. *Let* $f : I^N \to \{0,1\}$ *where I is a finite set, and X, Y be two sets of inputs* s.t. $f(x) \neq f(y)$ *if* $x \in X$ *and* $y \in Y$. *Let* $R \subseteq X \times Y$. *Let* w, u, v *be a weight scheme for X, Y, R. Then*

$$Q_2(f) = \Omega(\sqrt{\min_{(x,y)\in R, i\in[N], x_i\neq y_i} \frac{w_x w_y}{u_{x,i} v_{y,i}}})$$

We denote by $Alb_4(f)$ the best possible lower bound for function f achieved by this theorem. It is easy to see that Alb_4 generalizes Alb_3. It is actually stronger than the result proven Theorem 6 because here the minimum is over a set smaller than that in the left hand side of (2) due to the additional requirement $x_i \neq y_i$. However, according to Szegedy's recent result [31], Alb_3, Alb_4 and the quantum adversary method proposed by Barnum, Saks and Szegedy in [10] are all equivalent. Thus Alb_4 may be easier to use than Alb_3.

Acknowledgement. The author would like to thank Andrew Yao who introduced the quantum algorithm and quantum query complexity area to me, and made invaluable comments to this paper. Yaoyun Shi and Xiaoming Sun also read a preliminary version of the present paper and both, esp. Yaoyun Shi, gave many invaluable comments and corrections. Thanks also to Andris Ambainis for telling that it is still open whether $Alb_2(f) \leq \sqrt{C_0(f)C_1(f)}$ is true for total functions, and to Mario Szegedy for sending me his note [31].

References

[1] S. Aaronson. Quantum lower bound for the collision problem. Proceedings of STOC'02, pp. 635-642. Also quant-ph/0111102.

[2] S. Aaronson. Lower bounds for local search by quantum arguments. to appear in STOC'04. Also quant-ph/0307149

[3] A. Ambainis. A better lower bound for quantum algorithms searching an ordering list. FOCS'99, p.352-357. Also quant-ph/9902053

[4] A. Ambainis. Polynomial degree vs. quantum query complexity, FOCS'03, 2003. Earlier version at quant-ph/0305028

[5] A. Ambainis. Quantum lower bounds by quantum arguments, J. Comput. Sys. Sci. 64:750-767, 2002. Earlier version at STOC'00

[6] A. Ambainis. Quantum query algorithms and lower bounds, Proceedings of FOTFS III, to appear

[7] A. Ambainis. Quantum lower bounds for collision and element distinctness with small range, quant-ph/0305179

[8] A. Ambainis. Quantum walk algorithmn for element distinctness, quant-ph/0311001

[9] H. Barnum, M. Saks. A lower bound on the quantum query complexity of read-once functions, to appear in Journal of Computer and System Science. Earlier version at ECCC TR02-002, 2002

[10] H. Barnum, M. Saks, M. Szegedy. Quantum query complextiy and semidefinite programming, IEEE Conference on Computational Complexity 2003

[11] R. Beals, H. Buhrman, R. Cleve, M.Mosca, R. deWolf. Quantum lower bounds by polynomials. Journal of ACM, 48: 778-797, 2001. Earlier versions at FOCS'98 and quant-ph/9802049

[12] Berzina, Dubrovsky, Freivalds, Lace and Scegulnaja, Quantum query complexity for some graph problems. SOFSEM 2004: 140-150

[13] H. Buhrman, R. Cleve, A. Wigderson, Quantum vs. classical communication and computation, STOC'98, 63-68. quant-ph/9802040

[14] H. Buhrman, Durr, Ch., M. Heiligman, P. Hoyer, F. Magniez, M. Santha, R. De Wolf, Quantum algorithms for element distinctness, Complexity'01 131-137, quant-ph/0007016.

[15] H. Buhrman, R. de Wolf. Complexity measures and decision tree complexity: a survey. Theoretical Computer Science, Volume 288, Issue 1, 9 October 2002, Pages 21-43

[16] D. Deutsch, R. Jozsa. Rapid solution of problems by quantum computation. Proceeding of the Royal Society of London, A439(1992), 553-558

[17] C. Durr, M. Heiligman, P. Høyer, M. Mhalla, and Y. Lei. Quantum query complexity of some graph problems. Manuscript, 2003, cited by [21]

[18] C. Durr, M. Mhalla, Y. Lei. Quantum query complexity of graph connectivity, quant-ph/0303169

[19] L. Grover. A fast quantum mechaqnical algorithm for database search, STOC'96, 212-219

[20] P. Høyer, J. Neerbek, Y. Shi. Quantum lower bounds of ordered searching, sorting and element distinctness. Algorithmica, 34: 429-448, 2002. Eearlier versions at ICALP'01 and quant-ph/0102078

[21] S. Laplante, F. Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. quant-ph/0311189

[22] F. Magniez, M. Santha, M. Szegedy. An $O(n^{1.3})$ quantum algorithm for the Triangle problem, quant-ph/0310134
[23] A. Nayak, F. Wu. The quantum query complexity of approximating the median and related statistics. Proceedings of STOC'99, pp. 384-393, also quant-ph/9804066.
[24] R. Paturi. On the degree of polynomials that approximate symmetric Boolean functions, STOC'92, 468-474.
[25] A. Razborov. Quantum communication complexity of symmetric predicates, Izvestiya of the Russian Academy of Science, mathematics, 2002. also quant-ph/0204025.
[26] M. Saks, A. Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. FOCS 1986, pp. 29-38.
[27] M. Santha. On the Monte Carlo Boolean decision tree complexity of read-once formulae. Structures 1991, pp. 180-187.
[28] Y. Shi. Quantum lower bounds for the collision and the element distinctness problems. FOCS'02, 513-519, Earlier version at quant-ph/0112086
[29] P. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing, 26:1484-1509, 1997
[30] Mario Szegedy. On the quantum query complexity of detecting triangles in graphs, quant-ph/0310107
[31] Mario Szegedy. A note on the limitations of quantum adversary method, manuscript
[32] Andrew Yao, Shengyu Zhang. Graph property and circular functions: how low can quantum query complexity go?
[33] Shengyu Zhang. On the power of Ambainis's lower bounds, quant-ph/0311060

Author Index