

Lecture Notes in Computer Science

2719

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer

Berlin

Heidelberg

New York

Barcelona

Hong Kong

London

Milan

Paris

Tokyo

Jos C.M. Baeten Jan Karel Lenstra
Joachim Parrow Gerhard J. Woeginger (Eds.)

Automata, Languages and Programming

30th International Colloquium, ICALP 2003
Eindhoven, The Netherlands, June 30 – July 4, 2003
Proceedings



Springer

Volume Editors

Jos C.M. Baeten

Technische Universiteit Eindhoven, Dept. of Mathematics and Computer Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
E-mail: j.c.m.baeten@tue.nl

Jan Karel Lenstra

Georgia Institute of Technology, School of Industrial and Systems Engineering
765 Ferst Drive, Atlanta, GA 30332-0205, USA
E-mail: jkl@isye.gatech.edu

Joachim Parrow

Uppsala University, Department of Information Technology
P.O. Box 337, 75105 Uppsala, Sweden
E-mail: joachim@docs.uu.se

Gerhard J. Woeginger

University of Twente
Faculty of Electrical Engineering, Mathematics and Computer Science
P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: g.j.woeginger@math.utwente.nl

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

Bibliographic information published by Die Deutsche Bibliothek

Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie;
detailed bibliographic data is available in the Internet at <<http://dnb.ddb.de>>.

CR Subject Classification (1998): F, D, C.2-3, G.1-2, I.3, E.1-2

ISSN 0302-9743

ISBN 3-540-40493-7 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2003
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin GmbH
Printed on acid-free paper SPIN: 10928936 06/3142 5 4 3 2 1 0

Preface

The 30th International Colloquium on Automata, Languages and Programming (ICALP 2003) was held from June 30 to July 4 on the campus of the Technische Universiteit Eindhoven (TU/e) in Eindhoven, The Netherlands. This volume contains all contributed papers presented at ICALP 2003, together with the invited lectures by Jan Bergstra (Amsterdam), Anne Condon (Vancouver), Amos Fiat (Tel Aviv), Petra Mutzel (Vienna), Doron Peled (Coventry) and Moshe Vardi (Houston).

Since 1972, ICALP has been the main annual event of the European Association for Theoretical Computer Science (EATCS). The ICALP program can be divided into two tracks, viz. track A (algorithms, automata, complexity, and games) and track B (logics, semantics, and theory of programming).

In response to the Call for Papers, the program committee received 212 submissions: 131 for track A and 81 for track B. The committee met on March 14 and 15 in Haarlem, The Netherlands and selected 84 papers for inclusion in the scientific program. The selection was based on originality, quality and relevance to theoretical computer science. We wish to thank all authors who submitted extended abstracts for consideration, and all referees and subreferees who helped in the extensive evaluation process.

The EATCS Best Paper Award for Track A was given to the paper “The Cell Probe Complexity of Succinct Data Structures” by Anna Gál and Peter Bro Miltersen and the award for Track B was given to the paper “A Testing Scenario for Probabilistic Automata” by Mariëlle Stoelinga and Frits Vaandrager.

ICALP 2003 was a special ICALP. Two other computer science conferences co-located with ICALP this time: the 24th International Conference on Application and Theory of Petri Nets (ATPN 2003) and the Conference on Business Process Management (BPM 2003). During ICALP 2003 the following special events took place: the EATCS Distinguished Service Award was given to Grzegorz Rozenberg (Leiden), and the Lifetime Achievement Award of the NVTI (Dutch Association for Theoretical Computer Science) was given to N.G. de Bruijn (Eindhoven). Several high-level workshops were held as satellite events of ICALP 2003, coordinated by Erik de Vink. These included the following workshops: Algorithms for Massive Data Sets, Foundations of Global Computing (FGC), Logic and Communication in Multi-Agent Systems (LCMAS), Quantum Computing, Security Issues in Coordination Models, Languages and Systems (SecCo), Stochastic Petri Nets, Evolutionary Algorithms, the 1st International Workshop on the Future of Neural Networks (FUNN), and Mathematics, Logic and Computation (workshop in honor of N.G. de Bruijn’s 85th birthday). In addition, there was a discussion forum on Education Matters — the Challenge of Teaching Theoretical Computer Science organized by Hans-Joerg Kreowski.

The scientific program of ICALP 2003 and satellite workshops showed that theoretical computer science is a vibrant field, deepening our insights into the foundations and future of computing and system design in many application areas.

The sponsors of ICALP 2003 included the municipality of Eindhoven, Sodexho, Océ, the research school IPA, the European Educational Forum, Springer-Verlag, Elsevier, Philips Research, Atos Origin, Pallas Athena, Pearson Education Benelux, and ABE Foundation. We are very grateful to the Technische Universiteit Eindhoven for supporting and hosting ICALP 2003. The organizing committee consisted of Jos Baeten, Tijn Borghuis, Erik Luit, Emmy van Otterdijk, Anne-Meta Oversteegen, Thieu Rietjens, Karin Touw and Erik de Vink, all of the TU/e. Thanks is owed to them, and to everybody else who helped, for their outstanding effort in making ICALP 2003 a success.

June 2003

Jos Baeten
Jan Karel Lenstra
Joachim Parrow
Gerhard Woeginger

Program Committee

Track A

Harry Buhrman, CWI Amsterdam
Jens Clausen, DTK Lyngby
Martin Dyer, Leeds
Lars Engebretsen, KTH Stockholm
Uri Feige, Weizmann
Philippe Flajolet, INRIA Rocquencourt
Kazuo Iwama, Kyoto
Elias Koutsoupas, UCLA
Jan Karel Lenstra, Georgia Tech, Co-chair
Stefano Leonardi, Roma
Rasmus Pagh, Copenhagen
Jean-Eric Pin, CNRS and Paris 7
Uwe Schoening, Ulm
Jiri Sgall, CAS Praha
Micha Sharir, Tel Aviv
Vijay Vazirani, Georgia Tech
Ingo Wegener, Dortmund
Peter Widmayer, ETH Zürich
Gerhard Woeginger, Twente, Co-chair

Track B

Samson Abramsky, Oxford
Eike Best, Oldenburg
Manfred Broy, TU München
Philippe Darondeau, INRIA Rennes
Rocco De Nicola, Firenze
Rob van Glabbeek, Stanford
Ursula Goltz, Braunschweig
Roberto Gorrieri, Bologna
Robert Harper, Carnegie Mellon
Holger Hermanns, Twente
Kim Larsen, Aalborg
Jean-Jacques Levy, INRIA Rocquencourt
Flemming Nielson, DTU Lyngby
Prakash Panangaden, McGill
Joachim Parrow, Uppsala, chair
Amir Pnueli, Weizmann
Davide Sangiorgi, INRIA Sophia
Bernhard Steffen, Dortmund
Björn Victor, Uppsala

Referees

Karen Aardal	Julien Cassaigne	Paul Fischer
Parosh Abdulla	Didier Caucal	Hans Fleischhack
Luca Aceto	Amit Chakrabarti	Emmanuel Fleury
Jiri Adamek	Christian Choffrut	Wan Fokkink
Pankaj Agarwal	Marek Chrobak	Cédric Fournet
Susanne Albers	Mark Cieliebak	Gudmund Frandsen
Alessandro Aldini	Mario Coppo	Martin Fränzle
Jean-Paul Allouche	Robert Cori	Thomas Franke
Noga Alon	Flavio Corradini	Séverine Fratani
André Arnold	Cas Cremers	Ari Freund
Lars Arvestad	Vincent Cremet	Alan Frieze
Vincenzo Auletta	Maxime Crochemore	Toshihiro Fujito
Giorgio Ausiello	Mary Cryan	Naveen Garg
Holger Austinat	Artur Czumaj	Olivier Gascuel
Yossi Azar	Peter Damaschke	Michael Gatto
Marie-Pierre Béal	Ivan Damgaard	Stéphane Gaubert
Christel Baier	Zhe Dang	Cyril Gavoille
Amotz Bar-Noy	Olivier Danvy	Blaise Genest
Peter Baumgartner	Pedro D'Argenio	Dan Ghica
Danièle Beauquier	Giorgio Delzanno	Jeremy Gibbons
Luca Becchetti	Jörg Derungs	Oliver Giel
Marek Bednarczyk	Josee Desharnais	Inge Li Gørtz
Gerd Behrmann	Alessandra Di Pierro	Leslie Goldberg
Michael Bender	Volker Diekert	Mikael Goldmann
Thorsten Bernholt	Martin Dietzfelbinger	Roberta Gori
Vincent Berry	Dino Distefano	Mart de Graaf
Jean Berstel	Stefan Droste	Serge Grigorieff
Philip Bille	Abbas Edalat	Martin Grohe
Lars Birkedal	Stefan Edelkamp	Jan Friso Groote
Markus Blaeser	Stephan Eidenbenz	Roberto Grossi
Bruno Blanchet	Isaac Elias	Claudia Gsottberger
Luc Boasson	Leah Epstein	Joshua Guttman
Chiara Bodei	Thomas Erlebach	Johan Håstad
Hans Bodlaender	Eric Fabre	Stefan Haar
Beate Bollig	Rolf Fagerberg	Lisa Hales
Viviana Bono	Francois Fages	Mikael Hammar
Michele Boreale	Stefan Felsner	Chris Hankin
Ahmed Bouajjani	Paolo Ferragina	Rene Rydhof Hansen
Peter Braun	Jiří Fiala	Sariel Har-Peled
Franck van Breugel	Amos Fiat	Jerry den Hartog
Mikael Buchholtz	Andrzej Filinski	Gustav Hast
Daniel Bünzli	Bernd Finkbeiner	Anne Haxthausen
Marzia Buscemi	Alain Finkel	Fabian Hennecke
Nadia Busi	Thomas Firley	Thomas Hildebrandt

Yoram Hirshfeld
 Thomas Hofmeister
 Jonas Holmerin
 Juraj Hromkovic
 Michaela Huhn
 Hardi Hungar
 Thore Husfeldt
 Michael Huth
 Oscar H. Ibarra
 Keiko Imai
 Purush Iyer
 Jan Jürjens
 Radha Jagadeesan
 Jens Jägersküpper
 Petr Jančar
 Klaus Jansen
 Thomas Jansen
 Mark Jerrum
 Tao Jiang
 Magnus Johansson
 Georgi Jojgov
 Jørn Justesen
 Erich Kaltofen
 Viggo Kann
 Haim Kaplan
 Juhani Karhumäki
 Anna Karlin
 Joost-Pieter Katoen
 Claire Kenyon
 Rohit Khandekar
 Joe Kilian
 Josva Kleist
 Bartek Klin
 Jens Knoop
 Stavros Kolliopoulos
 Petr Kolman
 Jochen Konemann
 Guy Kortsarz
 Juergen Koslowski
 Michal Koucký
 Daniel Král'
 Jan Krájček
 Dieter Kratsch
 Matthias Krause
 Michael Krivelevich

Werner Kuich
 Dietrich Kuske
 Salvatore La Torre
 Anna Labella
 Ralf Laemmel
 Jim Laird
 Cosimo Laneve
 Martin Lange
 Ruggero Lanotte
 Francois Laroussinie
 Thierry Lecroq
 Troy Lee
 James Leifer
 Arjen Lenstra
 Reinhold Letz
 Francesca Levi
 Huimin Lin
 Andrzej Lingas
 Luigi Liquori
 Markus Lohrey
 Sylvain Lombardy
 Michele Loreti
 Roberto Lucchi
 Gerald Luetzgen
 Eva-Marta Lundell
 Parthasarathy
 Madhusudan
 Jean Mairesse
 Kazuhisa Makino
 Oded Maler
 Luc Maranget
 Alberto
 Marchetti-Spaccamela
 Martin Mareš
 Frank Marschall
 Fabio Martinelli
 Andrea Masini
 Sjouke Mauw
 Richard Mayr
 Colin McDiarmid
 Pierre McKenzie
 Michael Mendler
 Christian Michaux
 Kees Middelburg
 Stefan Milius

Peter Bro Miltersen
 Joe Mitchell
 Eiji Miyano
 Faron Moller
 Franco Montagna
 Christian Mortensen
 Peter Mosses
 Tilo Muecke
 Markus Mueller-Olm
 Madhavan Mukund
 Haiko Muller
 Ian Munro
 Andrzej Murawski
 Anca Muscholl
 Hiroshi Nagamochi
 Seffi Naor
 Margherita Napoli
 Uwe Nestmann
 Rolf Niedermeier
 Mogens Nielsen
 Stefan Nilsson
 Takao Nishizeki
 Damian Niwinski
 John Noga
 Thomas Noll
 Christian N.S. Pedersen
 Gethin Norman
 Manuel Núñez
 Marc Nunkesser
 Anna Östlin
 David von Oheimb
 Yoshio Okamoto
 Paulo Oliva
 Nicolas Ollinger
 Hirotaka Ono
 Vincent van Oostrom
 Janos Pach
 Catuscia Palamidessi
 Anna Palbom
 Mike Palis
 Alessandro Panconesi
 Christos Papadimitriou
 Andrzej Pelc
 David Peleg
 Holger Petersen

Seth Pettie	Norbert Schirmer	Mauro Torelli
Iain Phillips	Konrad Schlude	Stavros Tripakis
Giovanni Pighizzini	Philippe Schnoebelen	John Tromp
Henrik Pilegaard	Philip Scott	Emilio Tuosto
Sophie Pinchinat	Roberto Segala	Irek Ulidowski
G. Michele Pinna	Helmut Seidl	Yaroslav Usenko
Conrad Pomm	Peter Selinger	Frits Vaandrager
Ely Porat	Nicolas Sendrier	Frank Valencia
Giuseppe Prencipe	Maria Serna	Vincent Vanackère
Corrado Priami	Alexander Shen	Moshe Vardi
Guido Proietti	Natalia Sidorova	Helmut Veith
Pavel Pudlák	Detlef Sieling	Laurent Viennot
Rosario Pugliese	Marc Sihling	Alexander Vilbig
Uri Rabinovich	Hans Simon	Jørgen Villadsen
Theis Rauhe	Alex Simpson	Erik de Vink
Andreas Rausch	Michael Sipser	Paul Vitanyi
António Ravara	Martin Skutella	Berthold Voecking
Klaus Reinhardt	Michiel Smid	Walter Vogler
Michel A. Reniers	Pawel Sobocinski	Marc Voorhoeve
Arend Rensink	Eljas Soisalon-Soininen	Tjark Vredeveld
Christian Retoré	Ana Sokolova	Stephan Waack
James Riley	Frits Spieksma	Igor Walukiewicz
Martin Roetteler	Renzo Sprugnoli	Dietmar Wätjen
Maurice Rojas	Jiří Srba	Birgitta Weber
Marie-Francoise Roy	Rob van Stee	Heike Wehrheim
Oliver Ruething	Angelika Steger	Elke Wilkeit
Bernhard Rumpe	Christian Stehno	Tim Willemse
Wojciech Rytter	Ralf Steinbrueggen	Harro Wimmel
Géraud Sénizergues	Colin Stirling	Peter Winkler
Nicoletta Sabatini	Leen Stougie	Carsten Witt
Andrei Sabelfeld	Martin Strecker	Philipp Woelfel
Kunihiko Sadakane	Werner Struckmann	Ronald de Wolf
Marie-France Sagot	Hongyan Sun	Derick Wood
Louis Salvail	Ichiro Suzuki	Jürg Wullschleger
Bruno Salvy	Tetsuya Takine	Shigeru Yamashita
Christian Salzmann	Hisao Tamaki	Wang Yi
Peter Sanders	Amnon Ta-Shma	Heisung Yoo
Miklos Santha	David Taylor	Hans Zantema
Martin Sauerhoff	Pascal Tesson	Gianluigi Zavattaro
Daniel Sawitzki	Simone Tini	Pascal Zimmer
Andreas Schaefer	Takeshi Tokuyama	Uri Zwick

Table of Contents

Invited Lectures

Polarized Process Algebra and Program Equivalence	1
<i>Jan A. Bergstra, Inge Bethke</i>	
Problems on RNA Secondary Structure Prediction and Design	22
<i>Anne Condon</i>	
Some Issues Regarding Search, Censorship, and Anonymity in Peer to Peer Networks	33
<i>Amos Fiat</i>	
The SPQR-Tree Data Structure in Graph Drawing	34
<i>Petra Mutzel</i>	
Model Checking and Testing Combined	47
<i>Doron Peled</i>	
Logic and Automata: A Match Made in Heaven	64
<i>Moshe Y. Vardi</i>	

Algorithms

Pushdown Automata and Multicounter Machines, a Comparison of Computation Modes	66
<i>Juraj Hromkovič, Georg Schnitger</i>	
Generalized Framework for Selectors with Applications in Optimal Group Testing	81
<i>Annalisa De Bonis, Leszek Gąsieniec, Ugo Vaccaro</i>	
Decoding of Interleaved Reed Solomon Codes over Noisy Data	97
<i>Daniel Bleichenbacher, Aggelos Kiayias, Moti Yung</i>	

Process Algebra

On the Axiomatizability of Ready Traces, Ready Simulation, and Failure Traces	109
<i>Stefan Blom, Wan Fokkink, Sumit Nain</i>	
Resource Access and Mobility Control with Dynamic Privileges Acquisition	119
<i>Daniele Gorla, Rosario Pugliese</i>	

Replication vs. Recursive Definitions in Channel Based Calculi	133
<i>Nadia Busi, Maurizio Gabbrielli, Gianluigi Zavattaro</i>	

Approximation Algorithms

Improved Combinatorial Approximation Algorithms for the k -Level Facility Location Problem	145
<i>Alexander Ageev, Yinyu Ye, Jiawei Zhang</i>	

An Improved Approximation Algorithm for the Asymmetric TSP with Strengthened Triangle Inequality	157
<i>Markus Bläser</i>	

An Improved Approximation Algorithm for Vertex Cover with Hard Capacities	164
<i>Rajiv Gandhi, Eran Halperin, Samir Khuller, Guy Kortsarz, Aravind Srinivasan</i>	

Approximation Schemes for Degree-Restricted MST and Red-Blue Separation Problem	176
<i>Sanjeev Arora, Kevin L. Chang</i>	

Approximating Steiner k -Cuts	189
<i>Chandra Chekuri, Sudipto Guha, Joseph Naor</i>	

MAX k -CUT and Approximating the Chromatic Number of Random Graphs	200
<i>Amin Coja-Oghlan, Cristopher Moore, Vishal Sanwalani</i>	

Approximation Algorithm for Directed Telephone Multicast Problem ...	212
<i>Michael Elkin, Guy Kortsarz</i>	

Languages and Programming

Mixin Modules and Computational Effects	224
<i>Davide Ancona, Sonia Fagorzi, Eugenio Moggi, Elena Zucca</i>	

Decision Problems for Language Equations with Boolean Operations	239
<i>Alexander Okhotin</i>	

Generalized Rewrite Theories	252
<i>Roberto Bruni, José Meseguer</i>	

Complexity

Sophistication Revisited	267
<i>Luís Antunes, Lance Fortnow</i>	

Scaled Dimension and Nonuniform Complexity	278
<i>John M. Hitchcock, Jack H. Lutz, Elvira Mayordomo</i>	

Quantum Search on Bounded-Error Inputs	291
<i>Peter Høyer, Michele Mosca, Ronald de Wolf</i>	

A Direct Sum Theorem in Communication Complexity via Message Compression	300
<i>Rahul Jain, Jaikumar Radhakrishnan, Pranab Sen</i>	

Data Structures

Optimal Cache-Oblivious Implicit Dictionaries	316
<i>Gianni Franceschini, Roberto Grossi</i>	

The Cell Probe Complexity of Succinct Data Structures	332
<i>Anna Gál, Peter Bro Miltersen</i>	

Succinct Representations of Permutations	345
<i>J. Ian Munro, Rajeev Raman, Venkatesh Raman, Satti Srinivasa Rao</i>	

Succinct Dynamic Dictionaries and Trees	357
<i>Rajeev Raman, Satti Srinivasa Rao</i>	

Graph Algorithms

Labeling Schemes for Weighted Dynamic Trees	369
<i>Amos Korman, David Peleg</i>	

A Simple Linear Time Algorithm for Computing a $(2k - 1)$ -Spanner of $O(n^{1+1/k})$ Size in Weighted Graphs	384
<i>Surender Baswana, Sandeep Sen</i>	

Multicommodity Flows over Time: Efficient Algorithms and Complexity	397
<i>Alex Hall, Steffen Hippler, Martin Skutella</i>	

Multicommodity Demand Flow in a Tree	410
<i>Chandra Chekuri, Marcelo Mydlarz, F. Bruce Shepherd</i>	

Automata

Skew and Infinitary Formal Power Series	426
<i>Manfred Droste, Dietrich Kuske</i>	

Nondeterminism versus Determinism for Two-Way Finite Automata: Generalizations of Sipser's Separation	439
<i>Juraj Hromkovič, Georg Schnitger</i>	

Residual Languages and Probabilistic Automata	452
<i>François Denis, Yann Esposito</i>	

A Testing Scenario for Probabilistic Automata	464
<i>Mariëlle Stoelinga, Frits Vaandrager</i>	
The Equivalence Problem for t -Turn DPDA Is Co-NP	478
<i>Géraud Sénizergues</i>	
Flip-Pushdown Automata: $k + 1$ Pushdown Reversals Are Better than k	490
<i>Markus Holzer, Martin Kutrib</i>	

Optimization and Games

Convergence Time to Nash Equilibria	502
<i>Eyal Even-Dar, Alex Kesselman, Yishay Mansour</i>	
Nashification and the Coordination Ratio for a Selfish Routing Game	514
<i>Rainer Feldmann, Martin Gairing, Thomas Lücking, Burkhard Monien, Manuel Rode</i>	
Stable Marriages with Multiple Partners: Efficient Search for an Optimal Solution	527
<i>Vipul Bansal, Aseem Agrawal, Varun S. Malhotra</i>	
An Intersection Inequality for Discrete Distributions and Related Generation Problems	543
<i>Endre Boros, Khaled Elbassioni, Vladimir Gurvich, Leonid Khachiyan, Kazuhisa Makino</i>	

Graphs and Bisimulation

Higher Order Pushdown Automata, the Causal Hierarchy of Graphs and Parity Games	556
<i>Thierry Cachet</i>	
Undecidability of Weak Bisimulation Equivalence for 1-Counter Processes	570
<i>Richard Mayr</i>	
Bisimulation Proof Methods for Mobile Ambients	584
<i>Massimo Merro, Francesco Zappa Nardelli</i>	
On Equivalent Representations of Infinite Structures	599
<i>Arnaud Carayol, Thomas Colcombet</i>	

Online Problems

Adaptive Raising Strategies Optimizing Relative Efficiency	611
<i>Arnold Schönhage</i>	
A Competitive Algorithm for the General 2-Server Problem	624
<i>René A. Sitters, Leen Stougie, Willem E. de Paepe</i>	

On the Competitive Ratio for Online Facility Location	637
<i>Dimitris Fotakis</i>	

A Study of Integrated Document and Connection Caching	653
<i>Susanne Albers, Rob van Stee</i>	

Verification

A Solvable Class of Quadratic Diophantine Equations with Applications to Verification of Infinite-State Systems	668
<i>Gaoyan Xie, Zhe Dang, Oscar H. Ibarra</i>	

Monadic Second-Order Logics with Cardinalities	681
<i>Felix Klaedtke, Harald Rueß</i>	

$\Pi_2 \cap \Sigma_2 \equiv AFMC$	697
<i>Orna Kupferman, Moshe Y. Vardi</i>	

Upper Bounds for a Theory of Queues	714
<i>Tatiana Rybina, Andrei Voronkov</i>	

Around the Internet

Degree Distribution of the FKP Network Model	725
<i>Noam Berger, Béla Bollobás, Christian Borgs, Jennifer Chayes, Oliver Riordan</i>	

Similarity Matrices for Pairs of Graphs	739
<i>Vincent D. Blondel, Paul Van Dooren</i>	

Algorithmic Aspects of Bandwidth Trading	751
<i>Randeep Bhatia, Julia Chuzhoy, Ari Freund, Joseph Naor</i>	

Temporal Logic and Model Checking

CTL ⁺ Is Complete for Double Exponential Time	767
<i>Jan Johannsen, Martin Lange</i>	

Hierarchical and Recursive State Machines with Context-Dependent Properties	776
<i>Salvatore La Torre, Margherita Napoli, Mimmo Parente, Gennaro Parlato</i>	

Oracle Circuits for Branching-Time Model Checking	790
<i>Philippe Schnoebelen</i>	

Graph Problems

There Are Spanning Spiders in Dense Graphs (and We Know How to Find Them)	802
<i>Luisa Gargano, Mikael Hammar</i>	
The Computational Complexity of the Role Assignment Problem	817
<i>Jiří Fiala, Daniël Paulusma</i>	
Fixed-Parameter Algorithms for the (k, r) -Center in Planar Graphs and Map Graphs	829
<i>Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, Dimitrios M. Thilikos</i>	
Genus Characterizes the Complexity of Graph Problems: Some Tight Results	845
<i>Jianer Chen, Iyad A. Kanj, Ljubomir Perković, Eric Sedgwick, Ge Xia</i>	

Logic and Lambda-Calculus

The Definition of a Temporal Clock Operator	857
<i>Cindy Eisner, Dana Fisman, John Havlicek, Anthony McIsaac, David Van Campenhout</i>	
Minimal Classical Logic and Control Operators	871
<i>Zena M. Ariola, Hugo Herbelin</i>	
Counterexample-Guided Control	886
<i>Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar</i>	
Axiomatic Criteria for Quotients and Subobjects for Higher-Order Data Types	903
<i>Jo Hannay</i>	

Data Structures and Algorithms

Efficient Pebbling for List Traversal Synopses	918
<i>Yossi Matias, Ely Porat</i>	
Function Matching: Algorithms, Applications, and a Lower Bound	929
<i>Amihoud Amir, Yonatan Aumann, Richard Cole, Moshe Lewenstein, Ely Porat</i>	
Simple Linear Work Suffix Array Construction	943
<i>Juha Kärkkäinen, Peter Sanders</i>	

Types and Categories

Expansion Postponement via Cut Elimination in Sequent Calculi for Pure Type Systems	956
<i>Francisco Gutiérrez, Blas Ruiz</i>	
Secrecy in Untrusted Networks	969
<i>Michele Bugliesi, Silvia Crafa, Amela Prelic, Vladimiro Sassone</i>	
Locally Commutative Categories	984
<i>Arkadev Chattopadhyay, Denis Thérien</i>	

Probabilistic Systems

Semi-pullbacks and Bisimulations in Categories of Stochastic Relations ..	996
<i>Ernst-Erich Doberkat</i>	
Quantitative Analysis of Probabilistic Lossy Channel Systems	1008
<i>Alexander Rabinovich</i>	
Discounting the Future in Systems Theory	1022
<i>Luca de Alfaro, Thomas A. Henzinger, Rupak Majumdar</i>	
Information Flow in Concurrent Games	1038
<i>Luca de Alfaro, Marco Faella</i>	

Sampling and Randomness

Impact of Local Topological Information on Random Walks on Finite Graphs	1054
<i>Satoshi Ikeda, Izumi Kubo, Norihiro Okumoto, Masafumi Yamashita</i>	
Analysis of a Simple Evolutionary Algorithm for Minimization in Euclidean Spaces	1068
<i>Jens Jägersküpper</i>	
Optimal Coding and Sampling of Triangulations	1080
<i>Dominique Poulalhon, Gilles Schaeffer</i>	
Generating Labeled Planar Graphs Uniformly at Random	1095
<i>Manuel Bodirsky, Clemens Gröpl, Mihyun Kang</i>	

Scheduling

Online Load Balancing Made Simple: Greedy Strikes Back	1108
<i>Pilu Crescenzi, Giorgio Gambosi, Gaia Nicosia, Paolo Penna, Walter Unger</i>	
Real-Time Scheduling with a Budget	1123
<i>Joseph Naor, Hadas Shachnai, Tami Tamir</i>	

Improved Approximation Algorithms for Minimum-Space Advertisement
Scheduling 1138
Brian C. Dean, Michel X. Goemans

Anycasting in Adversarial Systems: Routing and Admission Control 1153
Baruch Awerbuch, André Brinkmann, Christian Scheideler

Geometric Problems

Dynamic Algorithms for Approximating Interdistances 1169
Sergei Bespamyatnikh, Michael Segal

Solving the Robots Gathering Problem 1181
Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro

Author Index 1197

Polarized Process Algebra and Program Equivalence

Jan A. Bergstra^{1,2} and Inge Bethke²

¹ Applied Logic Group, Department of Philosophy, Utrecht University,
Heidelberglaan 8, 3584 CS Utrecht, The Netherlands, Jan.Bergstra@phil.uu.nl

² Programming Research Group, Informatics Institute, Faculty of Science, University
of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands,
inge@science.uva.nl

Abstract. The basic polarized process algebra is completed yielding as a projective limit a cpo which also comprises infinite processes. It is shown that this model serves in a natural way as a semantics for several program algebras. In particular, the fully abstract model of the program algebra axioms of [2] is considered which results by working modulo behavioral congruence. This algebra is extended with a new basic instruction, named ‘entry instruction’ and denoted with ‘@’. Addition of @ allows many more equations and conditional equations to be stated. It becomes possible to find an axiomatization of program inequality. Technically this axiomatization is an infinite final algebra specification using conditional equations and auxiliary objects.

1 Introduction

Program algebra as introduced in [2] and [3] is a tool for the conceptualization of programs and programming. It is assumed that a program is executed in a context composed of components complementary to the program. While a program’s actions constitute requests to be processed by an environment, the complementary system components in an environment view actions as request issued by another party (the program being run). After each request the environment may undergo a state change whereupon it replies with a boolean value. The boolean return value is used to decide how the execution of the program will continue.

For theoretical work on program algebra a semantic model is important. It is assumed that the meaning of a program is a process. A particular kind of processes termed *polarized processes* is well-suited to serve as the semantic interpretation of a program. In this paper the semantic world of polarized processes is introduced following the presentation of [3]. Polarized process algebra can stand on its own feet though significant results allowing to maintain it as an independent subject are currently missing. Then program algebra is introduced as a formalism for denoting objects (programs) that can be mapped into the set of polarized processes in a natural fashion. Several program algebras are defined. One of these structures may be classified as fully abstract. The focus

of the paper is on an analysis of aspects of that model. This eventually leads to a final algebra specification of the fully abstract model. It seems to be the case that the fully abstract program algebra resists straightforward methods of algebraic specification. No negative results have been obtained, however. Several problems are left open.

2 Basic Polarized Process Algebra

Most process algebras (e.g. ACP from [1] and TCSP from [6]) are non-polarized. This means that in a parallel composition of process P and Q , both processes and their actions have a symmetric status. In a polarized setting each action has a definite asymmetric status. Either it is a request or it is (part of) the processing of a request. When a request action is processed a boolean value is returned to the process issuing the request. When this boolean value is returned the processing of the request is completed.

Non-polarized process algebra may be (but need not) considered the simplified case in which always **true** is returned. Polarized process algebra is less elegant than non-polarized process algebra. Its advantage lies in the more direct modeling of sequential deterministic systems. Polarized process algebra need not dive into the depths of choice and non-determinism when deterministic systems are discussed.

BPPA is based on a collection Σ of basic actions¹. Each action is supposed to be polarized and to produce a boolean value when executed. In addition its execution may have some side-effect in an environment. One imagines the boolean value mentioned above to be generated while this side-effect on the environment is being produced. BPPA has two constants which are meant to model termination and inaction and two composition mechanisms, the second one of these being defined in terms of the first one.

Definition 1. *For a collection Σ of atomic actions, $BPPA_{\Sigma}$ denotes the family of processes inductively defined by*

termination: $S \in BPPA_{\Sigma}$

With S (stop) terminating behavior is denoted; it does no more than terminate. Termination actions will not have any side effect on a state.

inaction: $D \in BPPA_{\Sigma}$

By D (sometimes just ‘loop’) an inactive behavior is indicated. It is a behav-

¹ The phrase ‘basic action’ is used in polarized process algebra in contrast with ‘atomic action’ as used in process algebra. Indeed from the point of view of ordinary process algebra the basic actions are not considered atomic. In program algebra the phrase ‘basic instruction’ is used. Basic instructions are mapped on basic actions if the semantics of program algebra is described in terms of a polarized process algebra. Program algebra also features so-called primitive instructions. These are the basic instructions without test (void uses) and with positive or negative test, the termination instruction as well as a jump instruction $\#n$ for each $n \in \mathbb{N}$.

ior that represents the impossibility of making real progress, for instance an internal cycle of activity without any external effect whatsoever².

postconditional composition: For action $a \in \Sigma$ and processes P and Q in $BPPA_\Sigma$

$$P \trianglelefteq a \triangleright Q \in BPPA_\Sigma$$

This composition mechanism denotes the behavior that first performs a and then either proceeds with P if **true** was produced or with Q otherwise.

For $a \in \Sigma$ and process $P \in BPPA_\Sigma$, we abbreviate the postconditional composition $P \trianglelefteq a \triangleright P$ by

$$a \circ P$$

and call this composition mechanism **action prefix**.

Thus all processes in $BPPA_\Sigma$ are made from S and D by means of a finite number of applications of postconditional composition. This suggests the existence of a partial ordering and an operator which finitely approximates every basic process.

Definition 2. 1. Let $\sqsubseteq$ be the partial ordering on $BPPA_\Sigma$ generated by the clauses

- a) for all $P \in BPPA_\Sigma$, $D \sqsubseteq P$, and
- b) for all $P, Q, X, Y \in BPPA_\Sigma$, $a \in \Sigma$,

$$P \sqsubseteq X \ \& \ Q \sqsubseteq Y \Rightarrow P \trianglelefteq a \triangleright Q \sqsubseteq X \trianglelefteq a \triangleright Y.$$

2. Let $\pi : \mathbb{N} \times BPPA_\Sigma \rightarrow BPPA_\Sigma$ be the approximation operator determined by the equations

- a) for all $P \in BPPA_\Sigma$, $\pi(0, P) = D$,
- b) for all $n \in \mathbb{N}$, $\pi(n+1, S) = S$, $\pi(n+1, D) = D$, and
- c) for all $P, Q \in BPPA_\Sigma$, $n \in \mathbb{N}$,

$$\pi(n+1, P \trianglelefteq a \triangleright Q) = \pi(n, P) \trianglelefteq a \triangleright \pi(n, Q).$$

We shall write $\pi_n(P)$ instead of $\pi(n, P)$.

π finitely approximates every process in $BPPA_\Sigma$. That is,

Proposition 1. For all $P \in BPPA_\Sigma$,

$$\exists n \in \mathbb{N} \ \pi_0(P) \sqsubseteq \pi_1(P) \sqsubseteq \cdots \sqsubseteq \pi_n(P) = \pi_{n+1}(P) = \cdots = P.$$

² Inaction typically occurs in case an infinite number of consecutive jumps is performed; for instance $(\#1)^\infty$.

Proof. We employ structural induction. If $P = D$ or $P = S$ then n can be taken 0 or 1, respectively. If $P = P_1 \trianglelefteq a \triangleright P_2$ let $n, m \in \mathbb{N}$ be such that $\pi_0(P_1) \sqsubseteq \pi_1(P_1) \sqsubseteq \cdots \sqsubseteq \pi_n(P_1) = \pi_{n+1}(P_1) = \cdots = P_1$ and $\pi_0(P_2) \sqsubseteq \pi_1(P_2) \sqsubseteq \cdots \sqsubseteq \pi_m(P_2) = \pi_{m+1}(P_2) = \cdots = P_2$. Thus for $k = \max\{n, m\}$ we have

$$\begin{aligned} \pi_0(P_1) \trianglelefteq a \triangleright \pi_0(P_2) &\sqsubseteq \pi_1(P_1) \trianglelefteq a \triangleright \pi_1(P_2) \\ &\vdots \\ &\sqsubseteq \pi_k(P_1) \trianglelefteq a \triangleright \pi_k(P_2) \\ &= \pi_{k+1}(P_1) \trianglelefteq a \triangleright \pi_{k+1}(P_2) \\ &\vdots \\ &= P_1 \trianglelefteq a \triangleright P_2. \end{aligned}$$

Hence $\pi_0(P) \sqsubseteq \pi_1(P) \sqsubseteq \cdots \sqsubseteq \pi_{k+1}(P) = \pi_{k+2}(P) = \cdots = P$.

Polarized processes can be finite or infinite. Following the metric process theory of [7] in the form developed as the basis of the introduction of processes in [1], BPPA_Σ has a completion $\text{BPPA}_\Sigma^\infty$ which comprises also the infinite processes. Standard properties of the completion technique yield that we may take $\text{BPPA}_\Sigma^\infty$ as consisting of all so-called *projective* sequences.

Recall that a *directed* set is a non-empty, partially ordered set which contains for any pair of its elements an upper bound. A *complete partial order (cpo)* is a partially ordered set with a least element such that every directed subset has a supremum. Let $C_0, C_1, \dots$ be a countable sequence of cpo's and let $f_i : C_{i+1} \rightarrow C_i$ be continuous for every $i \in \mathbb{N}$. The sequence (C_i, f_i) is called a *projective* (or *inverse*) system of cpo's. The *projective* (or *inverse*) *limit* of the system (C_i, f_i) is the poset $(C^\infty, \sqsubseteq)$ with

$$C^\infty = \{(x_i)_{i \in \mathbb{N}} \mid \forall i \in \mathbb{N} \ x_i \in C_i \ \& \ f_i(x_{i+1}) = x_i\}$$

and

$$(x_i)_{i \in \mathbb{N}} \sqsubseteq (y_i)_{i \in \mathbb{N}} \Leftrightarrow \forall i \in \mathbb{N} \ x_i \sqsubseteq y_i.$$

A fundamental theorem of domain theory states that C^∞ is a cpo with

$$\bigsqcup X = (\bigsqcup_{x \in X} x)_{i \in \mathbb{N}}$$

for directed $X \subseteq C^\infty$. If in addition there are continuous mappings $g_i : C_i \rightarrow C_{i+1}$ such that for every $i \in \mathbb{N}$

$$f_i(g_i(x)) = x \text{ and } g_i(f_i(x)) \sqsubseteq x$$

then, up to isomorphism, $C_i \subseteq C^\infty$. The isomorphism $h_i : C_i \rightarrow C^\infty$ can be given by

$$h_i(x) = \langle f_0(f_1 \cdots, f_{i-1}(x) \cdots), \cdots f_{i-1}(x), x, g_i(x), g_{i+1}(g_i(x)), \cdots \rangle.$$

Hence, up to isomorphism, $\bigcup_{i \in \mathbb{N}} C_i \subseteq C^\infty$. For a detailed account of this construction consult e.g. [11].

Definition 3.

1. For all $n \in \mathbb{N}$, $BPPA_\Sigma^n = \{\pi_n(P) \mid P \in BPPA_\Sigma\}$
2. $BPPA_\Sigma^\infty = \{(P_n)_{n \in \mathbb{N}} \mid \forall n \in \mathbb{N} (P_n \in BPPA_\Sigma^n \ \& \ \pi_n(P_{n+1}) = P_n)\}$

Lemma 1. *Let $(C, \sqsubseteq)$ be a finite directed set. Then C has a maximal element.*

Proof. Say $C = \{c_0, c_1, \dots, c_n\}$. If $n = 0$, c_0 is maximal. Otherwise pick $x_0 \in C$ such that $c_0, c_1 \sqsubseteq x_0$ and for $1 \leq i \leq n-1$ pick $x_i \in C$ such that $x_{i-1}, c_{i+1} \sqsubseteq x_i$. $x_0, x_1, \dots, x_{n-1}$ exist since C is directed. Now notice that x_{n-1} is the maximal element.

Proposition 2. *For all $n \in \mathbb{N}$,*

1. $BPPA_\Sigma^n$ is a cpo,
2. π_n is continuous,
3. for all $P \in BPPA_\Sigma$,
 - a) $\pi_n(P) \sqsubseteq P$,
 - b) $\pi_n(\pi_n(P)) = \pi_n(P)$, and
 - c) $\pi_{n+1}(\pi_n(P)) = \pi_n(P)$.

Proof.

1. We prove by induction on n that every directed set $X \subseteq BPPA_\Sigma^n$ is finite. It then follows from the previous lemma that suprema exist: they are the maximal elements. The base case is trivial since $BPPA_\Sigma^0 = \{D\}$. Now consider any directed $X \subseteq BPPA_\Sigma^{n+1}$. We distinguish two cases.
 - a) $S \in X$: Then $X \subseteq \{\bar{D}, S\}$. Thus X is finite.
 - b) $S \notin X$: Since X is directed there exists a unique $a \in \Sigma$ such that $X \subseteq \{D, \pi_n(P) \trianglelefteq a \triangleright \pi_n(Q) \mid P, Q \in BPPA_\Sigma\}$. Now let $X_1 = \{D, \pi_n(P) \mid \exists Q \in BPPA_\Sigma \ \pi_n(P) \trianglelefteq a \triangleright \pi_n(Q) \in X\}$ and $X_2 = \{D, \pi_n(Q) \mid \exists P \in BPPA_\Sigma \ \pi_n(P) \trianglelefteq a \triangleright \pi_n(Q) \in X\}$. Since X is directed it follows that both X_1 and X_2 are directed and hence finite by the induction hypothesis. Thus X is finite.
2. Since directed subsets are finite it suffices to show that π_n is monotone. Let $P \sqsubseteq Q \in BPPA_\Sigma$. We employ again induction on n . π_0 is constant and thus monotone. For $n+1$ we distinguish three cases.
 - a) $P = D$: Then $\pi_{n+1}(P) = D \sqsubseteq \pi_{n+1}(Q)$.
 - b) $P = S$: Then also $Q = S$. Hence $\pi_{n+1}(P) = \pi_{n+1}(Q)$.
 - c) $P = P_1 \trianglelefteq a \triangleright P_2$: Then $Q = Q_1 \trianglelefteq a \triangleright Q_2$ with $P_i \sqsubseteq Q_i$ for $i \in \{1, 2\}$. From the monotonicity of π_n it now follows that $\pi_n(P_i) \sqsubseteq \pi_n(Q_i)$ for $i \in \{1, 2\}$. Thus $\pi_{n+1}(P) \sqsubseteq \pi_{n+1}(Q)$.
3. Let $P \in BPPA_\Sigma$. (a) follows from Proposition 1. We prove (b) and (c) simultaneously by induction on n . For $n = 0$ we have $\pi_0(\pi_0(P)) = D = \pi_0(P)$ and $\pi_1(\pi_0(P)) = D = \pi_0(P)$. Now consider $n+1$. We distinguish two cases.

- a) $P \in \{D, S\}$: Then $\pi_{n+1}(\pi_{n+1}(P)) = P = \pi_{n+1}(P)$ and $\pi_{n+2}(\pi_{n+1}(P)) = P = \pi_{n+1}(P)$.
 b) $P = P_1 \trianglelefteq a \trianglerighteq P_2$: Then it follows from the induction hypothesis that

$$\begin{aligned}\pi_{n+1}(\pi_{n+1}(P)) &= \pi_n(\pi_n(P_1)) \trianglelefteq a \trianglerighteq \pi_n(\pi_n(P_2)) \\ &= \pi_n(P_1) \trianglelefteq a \trianglerighteq \pi(P_2) = \pi_{n+1}(P)\end{aligned}$$

and

$$\begin{aligned}\pi_{n+2}(\pi_{n+1}(P)) &= \pi_{n+1}(\pi_n(P_1)) \trianglelefteq a \trianglerighteq \pi_{n+1}(\pi_n(P_2)) \\ &= \pi_n(P_1) \trianglelefteq a \trianglerighteq \pi(P_2) = \pi_{n+1}(P).\end{aligned}$$

Theorem 1. $BPPA_\Sigma^\infty$ is a cpo and, up to isomorphism, $BPPA_\Sigma \subseteq BPPA_\Sigma^\infty$.

Proof. 1. and 2. of the previous proposition show that $(BPPA_\Sigma^n, \pi_n)$ is a projective system of cpo's. Thus $BPPA_\Sigma^\infty$ is a cpo. Note that it follows from 3(c) that $BPPA_\Sigma^n \subseteq BPPA_\Sigma^{n+1}$ for all n . Thus if we define for all P and n , $id_n(P) = P$ then $id_n : BPPA_\Sigma^n \rightarrow BPPA_\Sigma^{n+1}$ for all n . id_n is clearly continuous. Moreover, 3(a) yields $\pi_n(id_n(P)) \sqsubseteq P$ for all n and $P \in BPPA_\Sigma^n$. Likewise, 3(b) yields $id_n(\pi_n(P)) = P$ for all n and $P \in BPPA_\Sigma^{n+1}$. Thus, up to isomorphism, $\bigcup_{n \in \mathbb{N}} BPPA_\Sigma^n \subseteq BPPA_\Sigma^\infty$. Thus also $BPPA_\Sigma \subseteq BPPA_\Sigma^\infty$ since $BPPA_\Sigma = \bigcup_n BPPA_\Sigma^n$ by Proposition 1.

The set of polarized processes can serve in a natural fashion as a semantics for programs. As an example we shall consider PGA_Σ .

3 Program Algebra

Given a collection Σ of atomic instructions the syntax of program expressions (or programs) in PGA_Σ is generated from five kinds of constants and two composition mechanisms. The constants are made from Σ together with a termination instruction, two test instructions and a forward jump instruction. As in the case of BPPA, the atomic instructions may be viewed as requests to an environment to provide some service. It is assumed that upon every termination of the delivery of that service some boolean value is returned that may be used for subsequent program control. The two composition mechanisms are concatenation and infinite repetition.

Definition 4. For a collection Σ of atomic instructions, PGA_Σ denotes the collection of program expressions inductively defined by

termination: $! \in PGA_\Sigma$

The instruction $!$ indicates termination of the program and will not return any value.

forward jump instruction: $\#n \in PGA_\Sigma$ for every $n \in \mathbb{N}$

n counts how many subsequent instructions must be skipped, including the jump instruction itself.

void basic instruction: $a \in PGA_{\Sigma}$ for every $a \in \Sigma$

positive test instruction: $+a \in PGA_{\Sigma}$ for every $a \in \Sigma$

The execution of $+a$ begins with executing a . Thereafter, if **true** is replied, program execution continues with the execution of the next instruction following the positive test instruction in the program. Otherwise, if **false** is replied, the instruction immediately following the (positive) test instruction is skipped and program execution continues with the instruction thereafter.

negative test instruction: $-a \in PGA_{\Sigma}$ for every $a \in \Sigma$

The negative test instruction $(-a)$ reacts the other way around on the boolean values it receives as a feedback from its operating context. At a positive (**true**) reply it skips the next action, and at a negative reply it simply continues.

concatenation: For programs $X, Y \in PGA_{\Sigma}$, $X; Y \in PGA_{\Sigma}$

repetition: For a program $X \in PGA_{\Sigma}$, $X^{\omega} \in PGA_{\Sigma}$

Here are some program examples:

$$+a; !; +b; \#3; c; !; d; !$$

$$a; !; -b; \#3; c; \#0; d; !$$

$$-a; !; (-b; \#3; c; \#0; +d; !)^{\omega}.$$

The simplest model of the signature of program algebra interprets each term as a sequence of primitive instructions. This is the *instruction sequence model*. Equality within this model will be referred to as instruction sequence congruence ($=_{\text{isc}}$). Two programs X and Y are instruction sequence congruent if both denote the same sequence of instructions after unfolding the repetition operator, that is, if they can be shown to be equal by means of the program object equations in Table 1.

Table 1. Program object equations

$(X; Y); Z = X; (Y; Z)$	(PGA1)
$(X^n)^{\omega} = X^{\omega}$	(PGA2)
$X^{\omega}; Y = X^{\omega}$	(PGA3)
$(X; Y)^{\omega} = X; (Y; X)^{\omega}$	(PGA4)

Here $X^1 = X$ and $X^{n+1} = X; X^n$. The associativity of concatenation implies as usual that far fewer brackets have to be used. We will use associativity whenever confusion cannot emerge.

The program object equations allow some useful transformations, in particular the transformation into *first canonical form*.

Definition 5. Let $X \in PGA_\Sigma$. Then X is in first canonical form iff

1. X does not contain any repetition, or
2. $X = Y; Z^\omega$ with Y and Z not containing any repetition.

The existence of first canonical forms follows straightforwardly by structural induction. The key case is this:

$$\begin{aligned}
 (U; X^\omega)^\omega &=_{\text{isc}} (U; X^\omega; U; X^\omega)^\omega && \text{by PGA2} \\
 &=_{\text{isc}} (U; X^\omega); (U; X^\omega)^\omega && \text{by PGA4} \\
 &=_{\text{isc}} U; (X^\omega; (U; X^\omega)^\omega) && \text{by PGA1} \\
 &=_{\text{isc}} U; X^\omega && \text{by PGA3}
 \end{aligned}$$

First canonical forms need not be unique. For example, $a; a; a^\omega$ and $a; a; a; a^\omega$ are both canonical forms of $a; a^\omega$ which is already in canonical form itself. In the sequel we shall mean by *the* first canonical form the shortest one.

Definition 6. Let $X \in PGA_\Sigma$ be in first canonical form. The length of X , $l(X)$, is defined by

1. if X does not contain any repetition then $l(X) = (n, 0)$ where n is the number of instructions in X , and
2. if $X = Y; Z^\omega$ with both Y and Z not containing any repetition then $l(X) = (n, m)$ where n and m are the number of instructions in Y and Z , respectively.

Observe that $\mathbb{N} \times \mathbb{N}$ is a well-founded partial order by stipulating

$$(n_0, n_1) \leq (m_0, m_1) \Leftrightarrow n_0 \leq m_0 \text{ or } (n_0 = m_0 \text{ and } n_1 \leq m_1).$$

Definition 7. Let $X \in PGA_\Sigma$. The first canonical form of X , $cf(X)$, is a first canonical form X' with $X =_{\text{isc}} X'$ and minimal length, i.e. for all first canonical forms X'' with $X =_{\text{isc}} X''$, $l(X') \leq l(X'')$. We call X finite if $l(cf(X)) = (n, 0)$ and infinite if $l(cf(X)) = (n, m + 1)$ for some $n, m \in \mathbb{N}$.

Clearly $cf(X)$ is well-defined, that is, there exists a unique shortest first canonical form of X .

A second model of program algebra is $BPPA_\Sigma^\infty$. As a prerequisite we define a mapping $| \cdot |$ from finite programs, i.e. programs without repetition, to finite polarized processes. Prior to a formal definition some examples are of use:

$$|a; b; !| = a \circ (b \circ S)$$

$$|a; +b; !; \#0| = a \circ (S \trianglelefteq b \triangleright D)$$

$$|+ a; !| = S \trianglelefteq a \triangleright D.$$

The intuition behind the mapping to processes is as follows: view a program as an instruction sequence and turn that into a process from left to right. The mapping into processes removes all control aspects (tests, jumps) in favor of an unfolding of all possible behaviors. A forward jump instruction with counter zero jumps to itself, thereby creating a loop or divergence (D). Only via $!$ the proper termination (S) will take place. If the program is exited in another way this also counts as a divergence (D).

In the sequel we let $u, u_1, u_2, \dots$ range over $\{!, \#k, a, +a, -a \mid a \in \Sigma, k \in \mathbb{N}\}$.

Definition 8. Let $X \in \text{PGA}_\Sigma$ be finite. Then $|X|$ is defined by induction on its length $l(X)$.

1. $l(X) = (1, 0)$:
 - a) If $X = !$ then $|X| = S$,
 - b) if $X = \#k$ then $|X| = D$, and
 - c) if $X \in \{a, +a, -a\}$ then $|X| = a \circ D$.
2. $l(X) = (n + 2, 0)$:
 - a) if $X = !; Y$ then $|X| = S$,
 - b) if $X = \#0; Y$ then $|X| = D$,
 - c) if $X = \#1; Y$ then $|X| = |Y|$,
 - d) if $X = \#k + 2; u; Y$ then $|X| = |\#k + 1; Y|$,
 - e) if $X = a; Y$ then $|X| = a \circ |Y|$;
 - f) if $X = +a; Y$ then $|X| = |Y| \trianglelefteq a \triangleright |\#2; Y|$, and
 - g) if $X = -a; Y$ then $|X| = |\#2; Y| \trianglelefteq a \triangleright |Y|$.

Observe that $| \cdot |$ is monotone in continuations. That is,

Proposition 3. Let $X = u_1; \dots; u_n$ and $Y = u_1; \dots; u_n; \dots; u_{n+k}$. Then $|X| \sqsubseteq |Y|$.

Proof. Straightforward by induction on n and case ramification. E.g. if $n = 1$ and $X \in \{a, +a, -a\}$ then $|X| = a \circ D$ and $|Y| = |Z| \trianglelefteq a \triangleright |Z'|$ for some $Z, Z' \in \text{PGA}_\Sigma$. Thus $|X| \sqsubseteq |Y|$. If $n > 1$ consider e.g. the case where $X = \#k + 2; u_2; \dots; u_n$. Then $|X| = |\#k + 1; u_3; \dots; u_n| \sqsubseteq |\#k + 1; u_3; \dots; u_n; \dots; u_{n+k}| = |Y|$ by the induction hypothesis. Etc.

It follows that for repetition-free Y and Z , $|Y; Z| = |Y; Z^1| \sqsubseteq |Y; Z^2| \sqsubseteq |Y; Z^3| \sqsubseteq \dots$ is an ω -chain and hence directed. Thus $\bigsqcup_{n \in \mathbb{N}} |Y; Z^n|$ exists in $\text{BPPA}_\Sigma^\infty$. We can now extend Definition 8 to infinite processes.

Definition 9. Let $Y; Z^\omega \in \text{PGA}_\Sigma$ be in first canonical form. Then $|Y; Z^\omega| = \bigsqcup_{n \in \mathbb{N}} |Y; Z^n|$.

Moreover, for arbitrary programs we define

Definition 10.

Let $X \in \text{PGA}_\Sigma$. Then $\llbracket X \rrbracket = |cf(X)|$.

As an example consider:

$$\begin{aligned}
\llbracket + a; \#3; !; (b; c)^\omega \rrbracket &= \bigsqcup_{n \in \mathbb{N}} \llbracket + a; \#3; !; (b; c)^n \rrbracket \\
&= \bigsqcup_{n \in \mathbb{N}} \llbracket \#3; !; (b; c)^n \rrbracket \trianglelefteq a \triangleq \bigsqcup_{n \in \mathbb{N}} \llbracket \#2; \#3; !; (b; c)^n \rrbracket \\
&= \bigsqcup_{n \in \mathbb{N}} \llbracket \#2; (b; c)^n \rrbracket \trianglelefteq a \triangleq \bigsqcup_{n \in \mathbb{N}} \llbracket \#1; !; (b; c)^n \rrbracket \\
&= \bigsqcup_{n \in \mathbb{N}} \llbracket \#1; (c; b)^n \rrbracket \trianglelefteq a \triangleq \bigsqcup_{n \in \mathbb{N}} \llbracket !; (b; c)^n \rrbracket \\
&= \bigsqcup_{n \in \mathbb{N}} \llbracket (c; b)^n \rrbracket \trianglelefteq a \triangleq \bigsqcup_{n \in \mathbb{N}} \llbracket !; (b; c)^n \rrbracket \\
&= c \circ b \circ c \circ b \circ \dots \trianglelefteq a \triangleq S
\end{aligned}$$

Since instruction sequence congruent programs have identical *cf*-canonical forms we have

Theorem 2. *For all $X, Y \in PGA_\Sigma$, $X =_{\text{isc}} Y \Rightarrow \llbracket X \rrbracket = \llbracket Y \rrbracket$.*

The converse does not hold: e.g. $\#1; ! \neq_{\text{isc}} !$ but $\llbracket \#1; ! \rrbracket = S = \llbracket ! \rrbracket$.

Further models for program algebra will be found by imposing congruences on the instruction sequence model. Two congruences will be used: behavioral congruence and structural congruence.

4 Behavioral and Structural Congruence

X and Y are *behaviorally equivalent* if $\llbracket X \rrbracket = \llbracket Y \rrbracket$. Behavioral equivalence is not a congruence. For instance $\llbracket !; ! \rrbracket = S = \llbracket !; \#0 \rrbracket$ but $\llbracket \#2; !; ! \rrbracket = S \neq D = \llbracket \#2; !; \#0 \rrbracket$. This motivates the following definition.

Definition 11.

1. *The set of PGA-contexts is $\mathcal{C} ::= \_ \mid Z; \mathcal{C} \mid \mathcal{C}; Z \mid \mathcal{C}^\omega$.*
2. *Let $X, Y \in PGA_\Sigma$. X and Y are behaviorally congruent ($X =_{\text{bc}} Y$) if for all PGA $_\Sigma$ -contexts $C[\]$, $\llbracket C[X] \rrbracket = \llbracket C[Y] \rrbracket$.*

As a matter of fact it suffices to consider only one kind of context.

Theorem 3. *Let $X, Y \in PGA_\Sigma$. Then*

$$X =_{\text{bc}} Y \Leftrightarrow \forall Z, Z' \in PGA_\Sigma \llbracket Z; X; Z' \rrbracket = \llbracket Z; Y; Z' \rrbracket.$$

Proof. Left to right follows from the definition of behavioral congruence. In order to prove right to left observe first that—because of PGA3—we do not need to consider any contexts of the form $C[\]^\omega; Z'$ or $Z; C[\]^\omega; Z'$. The context we do have to consider are therefore the ones given in the table.

1.a	—	2.a	$-\omega$	3.a	$Z''; -\omega$
1.b	$Z; -$	2.b	$(Z; -)^\omega$	3.b	$Z''; (Z; -)^\omega$
1.c	$-; Z'$	2.c	$(-; Z')^\omega$	3.c	$Z''; (-; Z')^\omega$
1.d	$Z; -; Z'$	2.d	$(Z; -; Z')^\omega$	3.d	$Z''; (Z; -; Z')^\omega$

Assuming the right-hand side, we first show that for every context $C[\]$ in the first column we have $\llbracket C[X] \rrbracket = \llbracket C[Y] \rrbracket$. 1.d is obvious. 1.c follows by taking $Z = \#1$ in 1.d. Now observe that for every U , $\llbracket U; \#0 \rrbracket = \llbracket U \rrbracket$: for finite U this is shown easily with induction to the number of instructions, and for U involving repetition $\llbracket U; \#0 \rrbracket = \llbracket U \rrbracket$ follows from PGA3. This yields 1.a and 1.b by taking $Z' = \#0$ in 1.c. and 1.d, respectively. This covers all contexts in the first column.

We now turn to the third column. We shall first show that for all $n > 0$ and all Z'' , $\llbracket Z''; X^n \rrbracket = \llbracket Z''; Y^n \rrbracket$. The case $n = 1$ has just been established (1.b). Now consider $n + 1$: by taking $Z = Z''$ and $Z' = X^n$ in 1.d, $\llbracket Z''; X; X^n \rrbracket = \llbracket Z''; Y; X^n \rrbracket$. Moreover, from the induction hypothesis it follows that $\llbracket Z''; Y; X^n \rrbracket = \llbracket Z''; Y; Y^n \rrbracket$. Thus $\llbracket Z''; X^{n+1} \rrbracket = \llbracket Z''; Y^{n+1} \rrbracket$. From the limit characterization of repetition it now follows that $\llbracket Z''; X^\omega \rrbracket = \llbracket Z''; Y^\omega \rrbracket$ (3.a). 3.b is dealt with using the same argument with only a small notational overhead. For 3.c and 3.d observe that

$$\begin{aligned} \llbracket Z''; (X; Z')^\omega \rrbracket &= \llbracket Z''; X; (Z'; X)^\omega \rrbracket \\ &= \llbracket Z''; X; (Z'; Y)^\omega \rrbracket \\ &= \llbracket Z''; Y; (Z'; Y)^\omega \rrbracket \\ &= \llbracket Z''; (Y; Z')^\omega \rrbracket \end{aligned}$$

follows from PGA4, 3.b and 1.d, and

$$\begin{aligned} \llbracket Z''; (Z; X; Z')^\omega \rrbracket &= \llbracket Z''; Z; (X; Z'; Z)^\omega \rrbracket \\ &= \llbracket Z''; Z; (Y; Z'; Z)^\omega \rrbracket \\ &= \llbracket Z''; (Z; Y; Z')^\omega \rrbracket \end{aligned}$$

follows from PGA4 and 3.c. This covers all context in the third column.

Finally we consider the second column. Here every context can be dealt with by taking in the corresponding context in the third column $Z'' = \#1$.

Structural congruence is characterized by the four equation schemes in Table 2. The schemes take care of the simplification of chained jumps. The schemes are termed PGA5-8, respectively. PGA8 can be written as an equation by expanding X , but takes a more compact and readable form as a conditional equation. Program texts are considered structurally congruent if they can be proven equal by means of PGA1-8. Structural congruence of X and Y is indicated with $X =_{sc} Y$, omitting the subscript if no confusion arises. Some consequences of these axioms are

$$a; \#2; b; \#0; c = a; \#0; b; \#0; c$$

$$a; \#2; b; \#1; c = a; \#3; b; \#1; c$$

$$a; (\#3; b; c)^\omega = a; (\#0; b; c)^\omega$$

The purpose of structural congruence is to allow successive (and repeating) jumps to be taken together.

Table 2. Equation schemes for structural congruence

$\#n + 1; u_1; \dots; u_n; \#0 = \#0; u_1; \dots; u_n; \#0$	(PGA5)
$\#n + 1; u_1; \dots; u_n; \#m = \#n + m + 1; u_1; \dots; u_n; \#m$	(PGA6)
$(\#n + k + 1; u_1; \dots; u_n)^\omega = (\#k; u_1; \dots; u_n)^\omega$	(PGA7)
$X = u_1; \dots; u_n; (v_1; \dots; v_{m+1})^\omega \rightarrow$ $\#n + m + k + 2; X = \#n + k + 1; X$	(PGA8)

Structurally congruent programs are behaviorally congruent as well. This is proven by demonstrating the validity of each closed instance of the structural congruence equations modulo behavioral congruence.

5 The Entry Instruction

As it turns out behavioral congruence on PGA_Σ is not easy to axiomatize by means of equations or conditional equations. It remains an open problem how that can be done. Here the matter will be approached from another angle. First an additional primitive instruction is introduced: @, the entry instruction. The instruction @ in front of a program disallows any jumps into the program otherwise than jumps into the first instruction of the program. Longer jumps are discontinued, and the jump will be carried out as a jump to the control point following @. The entry instruction is new, in the sense that it coincides with no PGA_Σ program or primitive instruction. Its use lies in the fact that it allows an unexpected number of additional (conditional) equations for programs. As a consequence it becomes possible to find a concise final algebra specification of behavioral inequality of programs. This is plausible to some extent: it is much easier to see that programs differ, by finding input leading to different outputs, than to see that they don't differ and hence coincide in the behavioral congruence model of program algebra. The program notation extending PGA_Σ with '@' is denoted $\text{PGA}_{\Sigma, @}$.

In order to provide a mapping from $\text{PGA}_{\Sigma, @}$ into $\text{BPPA}_\Sigma^\infty$ we add to the clauses in Definition 8 the clauses 1.-4. of the following definition

Definition 12.

1. $|\text{@}| = D,$
2. $|\text{@}; X| = |X|,$
3. $|\#n + 1; \text{@}| = D,$
4. $|\#n + 1; \text{@}; X| = |X|,$

and change the clause 2d in Definition 8 into

$$(u \neq \text{@}) \Rightarrow |\#k + 2; u; X| = |\#k + 1; X|.$$

Using these additional rules $\llbracket \cdot \rrbracket$ can be defined straightforwardly for programs involving the entry instruction. Behavioral congruence has then exactly the same definition in the presence of the entry instruction and Theorem 3 extends trivially to $\text{PGA}_{\Sigma, @}$.

Because programs with different behavior may be considered observationally different it is reasonable to call $\text{PGA}_{\Sigma, @}/\equiv_{\text{bc}}$ a fully abstract model. It imposes a maximal congruence under the constraint that observationally different programs will not be identified.

A characterization of behavioral congruence in terms of behavioral equivalence will be given in Theorem 4. The intuition behind this characterization is that behavior extraction abstracts from two aspects that can be recovered by taking into account the influence of a context: the instruction that serves as initial instruction (which for $\llbracket u_1; \dots; u_n; \dots \rrbracket$ is always u_1) and the difference between divergence and exiting a program with some jump. To make these differences visible at the level of program behaviors only very simple contexts are needed: here are three examples (where $a \neq b$):

$$\#2 \not\equiv_{\text{bc}} \#1 \text{ because } \llbracket \#2; !; \#0^\omega \rrbracket = D \neq S = \llbracket \#1; !; \#0^\omega \rrbracket,$$

$$\#2; a \not\equiv_{\text{bc}} \#2; b \text{ because } \llbracket \#2; \#2; a \rrbracket = a \circ D \neq b \circ D = \llbracket \#2; \#2; b \rrbracket.$$

$$!; \#1 \not\equiv_{\text{bc}} !; \#2 \text{ because } \llbracket \#2; !; \#1; !; \#0^\omega \rrbracket = S \neq D = \llbracket \#2; !; \#2; !; \#0^\omega \rrbracket.$$

Theorem 4. *Let $X, Y \in \text{PGA}_{\Sigma, @}$. Then*

1. $X \equiv_{\text{bc}} Y \Leftrightarrow \forall n \in \mathbb{N} \forall Z' \in \text{PGA}_{\Sigma, @} \llbracket \#n + 1; X; Z' \rrbracket = \llbracket \#n + 1; Y; Z' \rrbracket$
2. $X \equiv_{\text{bc}} Y \Leftrightarrow \forall n, m \in \mathbb{N} \llbracket \#n + 1; X; !^m; \#0^\omega \rrbracket = \llbracket \#n + 1; Y; !^m; \#0^\omega \rrbracket$

Proof. Left to right follows for 1. and 2. from the definition of behavioral congruence.

1. Assume the right-hand side. We employ Theorem 3. Suppose that for some Z, Z' , $\llbracket Z; X; Z' \rrbracket \neq \llbracket Z; Y; Z' \rrbracket$. Then Z cannot contain an infinite repetition. Therefore it is finite. With induction on the length of Z one then proves the existence of a natural number k such that $\llbracket \#k + 1; X; Z' \rrbracket \neq \llbracket \#k + 1; Y; Z' \rrbracket$. For $l(Z) = (1, 0)$ we distinguish 6 cases:
 - a) $Z = !$: Then $\llbracket Z; X; Z' \rrbracket = S = \llbracket Z; Y; Z' \rrbracket$. Contradiction.
 - b) $Z = @$: Then $\llbracket X; Z' \rrbracket \neq \llbracket Y; Z' \rrbracket$. Thus also $\llbracket \#1; X; Z' \rrbracket \neq \llbracket \#1; Y; Z' \rrbracket$.
 - c) $Z = \#n$: As n cannot be 0 we are done.
 - d) $Z = a$: Then $a \circ \llbracket X; Z' \rrbracket \neq a \circ \llbracket Y; Z' \rrbracket$. Thus $\llbracket X; Z' \rrbracket \neq \llbracket Y; Z' \rrbracket$ and hence $\llbracket \#1; X; Z' \rrbracket \neq \llbracket \#1; Y; Z' \rrbracket$.
 - e) $Z \in \{+a, -a\}$: If $Z = +a$ then

$$\llbracket X; Z' \rrbracket \trianglelefteq a \triangleright \llbracket \#2; X; Z' \rrbracket \neq \llbracket Y; Z' \rrbracket \trianglelefteq a \triangleright \llbracket \#2; Y; Z' \rrbracket.$$

Then $\llbracket X; Z' \rrbracket \neq \llbracket Y; Z' \rrbracket$ or $\llbracket \#2; X; Z' \rrbracket \neq \llbracket \#2; Y; Z' \rrbracket$. In the latter case we are done and in the first case we can take $k = 0$. $-a$ is dealt with similarly.

Now consider $l(Z) = (m+2, 0)$. We have to distinguish 10 cases. Seven cases correspond to the repetition-free clauses in 2 of Definition 8. They follow from a straightforward appeal to the induction hypothesis. The remaining three cases correspond to 2.-4. of Definition 12.

- a) $Z = @; Z''$: Then $\llbracket Z''; X; Z' \rrbracket \neq \llbracket Z''; Y; Z' \rrbracket$. Hence $\llbracket \#k+1; X; Z' \rrbracket \neq \llbracket \#k+1; Y; Z' \rrbracket$ for some k by the induction hypothesis.
 - b) $Z = \#n+1; @$: Then $\llbracket X; Z' \rrbracket \neq \llbracket Y; Z' \rrbracket$. Hence $\llbracket \#1; X; Z' \rrbracket \neq \llbracket \#1; Y; Z' \rrbracket$.
 - c) $Z = \#n+1; @; Z''$: Then $\llbracket Z''; X; Z' \rrbracket \neq \llbracket Z''; Y; Z' \rrbracket$ and we can again apply the induction hypothesis.
2. Assume the right-hand side. We make an appeal to 1. Suppose there are k and Z' such that $\llbracket \#k+1; X; Z' \rrbracket \neq \llbracket \#k+1; Y; Z' \rrbracket$. If both X and Y are infinite then $\llbracket \#k+1; X \rrbracket \neq \llbracket \#k+1; Y \rrbracket$ and hence also $\llbracket \#k+1; X; \#0^\omega \rrbracket \neq \llbracket \#k+1; Y; \#0^\omega \rrbracket$. Suppose only one of the two, say Y , has a repetition, then writing $X = u_1; \dots; u_n$, it follows that: $\llbracket \#k+1; u_1; \dots; u_n; Z' \rrbracket \neq \llbracket \#k+1; Y \rrbracket$. At this point an induction on n can be used to establish the existence of an m with $\llbracket \#k+1; u_1; \dots; u_n; !^m; \#0^\omega \rrbracket \neq \llbracket \#k+1; Y \rrbracket$ and hence $\llbracket \#k+1; u_1; \dots; u_n; !^m; \#0^\omega \rrbracket \neq \llbracket \#k+1; Y; !^m; \#0^\omega \rrbracket$. If both X and Y are finite instruction sequences, an induction on their maximum length suffices to obtain the required fact (again involving a significant case ramification).

Example 1.

1. $@; ! =_{bc} !^\omega$ since for all n, Z , $\llbracket \#n+1; @; !; Z \rrbracket = \llbracket !; Z \rrbracket = S = \llbracket \#n+1; !^\omega; Z \rrbracket$, and
2. $@; \#0 =_{bc} \#0^\omega$ since for all n, Z , $\llbracket \#n+1; @; \#0; Z \rrbracket = \llbracket \#0; Z \rrbracket = D = \llbracket \#n+1; \#0^\omega; Z \rrbracket$.

The characterization above suggests that behavioral congruence may be undecidable. This of course is not the case: the quantifier over m can be bounded because m need not exceed the maximum of the counters of jump instructions in X and Y plus 1. An upper bound for n is as follows: if $l(X) = (k, m)$ and $l(Y) = (k', m')$ then $(k+m) \times (k'+m')$ is an upper bound of the n 's that must be checked.

Programs starting with the entry instruction can be distinguished by means of simpler contexts:

Corollary 1. *Let $X, Y \in PGA_{\Sigma, @}$. Then*

1. $@; X =_{bc} @; Y \Leftrightarrow \forall n \in \mathbb{N} \llbracket X; !^n; \#0^\omega \rrbracket = \llbracket Y; !^n; \#0^\omega \rrbracket$
2. $@; X =_{bc} @; Y \Leftrightarrow \forall Z \llbracket X; Z \rrbracket = \llbracket Y; Z \rrbracket$

Proof. 1. and 2. follow from that fact that for every $n, k \in \mathbb{N}$ and every X , $\llbracket \#k+1; @; X; !^n; \#0^\omega \rrbracket = \llbracket X; !^n; \#0^\omega \rrbracket$ and $\llbracket \#k+1; @; X; Z \rrbracket = \llbracket X; Z \rrbracket$.

Since $\llbracket X \rrbracket = \llbracket X; \#0^\omega; Z \rrbracket$ for all program expressions X and Z , it follows from Corollary 1.2 that behavioral equivalence can be recovered from behavioral congruence in the following way:

Corollary 2. *Let $X, Y \in PGA_{\Sigma, @}$. Then*

$$X =_{\text{be}} Y \Leftrightarrow @; X; \#0^\omega =_{\text{bc}} @; Y; \#0^\omega.$$

Programs ending with an entry instruction allow a simpler characterisation as well:

Corollary 3. *Let $X, Y \in PGA_{\Sigma, @}$. Then $X; @ =_{\text{bc}} Y; @$ iff for all $n \in \mathbb{N}$,*

$$\llbracket \#n + 1; X; !^\omega \rrbracket = \llbracket \#n + 1; Y; !^\omega \rrbracket \ \& \ \llbracket \#n + 1; X; \#0^\omega \rrbracket = \llbracket \#n + 1; Y; \#0^\omega \rrbracket$$

Proof. ‘ $\Rightarrow$ ’: Suppose that $X; @ =_{\text{bc}} Y; @$, then for all n and m ,

$$(\star) \quad \llbracket \#n + 1; X; @; !^m; \#0^\omega \rrbracket = \llbracket \#n + 1; Y; @; !^m; \#0^\omega \rrbracket.$$

Then

$$\begin{aligned} \llbracket \#n + 1; X; !^\omega \rrbracket &= \llbracket \#n + 1; X; !^\omega; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; X; @; !; \#0^\omega \rrbracket \text{ since } @; ! =_{\text{bc}} !^\omega \text{ (Example 1)} \\ &= \llbracket \#n + 1; Y; @; !; \#0^\omega \rrbracket \text{ take in } (\star) \ m = 1 \\ &= \llbracket \#n + 1; Y; !^\omega; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; Y; !^\omega \rrbracket \end{aligned}$$

Similarly

$$\begin{aligned} \llbracket \#n + 1; X; \#0^\omega \rrbracket &= \llbracket \#n + 1; X; \#0^\omega; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; X; @; \#0; \#0^\omega \rrbracket \text{ since } @; \#0 =_{\text{bc}} \#0^\omega \text{ (Example 1)} \\ &= \llbracket \#n + 1; X; @; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; Y; @; \#0^\omega \rrbracket \text{ take in } (\star) \ m = 0 \\ &= \llbracket \#n + 1; Y; @; \#0; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; Y; \#0^\omega; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; Y; \#0^\omega \rrbracket \end{aligned}$$

‘ $\Leftarrow$ ’: for $m = 0$, the above argument runs in the other direction

$$\begin{aligned} \llbracket \#n + 1; X; @; !^0; \#0^\omega \rrbracket &= \llbracket \#n + 1; X; @; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; X; @; \#0; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; X; \#0^\omega; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; Y; \#0^\omega; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; Y; @; \#0; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; Y; @; \#0^\omega \rrbracket \\ &= \llbracket \#n + 1; Y; @; !^0; \#0^\omega \rrbracket \end{aligned}$$

The case $m > 0$ is similar.

6 Axiomatization of the Fully Abstract Model

With $\text{CEQ}_@$ the collection of 20 equations and inequations in Table 3 will be denoted (CEQ for, ‘conditional and unconditional equations’). They can be viewed

Table 3. $CEQ_{@}$

(1)	$@; ! = !^{\omega}$
(2)	$@; \#0 = \#0^{\omega}$
(3)	$@; @ = @$
(4)	$\#n + 1; @ = @$
(5)	$+a; @ = a; @$
(6)	$-a; @ = a; @$
(7)	$\#n + l + 1; u_1; \dots; u_n; @ = \#n + 1; u_1; \dots; u_n; @$
(8)	$@; u_1; \dots; u_n; @ = @; u_1; \dots; u_n; \#1 \quad (\forall 1 \leq j \leq n \ u_j = \#k \Rightarrow k + j \leq n + 1)$
(9)	$@; u_1; \dots; u_n; @ = @; u_1; \dots; u_n; \#1 \Rightarrow @; (u_1; \dots; u_n; @)^{\omega} = @; (u_1; \dots; u_n; \#1)^{\omega}$
(10)	$@; \#1 = @$
(11)	$@; \#n + 2; u = @; \#n + 1 \quad (\text{if } u \neq @)$
(12)	$@; a; @ = @; a$
(13)	$@; a = @; +a; \#1$
(14)	$@; -a = @; +a; \#2$
(15)	$@; X = @; Y \ \& \ @; \#2; X = @; \#2; Y \Leftrightarrow @; +a; X = @; +a; Y$
(16)	$@; u; X = @; v; X \Rightarrow u; X = v; X$
(17)	$@; ! \neq @; \#j$
(18)	$@; ! \neq @; +a; X$
(19)	$@; \#0 \neq @; +a; X$
(20)	$@; +a; X \neq @; +b; Y \quad (a \neq b \in \Sigma)$

as axioms from which other facts may be derived using conditional equational logic. Inequations can be understood as a shorthand for conditional equation: e.g. $@; ! = @; \#j \Rightarrow X = Y$ represents $@; ! \neq @; \#j$. No attempt has been made to minimize or optimize this collection. We shall first show that $CEQ_{@}$ is valid in $PGA_{\Sigma, @} / =_{bc}$.

Proposition 4. $PGA_{\Sigma, @} / =_{bc} \models CEQ_{@}$

Proof. 1. See Example 1.1.

2. See Example 1.2.

3. Since $\llbracket @; @; Z \rrbracket = \llbracket @; Z \rrbracket$ for all Z , we can apply Corollary 1.2.

4. If $k = 0$, $\llbracket \#k + 1; \#n + 1; @; Z \rrbracket = \llbracket \#1; \#n + 1; @; Z \rrbracket = \llbracket \#n + 1; @; Z \rrbracket = \llbracket @; Z \rrbracket = \llbracket \#k + 1; @; Z \rrbracket$ and if $k > 0$ $\llbracket \#k + 1; \#n + 1; @; Z \rrbracket = \llbracket \#k; @; Z \rrbracket = \llbracket @; Z \rrbracket = \llbracket \#k + 1; @; Z \rrbracket$. Now apply Theorem 4.1.

5. We apply again Theorem 4.1. For $k > 0$ the process extraction equations match both sides. For $k = 0$ we obtain: $\llbracket \#1; +a; @; Z \rrbracket = \llbracket +a; @; Z \rrbracket = \llbracket @; Z \rrbracket \triangleleft a \triangleright \llbracket \#2; @; Z \rrbracket = \llbracket @; Z \rrbracket \triangleleft a \triangleright \llbracket @; Z \rrbracket = a \circ \llbracket @; Z \rrbracket = \llbracket a; @; Z \rrbracket = \llbracket \#1; a; @; Z \rrbracket$. For $k > 0$ we have

$$\llbracket \#k + 1; +a; @; Z \rrbracket = \llbracket \#k; @; Z \rrbracket = \llbracket \#k + 1; a; @; Z \rrbracket.$$

6. Similar to 5.

7. For $n = 1$, $\llbracket \#k + 2; u_1; @ \rrbracket = \llbracket \#k + 1; @ \rrbracket = \llbracket \#1; @ \rrbracket = \llbracket \#2; u_1; @ \rrbracket$ if $u_1 \neq @$, and otherwise $\llbracket \#k + 2; @; @ \rrbracket = \llbracket @ \rrbracket = \llbracket \#2; @; @ \rrbracket$. For $n > 1$ we apply the induction hypothesis.

8. This follows from the fact that the entry instruction simply behaves as a skip if it does not affect preceding jumps; that is, if the jumps are small enough to be not affected by discontinuation.
9. Let $\mathbf{u} = u_1; \dots; u_n$ and suppose $@; \mathbf{u}; @ =_{\text{bc}} @; \mathbf{u}; \#1$. We shall show by induction on l that

$$@; (\mathbf{u}; @)^l =_{\text{bc}} @; (\mathbf{u}; \#1)^l$$

for all $l > 0$. The base case follows from the assumption. For $l + 2$ we have

$$\begin{aligned} \llbracket (\mathbf{u}; @)^{l+2}; Z \rrbracket &= \llbracket (\mathbf{u}; @)^l; \mathbf{u}; @; \mathbf{u}; @; Z \rrbracket \\ &= \llbracket (\mathbf{u}; @)^l; \mathbf{u}; @; \mathbf{u}; \#1; Z \rrbracket \text{ by the assumption} \\ &= \llbracket (\mathbf{u}; @)^{l+1}; \mathbf{u}; \#1; Z \rrbracket \\ &= \llbracket (\mathbf{u}; \#1)^{l+1}; \mathbf{u}; \#1; Z \rrbracket \text{ by the induction hypothesis} \\ &= \llbracket (\mathbf{u}; \#1)^{l+2}; Z \rrbracket \end{aligned}$$

Thus also $@; (\mathbf{u}; @)^{l+2} =_{\text{bc}} @; (\mathbf{u}; \#1)^{l+2}$ by Corollary 1.2 and hence $\llbracket (\mathbf{u}; @)^l \rrbracket = \llbracket @; (\mathbf{u}; @)^l \rrbracket = \llbracket @; (\mathbf{u}; \#1)^l \rrbracket = \llbracket (\mathbf{u}; \#1)^l \rrbracket$ for all $l > 0$. It follows that $\llbracket (\mathbf{u}; @)^\omega \rrbracket = \llbracket (\mathbf{u}; \#1)^\omega \rrbracket$. Therefore we have $\llbracket (\mathbf{u}; @)^\omega; Z \rrbracket = \llbracket (\mathbf{u}; \#1)^\omega; Z \rrbracket$ for all Z . Thus $@; (\mathbf{u}; @)^\omega =_{\text{bc}} @; (\mathbf{u}; \#1)^\omega$ by Corollary 1.2.

10. Since $\llbracket \#1; @; Z \rrbracket = \llbracket @; Z \rrbracket = \llbracket Z \rrbracket$ for all Z , we can apply Corollary 1.2.
11. By Corollary 1.2 since for all Z , $\llbracket \#n + 2; u; Z \rrbracket = \llbracket \#n + 1; Z \rrbracket$ if $u \neq @$.
12. Again by Corollary 1.2 since for all Z , $\llbracket a; @; Z \rrbracket = a \circ \llbracket Z \rrbracket = \llbracket a; Z \rrbracket$.
13. Similar to (12).
14. Similar to (13).
15. This follows straightforwardly from Corollary 1.2 and the fact that

$$\forall Z \llbracket X; Z \rrbracket = \llbracket Y; Z \rrbracket \ \& \ \llbracket \#2; X; Z \rrbracket = \llbracket \#2; Y; Z \rrbracket$$

iff

$$\forall Z \llbracket X; Z \rrbracket \leq a \geq \llbracket \#2; X; Z \rrbracket = \llbracket Y; Z \rrbracket \leq a \geq \llbracket \#2; Y; Z \rrbracket.$$

16. Apply Theorem 4.1.
17. Since $\llbracket @; ! \rrbracket = S \neq D = \llbracket @; \#j \rrbracket$.
18. Since $\llbracket @; ! \rrbracket = S \neq \llbracket X \rrbracket \leq a \geq \llbracket \#2; X \rrbracket = \llbracket @; +a; X \rrbracket$.
19. Since $\llbracket @; \#0 \rrbracket = D \neq \llbracket X \rrbracket \leq a \geq \llbracket \#2; X \rrbracket = \llbracket @; +a; X \rrbracket$.
20. Since $\llbracket @; +a; X \rrbracket = \llbracket X \rrbracket \leq a \geq \llbracket \#2; X \rrbracket \neq \llbracket Y \rrbracket \leq b \geq \llbracket \#2; Y \rrbracket = \llbracket @; +b; Y \rrbracket$.

The axiom system $\text{PGA1-8} + \text{CEQ}_@$ is obtained by combining the equations for instruction sequence congruence, the axioms for structural equivalence and the axioms of $\text{CEQ}_@$. From the previous proposition it follows that this system is sound, i.e. applying its axioms and the rules of conditional equational logic always yields equations that are valid in $\text{PGA}_{\Sigma, @} / =_{\text{bc}}$. The converse, i.e. provable equality of behavioral congruence, can be shown in the repetition-free case. Completeness for infinite programs remains an open problem.

Theorem 5. *PGA1-8 + CEQ_@ is complete for finite programs, i.e. for repetition-free $X, Y \in \text{PGA}_{\Sigma, @}$,*

$$X =_{\text{bc}} Y \Leftrightarrow \text{PGA1-8} + \text{CEQ}_{@} \vdash X = Y$$

Proof. Right to left follows from the previous proposition. To prove the other direction, first notice that in the absence of entry instructions lengths must be equal, or else a separating context can be easily manufactured. Then, still without @, the fact is demonstrated with induction to program lengths, using (16) as a main tool, in addition to a substantial case distinction.

In the presence of entry instructions, (7) and (8) are used to transform both programs to instruction sequences involving at most a single entry instruction. If only one of the programs contains an entry instruction a separating context is found using a jump that can jump over the program without entry instruction entirely while halting at the other program's entry instruction. At this point it can be assumed that $X = X_1; @; X_2$ and $Y = Y_1; @; Y_2$. Let k be the maximum of the lengths of X_1 and Y_1 , then $\llbracket \#k + 1; X_1; @; X_2 \rrbracket = \llbracket @; X_2 \rrbracket$ and $\llbracket \#k + 1; Y_1; @; Y_2 \rrbracket = \llbracket @; Y_2 \rrbracket$. Now $@; X_2$ and $@; Y_2$ can be proven equal, and this is shown by means of an induction on the sum of the lengths of both. Finally the argument is concluded by an induction on the sum of the lengths of X_1 and Y_1 .

7 A Final Algebra Specification for Behavioral Congruence

In this section we shall show that $\text{PGA1-8} + \text{CEQ}_{@}$ constitutes a final algebra specification of the fully abstract program algebra with entry instruction.

Lemma 2. *Let $X \in \text{PGA}_{\Sigma, @}$. Then*

1. $\llbracket X \rrbracket = S \Rightarrow \text{PGA1-8} + \text{CEQ}_{@} \vdash @; X = @; !$
2. $\llbracket X \rrbracket = D \Rightarrow \text{PGA1-8} + \text{CEQ}_{@} \vdash @; X; \#0^\omega = @; \#0$
3. $\llbracket X \rrbracket = P \trianglelefteq a \triangleright Q \Rightarrow \text{PGA1-8} + \text{CEQ}_{@} \vdash @; X = @; +a; Y$ for some $Y \in \text{PGA}_{\Sigma, @}$

Proof. We shall write $\vdash$ instead of $\text{PGA1-8} + \text{CEQ}_{@} \vdash$ and consider the definition of $|X|$ as a collection of rewrite rules, working modulo instruction sequence equivalence (for which PGA1-4 are complete).

1. The assumption implies that after finitely many rewrites the result S is obtained. We use induction on the length of this rewrite sequence. If one step is needed (the theoretical minimum), there are two cases: $X = !$, or $X = !; Y$ for some Y . The first case is immediate; the second case follows by $\vdash @; X = @; !; Y = !^\omega; Y = !^\omega = @; !$ employing (1). If $k + 1$ steps are needed the last step must be either a rewrite of a jump or the removal of an entry instruction. We only consider the first case. Thus $X = \#n; Y$ for some Y . If $n = 1$ then $|Y| = S$ and hence $\vdash @; Y = @; !$ by the induction hypothesis.

Thus $\vdash @; X = @; \#1; Y = @; Y = @; !$ by (10). If $X = \#n + 2; u; Y$ there are two cases: u is the entry instruction, or not. Assume that it is not. Then $|\#n + 1; Y| = S$. Using the induction hypothesis and (11) it follows that $\vdash @; X = @; \#n + 2; u; Y = @; \#n + 1; Y = @; !$. If u is the entry instruction we have $\vdash @; X = @; \#n + 2; @; Y = @; @; Y = @; Y = @; !$ by (3), (4) and the induction hypothesis.

2. A proof of this fact uses a case distinction: either in finitely many steps the rewriting process of the process extraction leads to $\#0; Z$ for some Z , or an infinite sequence of rewrites results which must be of a cyclic nature. In the first case induction on the number of rewrite steps involved provides the required result without difficulty. The structural congruence equations will not be needed in this case. In the case of an infinite rewrite it follows that the rewriting contains a circularity. By means of the chaining of successive jumps the expression can be rewritten into an expression in which a single jump, contained in the repeating part traverses the whole repeating part and then chains with itself. PGA7 can be used to introduce an instruction $\#0$, thereby reducing the case to the previous one. This is best illustrated by means of an example.

$$\begin{aligned}
& @; \#5; !; \#0; (\#4; +a; \#2; !; \#1)^\omega \\
&= @; \#5; !; \#0; (\#5; +a; \#2; !; \#1)^\omega && \text{PGA6} \\
&= @; \#5; !; \#0; (\#0; +a; \#2; !; \#1)^\omega && \text{PGA7} \\
&= @; \#5; !; \#0; \#0; +a; \#2; !; \#1; (\#0; +a; \#2; !; \#1)^\omega && \text{PGA4} \\
&= @; \#5; !; \#1; \#0; +a; \#2; !; \#1; (\#0; +a; \#2; !; \#1)^\omega && \text{PGA5} \\
&= @; \#2; !; \#1; (\#0; +a; \#2; !; \#1)^\omega && \text{PGA4} \\
&= @; \#1; (\#0; +a; \#2; !; \#1)^\omega && (11) \\
&= @; (\#0; +a; \#2; !; \#1)^\omega && (10) \\
&= @; \#0; +a; \#2; !; \#1; (\#0; +a; \#2; !; \#1)^\omega && \text{PGA4} \\
&= \#0^\omega; +a; \#2; !; \#1; (\#0; +a; \#2; !; \#1)^\omega && (2) \\
&= \#0^\omega && \text{PGA3} \\
&= @; \#0 && (2).
\end{aligned}$$

3. This fact follows by means of an induction on the number of rewrite steps needed for the program extraction operator to arrive at an expression of the form $P \leq a \geq Q$.

The results can be taken together in the following theorem which can be read as follows: ‘PGA₁₋₈+CEQ_@ constitutes a final algebra specification of the fully abstract program algebra with entry instruction’.

Proposition 5.

$$\llbracket X \rrbracket \neq \llbracket Y \rrbracket \Rightarrow \text{PGA}_{1-8} + \text{CEQ}_{@} \vdash @; X \neq @; Y.$$

Proof. With induction on n it will be shown that $\pi_n(\llbracket X \rrbracket) \neq \pi_n(\llbracket Y \rrbracket)$ implies the provability of $@; X \neq @; Y$. The basis is immediate because zeroth projections are D in both cases, and a difference cannot exist. Then suppose that

$\pi_{n+1}(\llbracket X \rrbracket) \neq \pi_{n+1}(\llbracket Y \rrbracket)$ A case distinction has to be analysed. Suppose $\llbracket X \rrbracket = S$ and $\llbracket Y \rrbracket = D$. Then $\text{PGA}_{1-8} + \text{CEQ}_{@} \vdash @; X = @; !$ and $\text{PGA}_{1-8} + \text{CEQ}_{@} \vdash @; X = @; \#0$ by the previous lemma. Thus $\text{PGA}_{1-8} + \text{CEQ}_{@} \vdash @; X \neq @; Y$ using (17). All other cases are similar except one: $\llbracket X \rrbracket = P \trianglelefteq a \triangleright Q$ and $\llbracket Y \rrbracket = P' \trianglelefteq a \triangleright Q'$. Then there must be X' and Y' such that $\text{PGA}_{1-8} + \text{CEQ}_{@} \vdash @; X = @; +a; X'$ and $\text{PGA}_{1-8} + \text{CEQ}_{@} \vdash @; Y = @; +a; Y'$. It then follows that either $\pi_n(\llbracket X' \rrbracket) \neq \pi_n(\llbracket Y' \rrbracket)$ or $\pi_n(\llbracket \#2; X' \rrbracket) \neq \pi_n(\llbracket \#2; Y' \rrbracket)$. In both cases the induction hypothesis can be applied. Finally (15) is applied to obtain the required fact.

Theorem 6.

$$X \neq_{bc} Y \Rightarrow \text{PGA}_{1-8} + \text{CEQ}_{@} \vdash X \neq Y.$$

Proof. If $X \neq_{bc} Y$ then for some P and Q , $\llbracket P; X; Q \rrbracket \neq \llbracket P; Y; Q \rrbracket$. Using the previous proposition $\text{PGA}_{1-8} + \text{CEQ}_{@} \vdash @; P; X; Q \neq @; P; Y; Q$. This implies $\text{PGA}_{1-8} + \text{CEQ}_{@} \vdash X \neq Y$ by the laws of conditional equational logic.

8 Concluding Remarks

Polarized process algebra has been used in order to give a natural semantics for programs.

The question how to give an equational initial algebra specification of the program algebra (with or without entry instruction) modulo behavioral congruence remains open. As stated in [3] behavioral congruence is decidable on PGA expressions. For that reason an infinite equational specification exists. The problem remains to present such a specification either with a finite set of equations or with the help of a few comprehensible axiom schemes. General specification theory (see [4]) states that a finite equational specification can be found which is an orthogonal rewrite system (see [9,5]) at the same time, probably at the cost of some auxiliary functions. Following the proof strategy of [4] an unreadable specification will be obtained, however. The problem remains to obtain a workable specification with these virtues. Thus as it stands both finding an initial algebra specification and finding a ‘better’ final algebra specification (only finitely many equations, no additional object) for program algebra with behavioral congruence are open matters.

Another question left open for further investigation is whether the entry instruction can be naturally combined with the unit instruction operator as studied in [10]. This seems not to be the case. A similar question can be posed regarding the repetition instruction mentioned in [3].

References

1. J.A. Bergstra and J.-W. Klop. Process algebra for synchronous communication. *Information and Control*, 60 (1/3):109–137, 1984.

2. J.A. Bergstra and M.E. Loots. Program algebra for component code. *Formal Aspects of Computing*, 12(1):1–17, 2000.
3. J.A. Bergstra and M.E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002.
4. J.A. Bergstra and J.V. Tucker. Equational specifications, complete rewriting systems and computable and semi-computable algebras. *Journal of the ACM*, 42(6):1194–1230, 1995.
5. I. Bethke. Completion of equational specifications. In Terese, editors, *Term Rewriting Systems*, Cambridge Tracts in Theoretical Computer Science 55, pages 260–300, Cambridge University Press, 2003.
6. S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(8):560–599, 1984.
7. J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54(1/2):70–120, 1982.
8. W.J. Fokkink. Axiomatizations for the perpetual loop in process algebra. In P. Degano, R. Gorrieri, and A. Machetti-Spaccamela, editors, *Proceedings of the 24th ICALP, ICALP'97*, Lecture Notes in Comp. Sci. 1256, pages 571–581. Springer Berlin, 1997.
9. J.-W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science*, volume II, pages 1–116. Oxford University Press, 1992.
10. A. Ponse. Program algebra with unit instruction operators. *Journal of Logic and Algebraic Programming*, 51(2):157–174, 2002.
11. V. Stoltenberg-Hansen, I. Lindström, and E.R. Griffor. *Mathematical Theory of Domains*, Cambridge Tracts in Theoretical Computer Science 22, Cambridge University Press, 1994.

Problems on RNA Secondary Structure Prediction and Design^{*}

Anne Condon

The Department of Computer Science
2366 Main Mall
University of British Columbia
Vancouver, B.C. V6R 2C8
condon@cs.ubc.ca

Abstract. We describe several computational problems on prediction and design of RNA molecules.

1 Introduction

Almost a decade ago, I ventured two blocks from my Computer Sciences department to a very unfamiliar world - the Chemistry Department. This short walk was the start of a rewarding ongoing journey. Along the way, I have made wonderful new friends - both the real sort and the technical sort that like to make their home in the heads of us theoreticians, there to remain indefinitely. In this article, I will describe some of the the latter.

The subjects are nucleic acids: DNA and RNA. From a *biological perspective*, the role of double-helical DNA in storing genetic information is well known. The central dogma of molecular biology posits that in living cells, this genetic information is translated into proteins, which do the real work. The traditional view of RNA is as a helper molecule in the translation process. That view has changed in recent years, with RNA getting star billing in regulation of genes and as a catalyst in many cellular processes [9]. Attention on RNA stems also from the many diseases caused by RNA viruses. Accordingly, significant effort is now expended in understanding the function of RNA molecules. The structure of RNA molecules is key to their function, and so algorithms for prediction of RNA structure are of great value.

While the biological roles of DNA and RNA molecules are clearly of great importance, they are only part of the story. From an *engineering perspective*, DNA and RNA molecules turn out to be quite versatile, capable of functions not seen in nature. These molecules can be synthesized and used as molecular bar-codes in libraries of polymers [24] and as probes on DNA chips for analysis

^{*} This material is based upon work supported by the U.S. National Science Foundation under Grant No. 0130108, by the National Sciences and the Engineering Research Council of Canada, and by the by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-01-2-0555.

of gene expression data. RNA's with new regulatory properties are designed, with hopes of applications in therapeutics [25]. Tiny instances of combinatorial problems have been solved in a wet-lab, using DNA or RNA to represent a pool of solutions to a problem instance [4]. Novel topological and rigid three-dimensional structures have been built from DNA [22,30], and a theory of programmable self-assembly of such structures is emerging [20]. Scientists are working to create catalytic RNA molecules that support the so-called “RNA world hypothesis”: prior to our protein-dominated world, RNA molecules functioned as a complete biological system capable of the basic processes of life [26]. Naturally, advances in these areas also rely greatly on good understanding of function, and hence structure, of RNA and DNA molecules.

The problems described in this article are motivated more by the engineering, rather than the biological perspective of the potential roles of DNA and RNA. Even for the problem of predicting RNA structure, the two different perspectives suggest somewhat different approaches. In the biological setting, it is often possible to get sequences of homologous (i.e. evolutionarily and functionally related) molecules from several organisms. In this case, a comparative approach that uses clues about common structure from all molecules in the set are the most successful in structure prediction. However, in the engineering setting, this approach is typically not applicable. Moreover, the inverse to the prediction problem, namely design of a DNA or RNA molecule that has a particular structure, is of central importance when engineering novel molecules.

We focus on problems relating to RNA and DNA *secondary structure*, which we describe in Section 2. In Section 3, we describe problems on predicting the secondary structure of a given DNA or RNA molecule. Section 4 considers more general problems when the input is a set of molecules. Finally, in Section 5, we describe problems on the design of DNA and RNA molecules that fold to a given input secondary structure.

2 Basics on RNA Secondary Structure

To keep things simple, consider an RNA molecule to be a strand of four types of bases, with two chemically distinct ends, known as the 5' and 3' ends. In RNA the base types are Adenine (A), Cytosine (C), Guanine (G), and Uracil (U). DNA also has four types of bases, including A, C, G and replacing Uracil (U) with Thymine (T). We represent an RNA (DNA) molecule as a string over $\{A, C, G, U\}$ ($\{A, C, G, T\}$), with the left end corresponding to the 5' end of the molecule.

In a process called hybridization, pairs of bases in RNA and DNA form hydrogen bonds, with the complementary pairs C-G and A-U (or A-T in the case of DNA) being the strongest and others, particularly the “wobble” pair G-U also playing a role [29]. A folded molecule is largely held together by the resulting set of bonds. called its *secondary structure*. Knowledge of the secondary structure of a folded RNA molecule sheds valuable insight on its function [27]. We note that while the DNA that stores genetic information in living organisms

is formed from two complementary strands, single-stranded DNA folds and forms structures according to the same basic principles as does a single stand of RNA.

Figure 1 depicts the secondary structure of two DNA molecules. In the graphical depictions (top), dots indicate base pairs, and “stems” of paired bases and “loops” of unpaired bases can be identified. The graphical depictions do not convey the three-dimensional structure of the molecules. For example, stems twist to form double helices familiar in illustrations of DNA, and angles at which stems emanate from loops cannot be inferred from the diagrams. In the arc depiction (bottom), arcs connect paired bases. In the left structure, arcs are hierarchically nested, indicating that this is a *pseudoknot free* structure. In contrast, arcs cross in the arc depiction of the structure on the right, indicating that it is *pseudoknotted*.

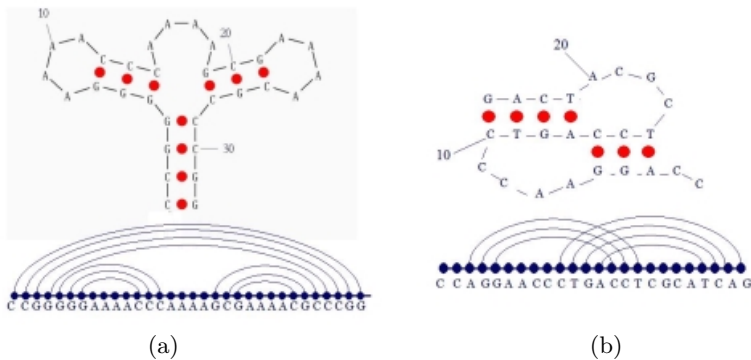


Fig. 1. (a) Pseudoknot free secondary structure. This structure contains 10 base pairs and three loops, two of which are hairpin loops (having one emanating stem) and one of which is a multi-loop (having three emanating stems). The numbers refer to base indices, in multiples of 10, starting at the 5′ end (leftmost base in arc depiction). The substructure from index 19 to index 28 contains a stem with two stacked pairs, namely (G-C,C-G) and (C-G,G-C), and a hairpin loop with four unpaired bases (all A’s) and closing base pair G-C. In set notation, this substructure is $\{(19, 28), (20, 27), (21, 26)\}$. The free energy contributions of the two stacked pairs and hairpin loop are -3.4 kcal/mol, -2.4 kcal/mol, and 4.5 kcal/mol, respectively, so the total free energy of the substructure from index 19 to 28 is -1.3 kcal/mol. (b) Pseudoknotted secondary structure.

Abstractly, we represent the secondary structure of a DNA or RNA molecule of length (i.e. number of bases) n as a set S of integer pairs $\{(i, j) \mid 1 \leq i < j \leq n\}$, where each i is contained in at most one pair of S . The pair (i, j) indicates a bond between the bases at positions i and j of the corresponding strand. The secondary structure is pseudoknot free if and only if for all pairs (i, j) and (i', j') , it is not the case that $i' < i < j' < j$.

The thermodynamic model for RNA structure formation posits that, out of the exponentially many possibilities, an RNA molecule folds into that structure with the *minimum free energy* (mfe). Free energy models typically assume that the total free energy of a given secondary structure for a molecule is the sum of independent contributions of adjacent, or stacked, base pairs in stems (which tend to stabilize the structure) and of loops (which tend to destabilize the structure). These contributions depend on temperature, the concentration of the molecule in solution, and the ionic concentration of the solution. *Standard models* additionally assume that the free energy contribution of a loop depends only on (i) the bases closing the stem and those unpaired bases in the loop adjacent to the stem, for each stem, (ii) the number of stems emanating from the loop, and (iii) the number of unpaired bases between consecutive stems. For loops with more than two stems, (ii) and (iii) are further simplified to be of the form $a + bs + cu$, where b, c are constants, s is the number of stems emanating from the loop, and u is the total number of unpaired bases in the loop.

Significant effort has been expended to determine many of these energy contributions experimentally [21,23]. Other contributions are estimated based on extrapolations from known data or existing databases of naturally occurring structures [17]. More sophisticated models also associate energy contributions with coaxially stacked pairs and other structural features, but we will ignore these here for the sake of simplicity.

3 RNA Secondary Structure Prediction

“If 10% of protein fold researchers switched to RNA, the problem could be solved in one or two years.” - I. Tinoco Jr. and C. Bustamente

The best known algorithms for predicting the secondary structure of a single input RNA or DNA molecule work by finding the minimum free energy (mfe) secondary structure of the given input RNA molecule, with respect to a given standard thermodynamic model. Lyngsø and Pedersen [15] have shown that the task is NP-hard. However, the problem is not as intractable as this might suggest, because in practice the range of structures into which a molecule will actually fold is somewhat limited.

Zuker and Steigler [32] describe a dynamic programming algorithm for finding the mfe pseudoknot free secondary structure of a given molecule. (In practice, the algorithm can be used to gain insight on secondary structure even for molecules with pseudoknotted structures, because there is some evidence that molecules fold to form a pseudoknot free secondary structure first, and pseudoknotted features are added only at the end of the folding process.) Conceptually the algorithm is quite simple, exploiting the following fact. Let the input strand be $b_1b_2 \dots b_n$. Suppose that $W(i, j)$ is the energy of the mfe pseudoknot free secondary structure for strand $b_i \dots b_j$, and $V(i, j)$ be the energy of the mfe pseudoknot free secondary structure for strand $b_i \dots b_j$, among those structures containing base pair (i, j) . Then, W satisfies the following recurrence (base cases excluded):

$$W(i, j) = \min[V(i, j), \min_{k:i \leq k \leq j} \{W(i, k) + W(k + 1, j)\}].$$

$V(i, j)$ also satisfies a recurrence that is expressed in terms of the different types of loops (omitted here). A refinement of the original Zuker-Steigler algorithm, due to Lyngsø et al. [16], has running time $O(n^3)$. We note that the algorithm exploits the simplified loop energy contributions of the standard thermodynamic model mentioned earlier. Implementations of this algorithm are available on the world wide web as part of the mfold [17] and the Vienna [13] packages.

Mathews et al. [17] report that on a large data set of RNA molecules of length up to 700, the algorithm reports 73% of known base pairs. On longer molecules, the prediction accuracy is poorer. Thus, there is certainly room for improvement in the current mfe approach to secondary structure prediction. Perhaps the most important problem listed in this article is to find algorithms for pseudoknot free secondary structure prediction that have improved accuracy. We expect that significant progress will only come through a greater understanding of the underlying biological forces that determine folding, perhaps by refining the currently used thermodynamic model or by considering the folding pathway of molecules. In light of this and the subtle interplays between algorithmic and modeling considerations, we believe that the best progress can be made only through productive collaborations between algorithm designers and experts on nucleic acids.

So far, we have focused on the problem of finding the mfe secondary structure (with respect to some thermodynamic model) of a DNA or RNA molecule. Other information on the stability of the molecule's structure can also be very useful. A better view is that each possible secondary structure S for molecule M occurs with a probability that is proportional to $e^{-\Delta G(S)/RT}$ where $\Delta G(S)$ is the free energy associated with structure S , R is the Boltzmann constant, and T is temperature. Associated with each possible base pair of the molecule is a weight, defined to be the sum of the probabilities of the structures in which it occurs. McCaskill [18] gave an $O(n^3)$ dynamic for calculating the set of base pair weights of a molecule. This algorithm is incorporated into standard folding packages [17,13], significantly enhancing their utility. Another useful enhancement to the Zuker-Steigler algorithm outputs not just the mfe structure, but all structures with energy below a user-supplied threshold [31,33].

From a purely algorithmic standpoint, the problem of predicting RNA and DNA secondary structure becomes more interesting when one considers pseudoknotted structures. The thermodynamic model for pseudoknot free secondary structures has been extended to include contributions of pseudoknotted stems and loops. Several algorithms have been proposed for predicting the mfe secondary structure from a class of secondary structures that allows limited types of pseudoknots [1,15,19,28]. Other algorithms are heuristic in nature, such as the genetic algorithm of Gulyaev et al. [12]. The dynamic programming algorithm of Rivas and Eddy [19] is the most general in terms of the class of structures handled. The authors claim that all known natural structures can be handled by the algorithm, although they do not provide evidence for this claim. However, the authors state that "we lack a systematic *a priori* characterization of the

class of configurations that this algorithm can solve”. Another limitation of the algorithm is its high running time of $\Theta(n^6)$. An algorithm of Akutsu [1] runs in $O(n^4)$ time and $O(n^2)$ space, but there are natural pseudoknotted structures that cannot be handled by this algorithm.

An interesting goal for further research is to precisely classify pseudoknotted structures, refining the current partition into pseudoknot free and pseudoknotted structures. As a first step in this direction, we have developed a characterization of the class of secondary structures that can be handled by the Rivas and Eddy algorithm. Roughly, a secondary structure can be handled by that algorithm if and only if in the arc depiction of that structure (see Figure 1), all arcs can be reduced to one arc by repeatedly applying a *collapse* operation. In a collapse operation, two arcs can be replaced by one arc if one can colour at most two line segments along the baseline of the depiction, and touch all four end points of the two arcs but no other arc. (We note that a natural approach to classification of secondary structures, which does not seem to be particularly fruitful, is to consider the crossing number of the arc depiction of the secondary structure.)

With a good classification of secondary structures in hand, one can then hope to clarify the trade-offs between the class of structures that can be handled, and the time or space requirements of algorithms for predicting mfe pseudoknotted structures. Perhaps the classification would provide a hierarchy of structure classes, parameterized by some measure k , and a fixed-parameter tractability result for this classification is possible, as in the work of Downey et al. [10].

It would be very useful to calculate the partition function for pseudoknotted structures. An extension of the Rivas and Eddy algorithm along the lines of McCaskill [18] should be possible, but would be computationally expensive and limited by the range of structures handled by the Rivas and Eddy algorithm. It may be possible to approximate the partition function via the Markov chain monte carlo method of Jerrum and Sinclair [14].

Finally, we note that secondary structures can also form between two or more RNA or DNA molecules in solution, so a natural generalization of the problem discussed so far is to predict the mfe secondary structure formed by two or more input molecules. Conceptually, the thermodynamic model for a secondary structure formed from multiple strands is very similar to that for a single strand, but an initiation penalty is added to the total free energy. An algorithm for predicting the secondary structure of a pair of molecules is publically available [2]. Some interesting algorithmic questions arise in design of algorithms for handling multiple strands. For example, what does it mean for a structure with multiple strands to be pseudoknot free?

4 Prediction for Combinatorial Sets of Strands

The problems in this section are motivated by the use of *combinatorial sets* of strands in various contexts. In the first context, described by Brenner et al. [7], the goal is to sequence millions of short DNA fragments (these fragments could be in a gene expression sample). DNA sequencing machines handle one sequence

at a time, and it would be infeasible to separate out the millions of short fragments and sequence each separately. Instead, Brenner described an ingenious “biomolecular algorithm” to sequence the molecules in a massively parallel fashion. One step of this algorithm attaches a unique DNA “tag” molecule to each of the DNA fragments. The tags are used to help to organize the DNA fragments in further steps of the algorithm. Let

$$S = \{\text{TTAC}, \text{AATC}, \text{TACT}, \text{ATCA}, \text{ACAT}, \text{TCTA}, \text{CTTT}, \text{CAAA}\}. \quad (1)$$

The tags constructed by Brenner et al. [8] are all of the 8^8 strands in the combinatorial set S^8 . The strands in S were carefully designed so that each contains no G’s, exactly one C, and differs from the other strands of S in three of the four bases. The reason for this design is to ensure that the tags do not fold on themselves (that is, have no secondary structure), in which case they would not be useful as tag molecules in the sequencing scheme.

The set S of tags given in (1) above is an example of a *complete combinatorial set*, defined as a set of strings (strands) in $S(1) \times S(2) \dots \times S(t)$, where for each $i, 1 \leq i \leq t$, $S(i)$ is a set of strings, all having the same length l_i . The l_i are not required to be equal. Complete combinatorial sets are also used to represent solution spaces in biocomputation that find a satisfying assignment to an instance of the Satisfiability problem [6,11]. Again, for this use, all strands in the complete combinatorial sets should form no secondary structure.

These applications motivate the the **structure freeness problem for combinatorial sets**: given the description of a complete combinatorial set S , determine whether all of the 2^t strands in S are structure free. Here, we consider a strand to be structure free if its mfe pseudoknot free secondary structure is the empty set. We limit our definition to pseudoknot free secondary structures here because in the case of predicting the mfe secondary structure of a single molecule, the pseudoknot free case is already well understood, as discussed in the last section of this article.

Given sets of strings $S(1), S(2), \dots, S(t)$, one can test that all strands in $S = S(1) \times S(2) \dots \times S(t)$ are structure free by running the Zuker-Steigler algorithm on each strand of S . This would take time proportional to $|S|n^3$, where $n = l_1 + l_2 + \dots + l_t$ is the total length of strands in S . In general, this running time is exponential in the input size. Andronescu et al. [3] describe a simple generalization of the Zuker-Steigler algorithm, which has running time $O(\max_i |S(i)|^2 n^3)$.

The algorithm of Andronescu et al. handles only complete combinatorial sets. More general combinatorial sets can be defined via an acyclic graph G with a special start node and end node. Suppose that each node i in the graph is labeled with a set of strands S_i . Then, each path $n_1, n_2, \dots, n_t$ in the graph from the start node to the end node corresponds to the set of strands $S(n_1) \times S(n_2) \dots \times S(n_t)$. The combinatorial set of strands $S(G)$ associated with the graph is the union of the set of strands for each path of G from the start node to the end node. (Since G is acyclic, there are a finite number of such paths). Such a combinatorial set of strands was used by Adleman [4] in his biomolecular computation for a

small instance of the Hamiltonian Path problem. It is open whether there is an efficient algorithm to test if all strands $S(G)$ are structure free, where the input is the graph G and the set $S(i)$ of strands for each node i of G . The case where all strands in $S(i)$ have the same length, for any node i of G , is also open. By adding cycles to G , the problem becomes even more general, and its complexity remains open even for the simplest case that the nodes and edges of G form a simple cycle.

5 Secondary Structure Design

“... rather than examining in detail what occurs in nature (biological organisms), we take the engineering approach of asking, what can we build?” - Erik Winfree.

The simplest version of the RNA design problem is as follows: given a secondary structure S (that is, set of desired base pairings), design a strand whose mfe secondary structure is S , according to the standard thermodynamic model. There has been relatively little previous theoretical work on algorithms for design of DNA or RNA molecules that have certain structural properties. Indeed, it is open whether the problem is NP-hard, although we conjecture that this is the case. Even if the range of secondary structures is restricted to be the pseudoknot free secondary structures, the complexity of the problem is open.

However, as with RNA secondary structure prediction, we expect that the range of structures one may wish to design in practice will be somewhat limited. Thus, it would certainly be useful to provide characterizations of secondary structure classes for which the design problem is efficiently solvable. More useful versions of the RNA design problem may pose additional requirements, perhaps on the stability of the mfe structure or on the base composition of the RNA molecule.

A generalization of the RNA secondary structure design problem above arises when the desired structure is composed of more than one strand. Many of the applications of RNA secondary structure design that we are familiar with involve multiple strands. For example, Seeman has designed several multi-strand structural motifs, and has developed an interactive software tool to help design the component strands [22]. Winfree et al. [30] proposed a method for self-assembly of DNA “tile” molecules in a programmable fashion, and has shown that programmable self-assembly is in principle capable of universal computation. The component tile molecules used in these self-assembly processes involve four component strands, which form a rigid two-dimensional structure with protruding short single strands, called sticky ends, that are available for hybridization with the sticky ends of other tile molecules. RNA molecules are designed as molecular switches, biosensors, and even for therapeutic uses. For example, it is possible to inhibit the action of certain pathogenic RNA molecules (such as viruses) using carefully-designed short RNA molecules, called *trans*-cleaving ribozymes, that can bind to the pathogenic RNA and cleave it [25]. The *trans*-cleaving ribozymes

are currently developed via *in-vitro* evolution, in which a large library of RNA molecules is screened to select for those that exhibit some tendency towards the desired function and the screened molecules are then randomly mutated, in order to diversify the pool. The screening and diversification steps are repeated until a molecule with the desired function is obtained. Computational methods for design of RNA molecules could help provide good starting points for *in-vitro* evolution processes. As with the RNA secondary structure design problem for a single strand, while ad-hoc techniques are in use by researchers in Chemistry, there is little theoretical knowledge of good algorithmic design principles.

Finally, a design problem that has received significant attention is that of designing combinatorial sets of molecules that have no secondary structure. This is the inverse of the prediction problem mentioned in Section 4. BenDor et al. [5] describe a combinatorial design scheme with provably good properties that address one version of this problem. Other approaches, such as the simple design of Brenner described in Section 4, construct strands in the component sets ($S(i)$) of the combinatorial sets to be over a three-letter alphabet and have certain coding-theoretic properties. In light of the wide uses of these designs, further insights as to good design strategies would be useful.

6 Conclusions

This article has described several problems of a combinatorial flavour relating to RNA secondary structure prediction and design. These problems are motivated by work in design of RNA and DNA strands for diverse applications with both biological and computational motivations. The prediction and design problems are inter-related, with good algorithms for prediction being a prerequisite to tackling the secondary structure design problems. In light of the importance of these problems in both the biological and engineering settings, and the relatively little attention they have received to date from the computer science community, they represent a fruitful direction for algorithms research.

Inevitably, the problems reflect my own interests and biases. Many other theoretically interesting problems, motivated by three-dimensional RNA structure prediction, visualization of secondary structures, and more are not covered here, but raise interesting questions in computational geometry and graph drawing.

Acknowledgements. I wish to express my great appreciation to the many friends that I have made on this interdisciplinary journey, who have shared their experience, wisdom, and enthusiasm with me. A special thank you to my collaborators Mirela Andronescu, Rob Corn, Holger Hoos, and Lloyd Smith, and Dan Tulpan, who have made this journey so rewarding.

References

1. T. Akutsu, “Dynamic programming algorithms for RNA secondary prediction with pseudoknots”, *Discrete Applied Mathematics*, 104, 2000, 45–62.

2. M. Andronescu, R. Aquirrez-Hernandez, H. Hoos, and A. Condon, "RNAssoft: a suite of RNA secondary structure prediction and design software tools", *Nucleic Acids Research*, In press.
3. M. Andronescu, D. Dees, L. Slaybaugh, Y. Zhao, A. Condon, B. Cohen, and S. Skiena, "Algorithms for testing that sets of DNA words concatenate without secondary structure", *Proc. Eighth International Workshop on DNA Based Computers*, Hokkaido, Japan, June 2002. To appear in LNCS.
4. L.M. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, Vol 266, 1994, 1021–1024.
5. A. Ben-Dor, R. Karp, B. Schwikowski, and Z. Yakhini, "Universal DNA tag systems: a combinatorial design scheme," *Proc. Fourth Annual International Conference on Computational Molecular Biology (RECOMB) 2000*, ACM, 65–75.
6. Braich, R. S., Chelyapov, N., Johnson, C., Rothmund, P. W.K. and Adleman, L. "Solution of a 20-variable 3-SAT Problem on a DNA computer", *Science* 296, 2002, 499–502.
7. S. Brenner, M. Johnson, J. Bridgham, G. Golda, D.H. Lloyd, D. Johnson, S. Luo, S. McCurdy, M. Foy, M. Ewan, R. Roth, D. George, S. Eletr, G. Albrecht, E. Vermaas, S.R. Williams, K. Moon, T. Burcham, M. Pallas, R.B. DuBridge, J. Kirchner, K. Fearon, J. Mao, and K. Corcoran, "Gene expression analysis by massively parallel signature sequencing (MPSS) on microbead arrays," *Nature Biotechnology*, 18, 2000, 630–634.
8. S. Brenner, "Methods for sorting polynucleotides using oligonucleotide tags," U.S. Patent Number 5,604,097, 1997.
9. C. Dennis, "The brave new world of RNA", *Nature*, 418, 11 2002, 122–124.
10. R. G. Downey and M. R. Fellows, "Fixed-Parameter Tractability and Completeness I: Basic Results", *SIAM J. Comput.* 24(4), 1995, 873–921.
11. D. Faulhammer, A.R. Cukras, R.J. Lipton, and L. F. Landweber, "Molecular computation: RNA solutions to chess problems," *Proc. Natl. Acad. Sci. USA*, 97, 2000, 1385–1389.
12. .P.Gulyaev, F.H.D.van Batenburg, and C.W.A.Pleij, "The computer simulation of RNA folding pathways using a genetic algorithm", *J. Mol. Biol.*, 250, 1995, 37–51.
13. I. L. Hofacker, W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker, and P. Schuster, "Fast Folding and Comparison of RNA Secondary Structures", *Monatsh.Chem.* 125, 1994, 167–188.
14. M. Jerrum and A. Sinclair, "Approximating the permanent", *SIAM Journal on Computing* 18, 1989, 1149–1178.
15. R.B. Lyngsø and C.N.S. Pedersen, "Pseudoknot prediction in energy based models", *Journal of Computational Biology* 7(3), 2000, 409–427.
16. R. B. Lyngsø, M. Zuker, and C. N. S. Pedersen, "Internal Loops in RNA Secondary Structure Prediction", *Proc. Third International Conference in Computational Molecular Biology (RECOMB)*, April 1999, 260–267.
17. D.H. Mathews, J. Sabina, M. Zuker, and D.H. Turner, "Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure", *J. Molecular Biology*, 288, 1999, 911–940.
18. J.S. McCaskill, "The equilibrium partition function and base pair binding probabilities for RNA secondary structure," *Biopolymers*, 29, 1990, 1105–1119.
19. E. Rivas and S. Eddy, "A dynamic programming algorithm for RNA structure prediction including pseudoknots," *Journal of Molecular Biology*, 285, 1999, 2053–2068.
20. P.W.K. Rothmund and E. Winfree, "The program-size complexity of self-assembled squares", *Symposium on Theory of Computing*, 2000.

21. J. SantaLucia, "A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics", *Proc. Natl Acad Sci USA* 95:4, 1998, 1460–1465.
22. N.C. Seeman, "De novo design of sequences for nucleic acid structural engineering," *Journal of Biomolecular Structure and Dynamics*, 8:3, 1990, 573–581.
23. M.J. Serra, D.H. Turner, and S.M. Freier, "Predicting thermodynamic properties of RNA", *Meth. Enzymol.*, 259, 1995, 243–261.
24. D. D. Shoemaker, D. A. Lashkari, D. Morris, M. Mittman, and R. W. Davis, "Quantitative phenotypic analysis of yeast deletion mutants using a highly parallel molecular bar-coding strategy," *Nature Genetics*, 16, 1996, 450–456.
25. B.A. Sullenger and E. Gilboa. "Emerging clinical applications of RNA", *Nature*, 418, 2002, 252–258.
26. J.W. Szostak, D.P. Bartel, and L. Luisi. "Synthesizing life", *Nature* 409, 2001, 387–389.
27. I. Tinoco Jr. and C. Bustamante, "How RNA folds", *J. Mol. Biol.* 293, 1999, 271–281.
28. Y. Uemura, A. Hasegawa, Y. Kobayashi, and T. Yokomori, "Tree adjoining grammars for RNA structure prediction", *Theoretical Computer Science*, 210, 1999, 277–303.
29. E. Westhof and V. Fritsch, "RNA folding: beyond Watson-Crick pairs", *Structure* 2000, 8:R55–R65, 2000.
30. E. Winfree, F. Liu, L. Wenzler, and N. Seeman, "Design and self-assembly of 2D DNA crystals," *Nature*, 394, 1998, 539–544.
31. S. Wuchty, W. Fontana, I. L. Hofacker, and P. Schuster, "Complete Suboptimal Folding of RNA and the Stability of Secondary Structures", *Biopolymers*, 1998, Vol. 49, 145–165.
32. M. Zuker and P. Steigler, "Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information," *Nucleic Acids Res* 9, 1981, 133–148.
33. M. Zuker, "On Finding All Suboptimal Foldings of an RNA Molecule", *Science*, 244, 1989, 48–52.

Some Issues Regarding Search, Censorship, and Anonymity in Peer to Peer Networks

Amos Fiat

School of Computer Science, Tel-Aviv University
`fiat@tau.ac.il`

Abstract. In this survey talk we discuss several problems related to peer to peer networks. A host of issues arises in the context of peer to peer networks, including efficiency issues, censorship issues, anonymity issues, etc. While many of these problems have been studied in the past, the file swapping application has taken over the Internet, given these problems renewed impetus. I will discuss papers co-authored with J. Saia, E. Cohen, H. Kaplan, R. Berman, A. Ta-Sham, and others.

The SPQR-Tree Data Structure in Graph Drawing

Petra Mutzel

Vienna University of Technology,
Karlsplatz 13 E186, A-1040 Vienna, Austria
mutzel@ads.tuwien.ac.at
<http://www.ads.tuwien.ac.at>

Abstract. The data structure SPQR-tree represents the decomposition of a biconnected graph with respect to its triconnected components. SPQR-trees have been introduced by Di Battista and Tamassia [13] based on ideas by Bienstock and Monma [9,10]. For planar graphs, SPQR-trees have the nice property to represent the set of all its combinatorial embeddings. Therefore, the data structure has mainly (but not only) been used in the area of planar graph algorithms and graph layout. The techniques are quite manifold, reaching from special purpose algorithms that merge the solutions of the triconnected components in a clever way to a solution of the original graph, to general branch-and-bound techniques and integer linear programming techniques. Applications reach from Steiner tree problems, to on-line problems in a dynamic setting as well as problems concerned with planarity and graph drawing. This paper gives a survey on the use of SPQR-trees in graph algorithms, with a focus on graph drawing.

1 Introduction

The data structure SPQR-tree represents the decomposition of a biconnected graph with respect to its triconnected components. SPQR-trees have been introduced by Di Battista and Tamassia [13] based on ideas used by Bienstock and Monma in [9,10], who studied the problem of identifying a polynomial solvable special case of the Steiner tree problem in graphs [9]. For this, they needed to compute a minimum-weight circuit in a planar graph $G = (V, E)$ separating a given vertex sub-set $F \subseteq V$ from the outer face in a plane drawing. Bienstock and Monma considered two cases: one in which a combinatorial embedding of G is specified, and the other in which the best possible combinatorial embedding is found. A (*combinatorial*) *embedding* essentially fixes the *faces* (*regions*) of a planar drawing (for a formal definition, see Section 2). While the problem for the specified embedding was relatively easy to solve, the *best embedding* problem needed a decomposition approach. Bienstock and Monma solved this problem using a decomposition of G into its seriell, parallel, and “general” (the remaining) components.

In [10], Bienstock and Monma used a very similar approach for computing an embedding of a planar graph $G = (V, E)$ that minimizes various distance

measures of G to the outer face (e.g., the radius, the width, the outerplanarity, and the depth). Observe, that a planar graph can have, in general, an exponential number of embeddings. Hence, it is not possible to simply enumerate over the set of all embeddings. Indeed, many optimization problems over the set of all possible embeddings of a planar graph are NP-hard.

In [13,4,15,14], the authors have suggested the SPQR-tree data structure in order to solve problems in a dynamic setting. In [13,15], Di Battista and Tamassia introduced the SPQR-tree data structure for planar graphs in order to attack the on-line planarity testing problem, while in [14], the data structure has been introduced for non-planar graphs for maintaining the triconnected components of a graph under the operations of vertex and edge insertions. In [4], Di Battista and Tamassia consider planar graphs under dynamic setting. E.g., they show how to maintain a minimum spanning tree under edge weight changes. The considered problems can be solved easier, if the graphs are already embedded in the plane and the edge insertion operation respects the embedding (i.e., it does not introducing crossings). The authors show that the fixed-embedding restriction can be removed by using the SPQR-tree data structure. They obtain a $O(\log n)$ time bound for the dynamic minimum spanning tree problem (amortized only for the edge insertion operation, worst-case for the other operations). For this, the authors use the property of SPQR-trees of representing the set of all embeddings in linear time and space. The SPQR-tree data structure can be computed in linear time [15,25,21] (see also Section 3).

Since then, SPQR-trees evolved to an important data structure in the field of graph algorithms, in particularly, in graph drawing. Many linear time algorithms that work for triconnected graphs only can be extended to work for biconnected graphs using SPQR-trees (e.g., [7,23,22]). Often it is essential to represent the set of all combinatorial embeddings of a planar graph, e.g. [29,6,10,15]. In a dynamic environment, SPQR-trees are useful for a variety of on-line graph algorithms dealing with triconnectivity, transitive closure, minimum spanning tree, and planarity testing [4,15,14].

The techniques are quite manifold, reaching from special purpose algorithms merging the solutions for the components in a clever way to general branch-and-bound techniques and integer linear programming techniques. Applications reach from Steiner tree problems [9], to on-line problems in a dynamic setting [4,15,14] as well as triangulation problems [8], planarity related problems [7,12,19,5] and graph drawing problems [6,29,30,23,24,17,22]. However, only few applications that are of interest outside the graph drawing community are reported. The Steiner tree application [9] has already be mentioned above.

Chen, He, and Huang [11] use SPQR-trees for the design of complimentary metal-oxide semi-conduct (CMOS) VLSI circuits. Their linear time algorithm is able to decide if a given planar graph has a plane embedding π such that π has an Euler trail $P = e_1, e_2, \dots, e_m$ and its dual graph has an Euler trail $P^* = e_1^*, e_2^*, \dots, e_m^*$, where e_i^* is the dual edge of e_i .

Biedl et al. [8] consider triangulation problems under constraints with applications to mesh generation in computational geometry, graph augmentation,

and planar network design. They suggest a linear time algorithm for the problem of deciding if a given planar graph has a plane embedding π with at most twice the optimal number of separating triangles (i.e., triangles which are not a face in the embedding). This directly gives an algorithm for deciding if a biconnected planar graph can be made 4-connected while maintaining planarity.

This talk gives a survey on the use of SPQR-trees in graph algorithms, with a focus on graph drawing. The first part gives an introduction in automatic graph drawing. We will discuss topics like planarity, upward planarity, cluster planarity, crossing minimization, and bend minimization (see Section 2) for which the SPQR-tree data structure has been used successfully. The second part introduces the SPQR-tree data structure in a formal way (see Section 3). The third part of my talk gives an overview of the various techniques used when dealing with the SPQR-tree data structure. In the last part of my talk, we will discuss some of the algorithms for solving specific problems. For this part, see, e.g. [23,30,6,15,22].

2 Automatic Graph Drawing

In graph drawing, the aim is to find a drawing of a given graph in the plane (or in three dimensions) which is easy to read and understand. Aesthetic criteria for *good* drawings are a small number of crossings, a small number of bends, a small resolution (with respect to the area of the drawing and the angles of the edges), and short edges. These aesthetics are taken into account in the so-called *topology-shape-metrics method*. Here, in the first step, the topology of the drawing is determined in order to get a small number of crossings. From then on, the *topology* is taken as fixed. This is achieved by introducing virtual vertices on the crossing points in order to get a so-called *planarized* graph. In the second step, the number of bends is computed; this is done usually using an approach based on network flow. This fixes the *shape* of the drawing. In the third step, everything but the *metrics* is already fixed. The task now is to compute the lengths of the edges; this determines the area of the final drawing. The topology-shape-metrics method often leads to drawings with a small number of crossings (much smaller than alternative drawing methods). Figure 1 displays a drawing, which has been computed with the topology-shape-metrics method¹.

If the first step of the topology-shape-metrics method is computed based on planarity testing, then this method guarantees that any planar graph will indeed be drawn without any edge crossings. Graphs that can be drawn without edge crossings are called *planar graphs*. (*Combinatorial*) *embeddings* are equivalence classes of planar drawings which can be defined by the sequence of the incident edges around each vertex in a drawing. We consider two drawings of the same graph equivalent, if the circular sequences of the incident edges around each vertex in clockwise order is the same. We say that they realize the same *combinatorial embedding*.

¹ The drawing has been generated with AGD [1]

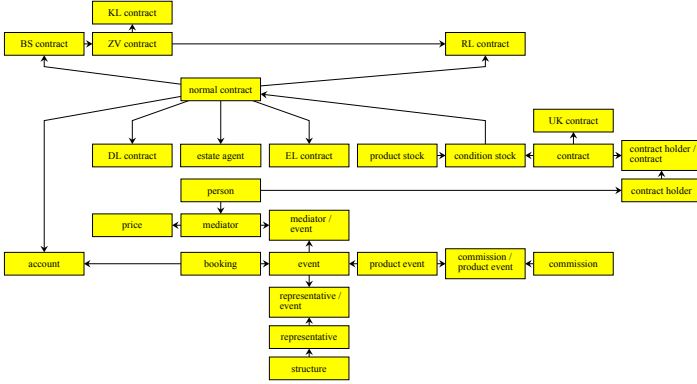


Fig. 1. A drawing of a graph using a topology-shape-metrics method

The first step of the planarization method is usually computed via a planar subgraph. Here, a small number of edges F is deleted from the graph G such that the resulting graph P gets planar. Then, the deleted edges are re-inserted into the planar subgraph in a second step. This re-insertion is done in an iterative way. If the embedding of the planar graph P is fixed, then re-insertion of one edge can be done with the minimum number of crossings via searching a shortest path in the extended geometric dual graph. Gutwenger et al. [23] have shown that SPQR-trees can be used in order to guarantee the minimum number of crossings over the set of all embeddings of the planar graph P . This algorithm runs in linear time. This is an example for which the linear time algorithm for triconnected graphs can be extended to work for biconnected graphs using the SPQR-tree data structure.

The second step is based on an idea by Tamassia [34] who suggested a polynomial time algorithm for computing a bend minimum drawing of a given graph with fixed embedding and maximal vertex degree four by transforming it to a network flow problem. Figure 2(a) shows a bend minimum drawing for the given embedding, while Figure 2(b) shows a bend minimum drawing over the set of all planar embeddings. Unfortunately, the bend minimization problem is NP-hard in the case that the embedding is not part of the input. Bertolazzi et al. [6] suggest a branch-and-bound algorithm based on the SPQR-tree data structure that essentially enumerates over the set of all planar embeddings and solves the corresponding network-flow problem. Moreover, it contains new methods for computing lower bounds by considering partial embeddings of the given graph. An alternative approach for the problem has been suggested by Mutzel and Weiskircher [30]. They have suggested a branch-and-cut algorithm based on an integer linear programming formulation for optimization over the set of

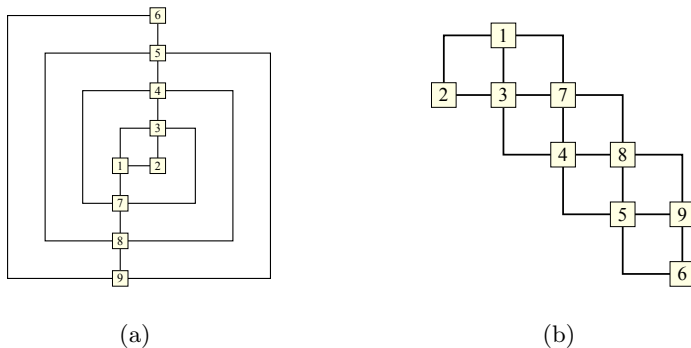


Fig. 2. Bend minimum drawings (a) for a given fixed embedding, and (b) over the set of all embeddings.

all planar embeddings as suggested in [29]. Both approaches are based on the SPQR-tree data structure and are not restricted to maximal vertex degree four.

Since bend minimization is NP-hard, but the choice of a *good* embedding is essential, Pizzonia and Tamassia [31] suggest alternative criteria. They argue that planar embeddings with *minimum depth* in the sense of topological nesting (other than the depth considered in [10]) will lead to good drawings in practice. However, their algorithm is only able to compute embeddings with minimum depth if the embeddings of the biconnected components are fixed. Recently, Gutwenger and Mutzel [22] came up with a linear time algorithm which is able to compute an embedding with minimum depth over the set of all possible embeddings using SPQR-trees. They also suggest to search — among all embeddings with minimum depth — the one providing a *maximum outer face* (i.e., the unbounded region bounded by a maximum number of edges). Also this problem can be solved in linear time using the SPQR-tree data structure.

For graphs representing some data flow such as directed acyclic graphs, a common graph layout method has been suggested by Sugiyama, Tagawa, and Toda [32]. Here, in a first step, the y -coordinates of the vertices are fixed (e.g., using a topological sort). Then in the second step, the vertices are permuted within the layers in order to get a small number of crossings. In the third step, the x -coordinates of the vertices are computed. However, unlike in the topology-shape-metrics method, no guarantee can be given that a digraph that can be drawn without edge crossings, a so-called *upward-planar* graph, will be drawn without crossings. Unfortunately, upward-planarity testing of directed acyclic graphs (DAGs) is NP-hard. However, if the given DAG has only one sink or only one source, then planarity testing can be done in linear time using the SPQR-tree data structure [7]. However, this condition is not true in general. E.g., Figure 3 shows a Sugiyama-style drawing of the same graph shown in

Figure 1, which has several sinks and sources². For these cases, Bertolazzi et al. [5] suggest introducing bends in the edges allowing them to be partially reversed. The authors have suggested a branch-and-bound algorithm based on the SPQR-tree data structure which computes a so-called *quasi-upward* drawing with the minimum number of bends.

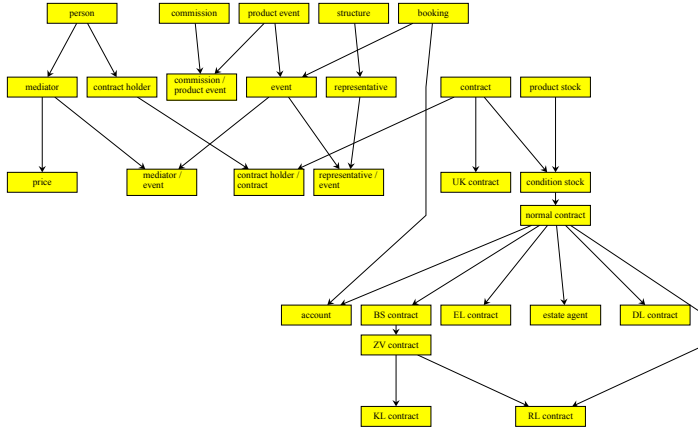


Fig. 3. The same graph as in Figure 1 drawn with a Sugiyama-style method

Drawing clustered graphs is becoming increasingly important in these days when the graphs and data to be displayed get increasingly larger. In *clustered graphs*, some of the nodes may be grouped together. The groups maybe nested, but they may not intersect each other. In a drawing of a clustered graph, such groups of nodes should stay together. Formally, the nodes and edges within one group should stay within a closed convex region. In a *cluster planar drawing*, no edge crossings and only at most one edge-region crossing per edge is allowed. Figure 4 shows a cluster planar drawing of a graph³. Naturally, the idea of the topology-shape-metrics method is also useful for generating cluster drawings. Unfortunately, so far it is unknown, if the problem of cluster planarity testing can be solved in polynomial time. So far, only algorithms are known in the case that the induced subgraphs of the clusters are connected [12,16]. Dahlhaus [12] uses the SPQR-tree data structure in order to test a planar connected clustered graph for cluster planarity in linear time. Unfortunately, in general the clusters induce non-connected subgraphs. Gutwenger et al. [19] have suggested a wider class of polynomially solvable instances using SPQR-trees.

² The drawing has been generated with AGD [1]

³ This drawing has been automatically generated by the GoVisual software (see <http://www.oreas.com>).

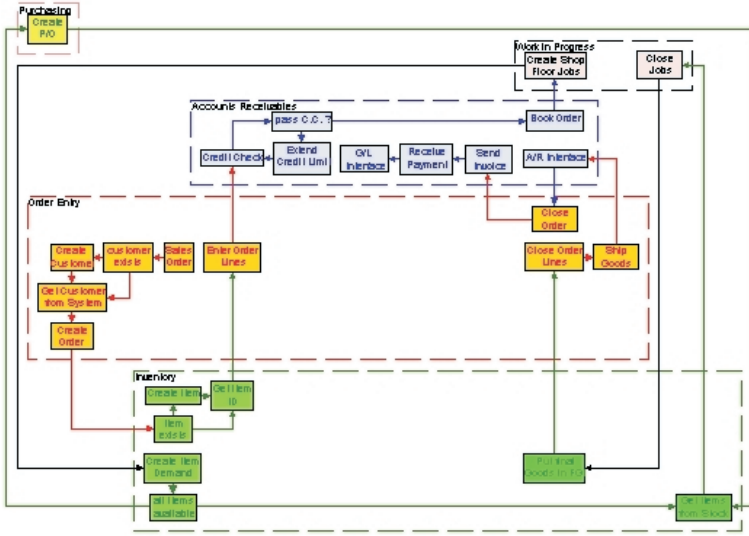


Fig. 4. A planar cluster drawing of a clustered graph

SPQR-trees have also been used in three dimensional graph drawing. Hong [24] uses SPQR-trees in order to get a polynomial time algorithm for drawing planar graphs symmetrically in three dimensions with the maximum number of symmetries. Giacomo et al.[17] show that every series-parallel graph with maximal vertex degree three has a so-called *box-drawing* with $O(n)$ volume. For series-parallel graphs, the corresponding SPQR-tree has no R-vertices.

For further information on graph drawing, see e.g. [3,28,26].

3 The SPQR-Tree Data Structure

We will see that SPQR-trees are only defined for biconnected graphs. However, once a problem has been solved using the SPQR-tree data structure for the biconnected components, then it can mostly be solved using a block-cut tree decomposition (based on the decomposition of G into its biconnected components).

Before introducing the data structure of SPQR-trees, we need some graph theoretic definitions. An undirected multigraph $G = (V, E)$ is *connected* if every pair $v, w \in V$ of vertices in G is connected by a path. A connected multigraph G is *biconnected* if for each triple of distinct vertices u, v, a , there is a path $p : v \xrightarrow{*} w$ such that a is not on p . Let $G = (V, E)$ be a biconnected multigraph and $a, b \in V$. E can be divided into equivalence classes $E_1, \dots, E_k$ such that two edges which lie on a common path not containing any vertex of $\{a, b\}$ except as an endpoint are in the same class. The classes E_i are called the *separation classes* of G with respect to $\{a, b\}$. If there are at least two separation classes, then $\{a, b\}$ is a *separation pair* of G unless (i) there are exactly two separation

classes, and one class consists of a single edge, or (ii) there are exactly three classes, each consisting of a single edge. If G contains no separation pair, G is called *triconnected*.

Let $G = (V, E)$ be a biconnected multigraph, $\{a, b\}$ a separation pair of G , and $E_1, \dots, E_k$ the separation classes of G with respect to $\{a, b\}$. Let $E' = \bigcup_{i=1}^{\ell} E_i$ and $E'' = \bigcup_{i=\ell}^k E_i$ be such that $|E'| \geq 2$ and $|E''| \geq 2$. The two graphs $G' = (V(E'), E' \cup \{e\})$ and $G'' = (V(E''), E'' \cup \{e\})$ are called *split graphs* of G with respect to $\{a, b\}$, where $e = (a, b)$ is a new edge. Replacing a multigraph G by two split graphs is called *splitting* G . Each split graph is again biconnected. The edge e is called *virtual edge* and identifies the split operation. Suppose G is split, the split graphs are split, and so on, until no more split operations are possible. The resulting graphs are called the *split components* of G . They are each either a set of three multiple edges (*triple bond*), or a cycle of length three (*triangle*), or a triconnected simple graph. The split components are not necessarily unique. In a multigraph $G = (V, E)$, each edge in E is contained in exactly one, and each virtual edge in exactly two split components. The total number of edges in all split components is at most $3|E| - 6$. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two split components containing the same virtual edge e . The graph $G' = (V_1 \cup V_2, (E_1 \cup E_2) \setminus \{e\})$ is called a *merge graph* of G_1 and G_2 . The *triconnected components* of G are obtained from its split components by merging the triple bonds into maximal sets of multiple edges (*bonds*) and the triangles into maximal simple cycles (*polygons*). The triconnected components of G are unique [27,35,25].

The triconnected components of a graph are closely related to SPQR-trees. SPQR-trees were originally defined in [13] for planar graphs only. Here, we cite the more general definition given in [14], that also applies to not necessarily planar graphs.

Let G be a biconnected graph. A *split pair* of G is either a separation pair or a pair of adjacent vertices. A *split component* of a split pair $\{u, v\}$ is either an edge (u, v) or a maximal subgraph C of G such that $\{u, v\}$ is not a split pair of C . Let $\{s, t\}$ be a split pair of G . A *maximal split pair* $\{u, v\}$ of G with respect to $\{s, t\}$ is such that, for any other split pair $\{u', v'\}$, vertices u, v, s , and t are in the same split component.

Let $e = (s, t)$ be an edge of G , called the *reference edge*. The SPQR-tree $\mathcal{T}$ of G with respect to e is a rooted ordered tree whose nodes are of four types: S, P, Q, and R. Each node μ of $\mathcal{T}$ has an associated biconnected multigraph, called the *skeleton* of μ . Tree $\mathcal{T}$ is recursively defined as follows:

Trivial Case: If G consists of exactly two parallel edges between s and t , then

$\mathcal{T}$ consists of a single Q-node whose skeleton is G itself.

Parallel Case: If the split pair $\{s, t\}$ has at least three split components

$G_1, \dots, G_k$, the root of $\mathcal{T}$ is a P-node μ , whose skeleton consists of k parallel edges $e = e_1, \dots, e_k$ between s and t .

Series Case: Otherwise, the split pair $\{s, t\}$ has exactly two split components, one of them is e , and the other one is denoted with G' . If G' has cutvertices $c_1, \dots, c_{k-1}$ ($k \geq 2$) that partition G into its blocks $G_1, \dots, G_k$, in this

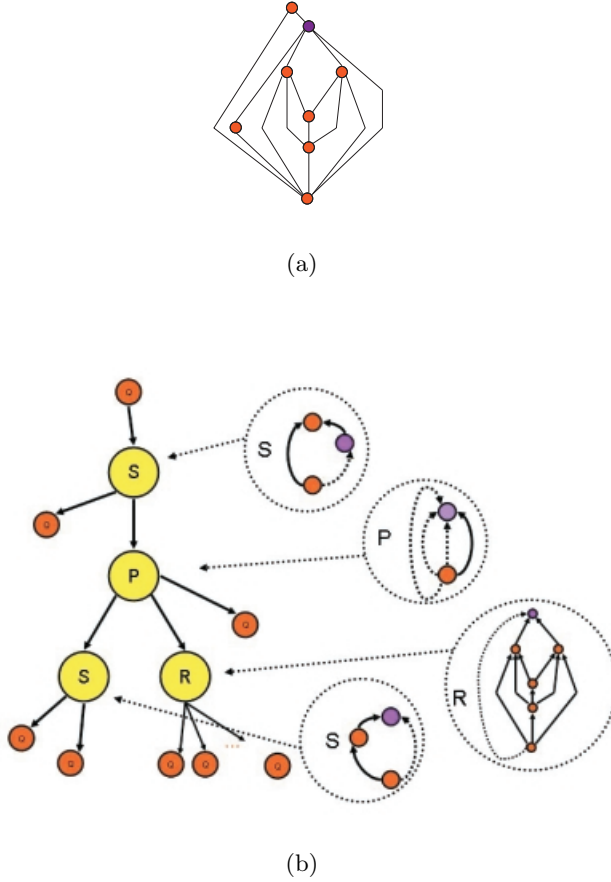


Fig. 5. A graph, its SPQR-tree, and the corresponding skeletons

order from s to t , the root of $\mathcal{T}$ is an S-node μ , whose skeleton is the cycle $e_0, e_1, \dots, e_k$, where $e_0 = e$, $c_0 = s$, $c_k = t$, and $e_i = (c_{i-1}, c_i)$ ($i = 1, \dots, k$).
Rigid Case: If none of the above cases applies, let $\{s_1, t_1\}, \dots, \{s_k, t_k\}$ be the maximal split pairs of G with respect to $\{s, t\}$ ($k \geq 1$), and, for $i = 1, \dots, k$, let G_i be the union of all the split components of $\{s_i, t_i\}$ but the one containing e . The root of $\mathcal{T}$ is an R-node, whose skeleton is obtained from G by replacing each subgraph G_i with the edge $e_i = (s_i, t_i)$.

Except for the trivial case, μ has children $\mu_1, \dots, \mu_k$, such that μ_i is the root of the SPQR-tree of $G_i \cup e_i$ with respect to e_i ($i = 1, \dots, k$). The virtual edge of node μ_i is edge e_i of skeleton of μ . Graph G_i is called the *pertinent graph* of node μ_i . Tree $\mathcal{T}$ is completed by adding a Q-node, representing the reference edge e ,

and making it the parent of μ so that it becomes the root. Figures 5(a) and (b) show a biconnected graph and its corresponding SPQR-tree. The skeletons of the S-, P-, and R-nodes are shown in the right part of Figure 5(b).

Theorem 1. *Let G be a biconnected multigraph and $\mathcal{T}$ its SPQR-tree.*

1. [14] *The skeletons of the internal nodes of $\mathcal{T}$ are in one-to-one correspondence to the triconnected components of G . P-nodes correspond to bonds, S-nodes to polygons, and R-nodes to triconnected graphs.*
2. [21] *There is an edge between two nodes $\mu, \nu \in \mathcal{T}$ if and only if the two corresponding triconnected components share a common virtual edge.*

Each edge in G is associated with a Q-node in $\mathcal{T}$. It is possible to root $\mathcal{T}$ at an arbitrary Q-node μ' , resulting in an SPQR-tree with respect to the edge associated with μ' [14].

During my talk, we consider a slightly different, but equivalent, definition of SPQR-tree. We omit Q-nodes and distinguish between *real edges* (corresponding to edges in G) and *virtual edges* in the skeletons instead. Then, the skeleton of each P-, S-, and R-node is exactly the graph of the corresponding triconnected component.

In the papers based on SPQR-trees, the authors suggest to construct the data structure SPQR-tree in linear time “using a variation of the algorithm of [25] for finding the triconnected components of a graph...[15]”. To our knowledge, until 2000, no correct linear time implementation was publically available. In [21], the authors present a correct linear time implementation of the data structure SPQR-tree. The implementation is based on the algorithm described in [25]. However, some modifications of this algorithm were necessary in order to get a correct implementation. This implementation (in a re-usable form) is publically available in AGD, a library of graph algorithms and data structures for graph layout [2,18]. The only other correct linear implementation of SPQR-trees we are aware of is part of GoVisual [20].

4 The Techniques Used with SPQR-Trees

We have seen that the SPQR-tree data structure represents the decomposition of a (planar) biconnected graph into its triconnected components. It also represents the set of all embeddings of a planar graph. It is often used for problems which are easy solvable if the embedding of the graph is fixed, but more difficult if the embedding is not part of the input. Indeed, problems involving embeddings of a planar graph, are easy to solve for triconnected components, while they are harder for non-triconnected graphs. If we can find a way, to combine the solutions for all the triconnected components in order to construct a solution for the original graph, we have solved the problem. This is how many algorithms proceed. However, this is not straightforward in most cases.

Another technique is to use the integer linear program based on the SPQR-tree data structure suggested in [29] and to combine this with a (mixed) integer

linear program for the problem under consideration. This approach has been successfully applied in [30].

A rather straightforward way is to simply enumerate the set of all embeddings. However, this will take too long in general. Bertolazzi et al. [6] have shown that it makes sense to define only parts of the configuration of the tree representing only partial embeddings. This can be used for getting strong lower bounds within a branch-and-bound algorithm.

The SPQR-decomposition is also useful for problems that are solvable in linear time for series-parallel graphs [17]. In this case, no R-nodes exist in the SPQR-tree. The SPQR-tree decomposition is an alternative way to the standard series-parallel decomposition which has been used so far in the literature [33].

Finally, we suggest a new method which maybe useful for many graph algorithmic problems that are, in general, NP-hard.

References

1. *AGD User Manual (Version 1.1)*, 1999. Technische Universität Wien, Max-Planck-Institut Saarbrücken, Universität zu Köln, Universität Halle. See also <http://www.ads.tuwien.ac.at/AGD/>.
2. D. Alberts, C. Gutwenger, P. Mutzel, and S. Näher. AGD-library: A library of algorithms for graph drawing. In G. F. Italiano and S. Orlando, editors, *Proceedings of the Workshop on Algorithm Engineering (WAE '97)*, Sept. 1997.
3. G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing*. Prentice Hall, 1999.
4. G. Di Battista and R. Tamassia. On-line graph algorithms with SPQR-trees. In M. S. Paterson, editor, *Proc. of the 17th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 443 of *Lecture Notes in Computer Science*, pages 598–611. Springer-Verlag, 1990.
5. P. Bertolazzi, G. Di Battista, and W. Didimo. Quasi upward planarity. In S. Whitesides, editor, *Proc. International Symposium on Graph Drawing*, volume 1547 of *LNCS*, pages 15–29. Springer Verlag, 1998.
6. P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Transactions on Computers*, 49(8):826–840, 2000.
7. P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino. Optimal upward planarity testing of single-source digraphs. *SIAM J. Comput.*, 27(1):132–169, 1998.
8. T. Biedl, G. Kant, and M. Kaufmann. On triangulating planar graphs under the four-connectivity constraint. *Algorithmica*, 19:427–446, 1997.
9. D. Bienstock and C. L. Monma. Optimal enclosing regions in planar graphs. *Networks*, 19:79–94, 1989.
10. D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990.
11. Z.Z. Chen, X. He, and C.-H. Huang. Finding double euler trails of planar graphs in linear time. In *40th Annual Symposium on Foundations of Computer Science*, pages 319–329. IEEE, 1999.
12. E. Dahlhaus. Linear time algorithm to recognize clustered planar graphs and its parallelization. In *Proc. 3rd Latin American Symposium on theoretical informatics (LATIN)*, volume 1380 of *LNCS*, pages 239–248. Springer Verlag, 1998.

13. G. Di Battista and R. Tamassia. Incremental planarity testing. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 436–441, 1989.
14. G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15:302–318, 1996.
15. G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25(5):956–997, 1996.
16. Q.-W. Feng, R.-F. Cohen, and P. Eades. Planarity for clustered graphs. In P. Spirakis, editor, *Algorithms – ESA ’95, Third Annual European Symposium*, volume 979 of *Lecture Notes in Computer Science*, pages 213–226. Springer-Verlag, 1995.
17. E.D. Giacomo, G. Liotta, and S.K. Wismath. Drawing series-parallel graphs on a box. In *Proc. 14th Canadian Conference on Computational Geometry*, 2002.
18. C. Gutwenger, M. Jünger, G. W. Klau, S. Leipert, and P. Mutzel. Graph drawing algorithm engineering with AGD. In S. Diehl, editor, *Software Visualization*, volume 2269 of *LNCS*, pages 307–323. Springer Verlag, 2002.
19. C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, and M. Percan. Advances in c-planarity testing of clustered graphs. In M.T. Goodrich and S.G. Kobourov, editors, *Proc. 10th International Symposium on Graph Drawing*, volume 2528 of *LNCS*, pages 220–235. Springer Verlag, 2002.
20. C. Gutwenger, K. Klein, J. Kupke, S. Leipert, P. Mutzel, and M. Jünger. Graph drawing library by OREAS.
21. C. Gutwenger and P. Mutzel. A linear time implementation of SPQR trees. In J. Marks, editor, *Graph Drawing (Proc. 2000)*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer-Verlag, 2001.
22. C. Gutwenger and P. Mutzel. Graph embedding with maximum external face and minimum depth. Technical report, Vienna University of Technology, Institute of Computer Graphics and Algorithms, 2003.
23. C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA ’2001)*, pages 246–255, Washington, DC, 2001. ACM Press.
24. S. Hong. Drawing graphs symmetrically in three dimensions. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. 9th International Symposium on Graph Drawing (GD 2001)*, volume 2265 of *LNCS*, pages 220–235. Springer Verlag, 2002.
25. J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.
26. M. Jünger and P. Mutzel. *Graph Drawing Software*. Mathematics and Visualization. Springer-Verlag, 2003. to appear.
27. S. MacLaine. A structural characterization of planar combinatorial graphs. *Duke Math. J.*, 3:460–472, 1937.
28. P. Mutzel, S. Leipert, and M. Jünger, editors. *Graph Drawing 2001 (Proc. 9th International Symposium)*, volume 2265 of *LNCS*. Springer Verlag, 2002.
29. P. Mutzel and R. Weiskircher. Optimizing over all combinatorial embeddings of a planar graph. In G. Cornuéjols, R. Burkard, and G. Woeginger, editors, *Proceedings of the Seventh Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 1610 of *LNCS*, pages 361–376. Springer Verlag, 1999.
30. P. Mutzel and R. Weiskircher. Computing optimal embeddings for planar graphs. In D.-Z. Du, P. Eades, V. Estivill-Castro, X. Lin, and A. Sharma, editors, *Computing and Combinatorics, Proc. Sixth Annual Internat. Conf. (COCOON ’2000)*, volume 1858 of *LNCS*, pages 95–104. Springer Verlag, 2000.
31. M. Pizzonia and R. Tamassia. Minimum depth graph embedding. In M. Paterson, editor, *Algorithms – ESA 2000, Annual European Symposium*, volume 1879 of *Lecture Notes in Computer Science*, pages 356–367. Springer-Verlag, 2000.

32. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Syst. Man Cybern.*, SMC-11(2):109–125, 1981.
33. K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *J. Assoc. Comput. Mach.*, 29:623–641, 1982.
34. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.
35. R. Tarjan and J. Hopcroft. Finding the triconnected components of a graph. Technical Report 72-140, Dept. of Computer Science, Cornell University, Ithaca, 1972.

Model Checking and Testing Combined

Doron Peled*

Dept. of Computer Science
The University of Warwick
Coventry, CV4 7AL, UK

Abstract. Model checking is a technique for automatically checking properties of models of systems. We present here several combinations of model checking with testing techniques. This allows checking systems when no model is given, when the model is inaccurate, or when only a part of its description is given.

1 Introduction

Formal verification of programs was pioneered by Floyd [10] and Hoare [15]. The idea of being able to support the correctness of a program with a mathematical proof is very desirable, as the effect of software errors can be catastrophic. Hardware verification is equally important, trying to eliminate the mass manufacturing of bogus electronic devices. It was quickly evident that although a formal verification of systems has a large theoretical appeal, it is restricted with respect to the size of systems it can handle.

The idea of model checking was proposed in the early eighties [5,9,25]. The main idea is simple: restrict the domain of interest to a finite model and check it against a logic specification, as in finite model theory. The finiteness of the model, and the structure of the specification allows devising algorithms for performing the verification. Model checking has become very successful, in particular in the hardware design industry. Recent advances have also contributed to encouraging successes in verifying software.

Basic methods for model checking are based on graph and automata theory and on logic. The particular algorithm depends, in part, on the type of logic used. We survey here explicit state model checking, which translates both the verified system and the specification into automata, and performs automata based (i.e., graph theoretic) algorithms. There are other approaches, including a structural induction on the checked property [5], in particular using the data structure of binary decision diagrams [22], and algorithms based on solving satisfiability [4].

Albeit the success of model checking, the main effort in verifying software is based on testing. Testing is less comprehensive than model checking and is largely informal. It is well expected that some programming and design errors

* This research was partially supported by Subcontract UTA03-031 to The University of Warwick under University of Texas at Austin's prime National Science Foundation Grant #CCR-0205483.

would remain undetected even after an extensive testing effort. Testing is often restricted to sampling the code of the system [18], using some informal ideas of how to achieve a good coverage (e.g., try to cover every node in the flow chart). Testing has several important features, which makes it useful even in cases where model checking may not be directly applicable:

- Testing can be performed on the actual system (with minimal changes).
- Testing can be performed even when there is a severe state space explosion; in fact it does not rely on finiteness.
- Testing does not require modeling of the system.
- Testing can be done even when no precise specification of the checked properties is given, by using the intuition of the tester (which is usually a very experienced programmer or hardware designer).

We survey here several new approaches that combine model checking and testing techniques. These approaches are designed to exploit the benefits of both testing and model checking and alleviate some of their restrictions.

2 Explicit States Model Checking

First order and propositional logic can be used to express properties of states. Each formula can represent a set of states that satisfy it. Thus, a formula can express, for example, an initial condition, an assertion about the final states, or an invariant. However, such logics are *static* in the sense that they represent a collection of states, but not the dynamic evolution between them during the execution of a program.

Modal logics (see, e.g., [16]) extend static logics by allowing the description of a relation between different states. This is in particular appropriate for asserting about concurrent and distributed systems, where we are interested in describing properties related to the sequence of states or events during an execution.

Linear Temporal Logic (LTL) [21] is an instance of modal logics. LTL is often used to specify properties of interleaving sequences [24], modeling the execution of a program. LTL is defined on top of a static logic $\mathcal{U}$, whose formulas describe properties of states. We will use propositional and first order logic as specific instances of $\mathcal{U}$. The syntax of LTL is as follows:

- Every formula of $\mathcal{U}$ is a formula of LTL,
- If φ and ψ are formulas, then so are $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\bigcirc\varphi)$, $(\Diamond\varphi)$, $(\Box\varphi)$, $(\varphi \mathbf{U} \psi)$, and $(\varphi \mathbf{V} \psi)$.

An LTL formula is interpreted over an infinite sequence of states $x_0x_1x_2\dots$. We write ξ^k for the suffix of $\xi = x_0x_1x_2\dots$ starting at x_k , i.e., the sequence $x_kx_{k+1}x_{k+2}\dots$. It is convenient to define the semantics of LTL for an arbitrary suffix ξ^k of a sequence ξ as follows:

- $\xi^k \models \eta$, where η is a formula in the static logic $\mathcal{U}$, when $x_k \models \eta$
- $\xi^k \models (\neg\varphi)$ when not $\xi^k \models \varphi$,

- $\xi^k \models (\varphi \wedge \psi)$ when $\xi^k \models \varphi$ and $\xi^k \models \psi$,
- $\xi^k \models (\bigcirc\varphi)$ when $\xi^{k+1} \models \varphi$,
- $\xi^k \models (\varphi \mathbf{U}\psi)$ when there is an $i \geq k$ such that $\xi^i \models \psi$ and for all j , where $k \leq j < i$, $\xi^j \models \varphi$

The rest of the modal operators can be defined using the following equivalences: $\varphi \vee \psi = \neg((\neg\varphi) \wedge (\neg\psi))$, $\Diamond\varphi = \text{true}\mathbf{U}\varphi$, $\varphi\mathbf{V}\psi = \neg((\neg\varphi)\mathbf{U}(\neg\psi))$, $\Box\varphi = \text{false}\mathbf{V}\varphi$. The modal operator ' $\bigcirc$ ' is called *nexttime*. The formula $\bigcirc\varphi$ holds in a sequence $x_k x_{k+1} x_{k+2} \dots$ when φ holds starting with the next state x_{k+1} , namely in the suffix sequence $x_{k+1} x_{k+2} \dots$. Similarly, $\bigcirc\bigcirc\varphi$ holds provided that φ holds in the sequence $x_{k+2} x_{k+3} \dots$. The modal operator ' $\Diamond$ ' is called *eventually*. The formula $\Diamond\varphi$ holds in a sequence ξ provided that there is a suffix of ξ where φ holds. The modal operator ' $\Box$ ' is called *always*. The formula $\Box\varphi$ holds in a sequence ξ provided that φ holds in every suffix of ξ .

We can construct formulas that combine different modal operators. For example, the formula $\Box\Diamond\varphi$ holds in a sequence ξ provided that for every suffix ξ' of ξ , $\Diamond\varphi$ holds. That is, there is a suffix ξ'' of ξ' where φ holds. In other words, φ holds in ξ 'infinitely often'. The operator ' $\mathbf{U}$ ' is called *until*. Intuitively, $\varphi\mathbf{U}\psi$ asserts that φ holds until some point (i.e., some suffix) where ψ holds. We can view ' $\Diamond$ ' as a special case of ' $\mathbf{U}$ ' since $\Diamond\varphi = \text{true}\mathbf{U}\varphi$.

The simplest class of automata over infinite words is that of Büchi automata [2]. (We describe here a version where the labels are defined on the states rather than on the transitions.) A Büchi automaton $\mathcal{A}$ is a sextuple $\langle \Sigma, S, \Delta, I, L, F \rangle$ such that

- Σ is the finite *alphabet*.
- S is the finite set of *states*.
- $\Delta \subseteq S \times S$ is the *transition relation*.
- $I \subseteq S$ are the *starting states*.
- $L : S \rightarrow \Sigma$ is a *labeling* of the *states*.
- $F \subseteq S$ is the set of *accepting states*.

A *run* ρ of $\mathcal{A}$ on an infinite word $v \in \Sigma^\omega$ corresponds to an infinite path in the automaton graph from an initial state, where the nodes on this path are labeled according to the letters in v . Let $\text{inf}(\rho)$ be the set of states that appear infinitely often in the run ρ (when treating the run as an infinite path). A run ρ of a Büchi automaton $\mathcal{A}$ over an infinite word is *accepting* when $\text{inf}(\rho) \cap F \neq \emptyset$. That is, when some accepting state appears in ρ infinitely often. The *language* $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$ of a Büchi automaton $\mathcal{A}$ consists of all the words accepted by $\mathcal{A}$. We can model the checked system using a Büchi automaton. Finite executions can be artificially completed into infinite ones by adding self loops to terminal (sink) states.

Similarly, we can talk about the language $\mathcal{L}(\varphi)$ of a temporal property φ , referring to the set of sequences satisfying φ . In fact, we can easily translate a propositional LTL formula into a Büchi automaton. In this case, if $\mathcal{P}$ is the set of propositions appearing in φ , then $\Sigma = 2^{\mathcal{P}}$. A simple and practical translation appears in [13]. At worst, the size of the obtained automaton is exponential in

the length of the LTL formula. We assume that the system is modeled by a Büchi automaton with states labeled by $\Sigma = 2^P$ as well. The label of a state reflects the set of propositions that hold in it.

Under the automata theoretic framework for model checking [19,29], we represent both the state space and the specification as automata over the same alphabet. The system model $\mathcal{A}$ satisfies the specification $\mathcal{B}$ if there is an inclusion between the language of the system $\mathcal{A}$ and the language of the specification $\mathcal{B}$, i.e.,

$$\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B}). \quad (1)$$

Let $\overline{\mathcal{L}(\mathcal{B})}$ be the language $\Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$ of words *not accepted* by $\mathcal{B}$, i.e., the *complement* of the language $\mathcal{L}(\mathcal{B})$. Then, the above inclusion (1) can be rewritten as

$$\mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathcal{B})} = \emptyset \quad (2)$$

This means that there is no accepted word of $\mathcal{A}$ that is disallowed by $\mathcal{B}$. If the intersection is nonempty, any element in it is a counterexample to (1).

Implementing the language intersection in (2) is simpler than implementing the language inclusion in (1). Complementing a Büchi automaton is hard [27]. When the source of the specification is an LTL formula φ , we can avoid complementation. This is done by translating the negation of the checked formula φ , i.e., translating $\neg\varphi$ into an automaton $\overline{\mathcal{B}}$ directly rather than translating φ into an automaton $\mathcal{B}$ and then complementing.

In order to define an automaton $\mathcal{A}_1 \cap \mathcal{A}_2$ that accepts the intersection $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ of the languages of $\mathcal{A}_1$ and $\mathcal{A}_2$, we generalize the definition of Büchi automata. The structure of generalized Büchi automata differ from (simple) Büchi automata by allowing multiple accepting sets rather than only one. The structure is a sextuple $\langle \Sigma, S, \delta, I, L, F \rangle$, where $F = \{f_1, f_2, \dots, f_m\}$, and for $1 \leq i \leq m$, $f_i \subseteq S$. The other components are the same as in simple Büchi automata. An accepting run needs to pass through each one of the sets in F infinitely often. Formally, a run ρ of a generalized Büchi automaton is accepting if for each $f_i \in F$, $\inf(\rho) \cap f_i \neq \emptyset$.

We present a simple translation [7] from a generalized Büchi automaton $\langle \Sigma, S, \delta, I, L, F \rangle$ to a (simple) Büchi automaton. If the number of accepting sets $|F|$ is m , we create m separate copies of the set of states S , namely, $\bigcup_{i=1, m} S_i$, where $S_i = S \times \{i\}$ for $1 \leq i \leq m$. Hence, a state of S_i will be of the form (s, i) . Denote by $\oplus_m$ the addition operation changed such that $i \oplus_m 1 = i + 1$, when $1 \leq i < m$, and $m \oplus_m 1 = 1$. This operator allows us to count cyclically from 1 through m . In a run of the constructed Büchi automaton, when visiting the states in S_i , if a copy of a state from f_i occurs, we move to the corresponding successor state in $S_{i \oplus_m 1}$. Otherwise, we move to the corresponding successor in S_i . Thus, visiting accepting states from all the sets in F in increasing order will make the automaton cycle through the m copies.

We need to select the accepting states such that in an accepting run, each one of the copies S_1 through S_m is passed infinitely often. Since moving from one of the sets to the next one coincides with the occurrence of an accepting

state from some f_i , this guarantees that all of the accepting sets occur infinitely often. We can select the Cartesian product $f_i \times \{i\}$ for some arbitrary $1 \leq i \leq m$. This guarantees that we are passing through a state in $f_i \times \{i\}$ on our way to a state in $S_{i \oplus_m 1}$. In order to see a state in $f_i \times \{i\}$ again, we need to go cyclically through all the other copies once more. In the case where the set of accepting sets F of the generalized Büchi automaton is empty, we define the translation as $\langle \Sigma, S, \delta, I, L, S \rangle$, i.e., all the states of the generated Büchi automaton are accepting.

We can now define the intersection of two Büchi automata as a generalized Büchi automaton, and later translate it into a simple Büchi automaton. The intersection is constructed as follows:

$$\mathcal{A}_1 \cap \mathcal{A}_2 = \langle \Sigma, S, \delta, (I_1 \times I_2) \cap S, L, \{(F_1 \times S_2) \cap S, (S_1 \times F_2) \cap S\} \rangle$$

where $S = \{\langle s_1, s_2 \rangle \mid s_1 \in S_1, s_2 \in S_2, L_1(s_1) = L_2(s_2)\}$. That is, we restrict the intersection to states with matching labels. The transition relation δ of the intersection is defined by $((l, q), (l', q')) \in \delta$ iff $(l, l') \in \delta_1$, and $(q, q') \in \delta_2$. The labeling of each state (l, q) in the intersection, denoted $L(l, q)$, is $L_1(l)$ (or equivalently $L_2(q)$).

The intersection in (2) usually corresponds to a more restricted case, where *all* the states of the automaton $\mathcal{A}$ representing the modeled system are accepting. In this restricted case, where the automaton $\mathcal{A}_1$ has all its states accepting and the automaton $\mathcal{A}_2$ is unrestricted, we have

$$\mathcal{A}_1 \cap \mathcal{A}_2 = \langle \Sigma, S, \delta, (I_1 \times I_2) \cap S, L, (S_1 \times F_2) \cap S \rangle, \quad (3)$$

where S , δ and L are defined as above. This is already a simple Büchi automaton. Thus, the accepting states are the pairs with accepting second component. Nevertheless, the more general case of intersection is useful for modeling systems where fairness constraints are imposed. In this case, not all the states of the system automaton are necessarily accepting.

The last building block that is needed for checking (2) is an algorithm for checking the emptiness of the language of a Büchi automaton. This can be done by performing Tarjan's DFS algorithm for finding maximal strongly connected components (MSSCs). The language is nonempty if there is a nontrivial MSSC that is reachable from an initial state, and which contains an accepting state s . In this case, we can find a finite path u from the initial state to s , and a finite path v from s back to itself. We obtain a counterexample for the emptiness of the language of the automaton of the form $u v^\omega$, i.e., an *ultimately periodic* sequence.

3 Combination 1: Black Box Checking

Black box checking (BBC) [23] allows checking whether a system whose model is unavailable satisfies a temporal property. It combines comprehensive verification against a specification, as in model checking, with the direct testing of a black box system. We are given only a limited information about the black box system:

an upper bound on the number of the states, and its possible interactions, which it can observably allow or refuse from each state. We are also given a reliable reset capability, that allows us to force it to start from its initial state. Since the states of the checked system are inaccessible, the temporal specification refers to the sequences of inputs allowed by the system.

According to the black box checking algorithm, we alternate between incremental learning of the system, according to Angluin's algorithm [1], and the black box testing of the learned model against the actual system, using the Vasilevskii-Chou (VC) algorithm [3,30].

Black box checking can be used to verify properties of a system that is representable as a finite transition system (i.e., an automaton with no accepting states) $\mathcal{S} = \langle \Sigma, S, \delta, \iota \rangle$. Here, the states S are not labeled (we initially do not even know them), and there is only one initial state $\iota \in S$ (rather than a set of such states I). The alphabet Σ models the *inputs*, which cause a transition between the states. We assume that the transition relation $\delta \subseteq S \times \Sigma \times S$ is deterministic. We know the possible inputs, and an upper bound n on the number of states $|S| = m$. But we do not know the set of states or the transition relation.

We say that an input a is *enabled* from a state $s \in S$, if there exists $r \in S$, such that $(s, a, r) \in \delta$. Similarly, $a_1 a_2 \dots a_n$ is enabled from s if there is a sequence of states $s_0, s_2, \dots, s_n$ with $s_0 = s$ such that for $1 \leq i \leq n$, $(s_{i-1}, a_i, s_i) \in \delta$. An *execution* of the black box system $\mathcal{S}$ is a finite or infinite sequence of inputs enabled from the initial state. Let $\mathcal{T} \subset \Sigma^*$ be the finite set of executions of $\mathcal{S}$. Since $|\Sigma|$ is finite, if $\mathcal{T}$ is an infinite set, then according to König's Lemma, $\mathcal{S}$ has also infinite executions. We assume that we can perform the following experiments on $\mathcal{S}$:

- **Reset** the system to its initial state.
- Check whether an input a can be currently executed by the system. The system provides us with information on whether a was executable.

An approximation transition system M *accurately models* a system $\mathcal{S}$ if $\mathcal{S}$ and M have exactly the same executions. We use Angluin's learning algorithm [1] to guide experiments on the system $\mathcal{S}$ and produce a *minimized* finite transition system representing it.

The basic data structure of Angluin's algorithm consists of two finite sets of finite strings V and W over the alphabet Σ , and a table t . The set V is prefix closed, and thus contains in particular the empty string ε . The rows of the table t are the strings in $V \cup V.\Sigma$, while the columns are the strings in W . The set W must also contain the empty string. Let $t(v, w) = 1$ when the sequence of transitions vw is a successful execution of $\mathcal{S}$, and 0 otherwise. The entry $t(v, w)$ can be computed by performing the experiment **Reset** vw .

The sequences in V are the *access* sequences, as they are used to access the different states of the system $\mathcal{S}$, when starting the execution from its initial state. The sequences in W are called the *separating sequences*, as their goal is to separate between different states of the constructed transition system. Namely, if $v, v' \in V$ lead from the initial state into a different state, than we will find

some $w \in W$ such that $\mathcal{S}$ allows either vw or $v'w$ as a successful experiment, but not both.

We define an equivalence relation $\equiv \text{mod}(W)$ over strings in Σ^* as follows: $v_1 \equiv v_2 \text{ mod}(W)$ when the two rows, of v_1 and v_2 in the table t are the same. Denote by $[v]$ the equivalence class that includes v . A table t is *closed* if for each $va \in V.\Sigma$ such that $t(v, \varepsilon) \neq 0$ there is some $v' \in V$ such that $va \equiv v' \text{ mod}(W)$. A table is *consistent* if for each $v_1, v_2 \in V$ such that $v_1 \equiv v_2 \text{ mod}(W)$, either $t(v_1, \varepsilon) = t(v_2, \varepsilon) = 0$, or for each $a \in \Sigma$, we have that $v_1a \equiv v_2a \text{ mod}(W)$. Notice that if the table is not consistent, then there are $v_1, v_2 \in V$, $a \in \Sigma$ and $w \in W$, such that $v_1 \equiv v_2 \text{ mod}(W)$, and exactly one of v_1aw and v_2aw is an execution of $\mathcal{S}$. This means that $t(v_1a, w) \neq t(v_2a, w)$. In this case we can add aw to W in order to separate v_1 from v_2 .

Given a closed and consistent table t over the sets V and W , we construct a *proposed approximation* $M = \langle S, s_0, \Sigma, \delta \rangle$ as follows:

- The set of states S is $\{[v] \mid v \in V, t(v, \varepsilon) \neq 0\}$.
- The initial state s_0 is $[\varepsilon]$.
- The transition relation δ is defined as follows: for $v \in V, a \in \Sigma$, the transition from $[v]$ on input a is enabled iff $t(v, a) = 1$ and in this case $\delta([v], a) = [va]$.

The facts that the table t is closed and consistent guarantee that the transition relation is well defined. In particular, the transition relation is independent of which state v of the equivalence class $[v]$ we choose; if v, v' are two equivalent states in V , then for all $a \in \Sigma$ we have that $[va]$ coincides with $[v'a]$ (by consistency) and is equal to $[u]$ for some $u \in V$ (by closure).

There are two basic steps used in the learning algorithms for extending the table t :

- add.rows*(v) : Add v to V . Update the table by adding a row va for each $a \in \Sigma$ (if not already present), and by setting $t(va, w)$ for each $w \in W$ according to the result of the experiment **Reset** vaw .
- add.column*(w) : Add w to W . Update the table t by adding the column w , i.e., set $t(v, w)$ for each $v \in V \cup V.\Sigma$, according to the experiment **Reset** vw .

The Angluin algorithm is executed in phases. After each phase, a new proposed approximation M is generated. The proposed approximation M may not agree with the system $\mathcal{S}$. We compare M and $\mathcal{S}$. If the comparison succeeds, the learning algorithm terminates. If it does not, we obtain a run σ on which M and $\mathcal{S}$ disagree, and add all its prefixes to the set of rows V . We then execute a new phase of the learning algorithm, where more experiments due to the prefixes of σ and the requirement to obtain a closed and consistent table are called for.

Comparing an approximation M with $\mathcal{S}$ is very expensive, as will be explained below. We try to eliminate it by using the current approximation M for model checking the given temporal property. If this results in a counterexample (i.e., a sequence of M that satisfies the negation of the checked property), then in particular there is one of the form uv^ω . We need to check whether the actual system $\mathcal{S}$ accepts this sequence. It is sufficient to check whether $\mathcal{S}$ accepts uv^n .

In this case, using the pigeon hole principle, since $\mathcal{S}$ has at most n states, the n repetitions of v must pass (start or terminate) at least twice in the same state. This means that $\mathcal{S}$ also accepts uv^ω . In this case, we have found a bad execution of the original system and we are done. If $\mathcal{S}$ does not accept uv^ω , the smallest prefix of it (in fact, of uv^n) that is not accepted by $\mathcal{S}$ is a sequence distinguishing between M and $\mathcal{S}$. We can use this prefix to start the next phase of the learning algorithm, which will obtain a better approximation. Finally, if M happens to satisfy the temporal property, we need to perform the comparison between M and $\mathcal{S}$, as explained below.

An incremental step of learning starts with either an empty table t (and empty sets V and W), or with a table that was prepared in the previous step, and a sequence σ that distinguishes the behavior of the proposed approximation (as constructed from the table t) and the actual system. The subroutine ends when the table t is closed and consistent, hence a proposed approximation can be constructed from it.

A *spanning tree* of an transition system $M = \langle \Sigma, S, \delta, \iota \rangle$ is a graph $G = \langle \Sigma, S, \delta', \iota \rangle$ whose transition relation $\delta' \subseteq \delta$ is generated using the following depth first search algorithm, called initially with $explore(\iota)$.

subroutine $explore(s)$:

 set $old(s)$;

 for each $a \in \Sigma$ do

 if $\exists s' \in S$ such that $(s, a, s') \in \delta$ and $\neg old(s')$

 add (s, a, s') to δ' ;

$explore(s')$;

Let T be the corresponding executions of G . Notice that in Angluin's algorithm, when an approximation M has been learned, the set V of access sequences includes the runs of a spanning tree of M .

Let M be a transition system with a set of states S . A function $ds : S \rightarrow 2^{\Sigma^*}$ is a *separation function* of M if for each $s, s' \in S$, $s \neq s'$, there are $w \in ds(s)$ and $w' \in ds(s')$, such that some $\sigma \in prefix(w) \cap prefix(w')$ is enabled from exactly one of s and s' (thus, σ separates s from s'). A simple case of a separation function is a constant function, where for each s, s' , $ds(s) = ds(s')$. In this case, we have *separation set* [20]. The set W generated by Angluin's algorithm is a separation set.

Comparing an approximation M with a finite state system $\mathcal{S}$ can be performed using the Vasilevskii-Chow [30,3] algorithm. As a preparatory step, we require the following:

- A spanning tree G for M , and its corresponding runs T .
- A separation function [20] ds , such that for each $s \in S$, $|ds(s)| \leq n$, and for each $\sigma \in ds(s)$, $|\sigma| \leq n$.

Let $\Sigma^{\leq k}$ be all the strings over Σ with length smaller or equal to k . Further, let m be the number of states of the transition system M . We do the experiments with respect to a conjectured maximal size that grows incrementally up to upper

bound n on the number of states of $\mathcal{S}$. That is, our comparison is correct as long as representing $\mathcal{S}$ faithfully (using a finite transition system) does not need to have more than n states. The black box testing algorithm prescribes experiments of the form **Reset** σ ρ , performed on $\mathcal{S}$, as follows:

- The sequence σ is taken from $T.\Sigma^{\leq n-m+1}$.
- Run σ from the initial state ι of M . If σ is enabled from ι , let s be the state of M that is reached after running σ . Then ρ is taken from the set $ds(s)$.

The complexity of the VC algorithm is $\mathcal{O}(m^2 n |\Sigma|^{n-m+1})$.

4 Combination 2: Adaptive Model Checking

Model checking is performed with respect to a model. Because of the possibility of modeling errors, when a counterexample is found, it still needs to be compared against the actual system. If the counterexample does not reflect an actual execution of the system, i.e., it is a *false negative*, the model needs to be refined, and the automatic verification is repeated.

In *adaptive model checking* (AMC) [14], we deal with the problem of model checking in the presence of an inaccurate model. We suggest a methodology in which model checking is performed on some preliminary model. Then, if a counterexample is found, it is compared with the actual system. This results in either the conclusion that the system does not satisfy its property, or an automatic refinement of the model. The adaptive model checking approach can be used in the following cases:

- When the model includes a modeling error.
- After some previously occurring bug in the system was corrected.
- When a new version of the system is presented.
- When a new feature is added to the system.

The adaptive model checking methodology is a variant of black box checking. While the latter starts the automatic verification process without having a model, adaptive model checking assumes some initial model, which may be inaccurate. The observation is that the inaccurate model is still useful for the verification. First, it can be used for performing model checking. Caution must be taken as any counterexample found must still be compared against the actual system; in the case that no counterexample is found, no conclusion about the correctness of the system can be made. In addition, the assumption is that the given model shares some nontrivial common behavior with the actual system. Thus, the current model can be used for obtaining a better model. The methodology consists of the following steps.

1. Perform model checking on the given model.
2. Provided that an error trace was found, compare the error trace with the actual system. If this is an actual execution of the system, report it and stop.

3. Start the learning algorithm. Unlike the black box checking case, we do not begin with $V = W = \{\varepsilon\}$. Instead, we initiate V and W to values obtained from the given model M as described below.
4. If no error trace was found, we can either decide to terminate the verification attempt (assuming that the model is accurate enough), or perform some black box testing algorithm, e.g., VC, to compare the model with the actual system. A manual attempt to correct or update the model is also possible. Notice that black box testing is a rather expensive step that should be eliminated.

In the black box checking algorithm, we start the learning with an empty table t , and empty sets $V = W = \{\varepsilon\}$. As a result, the black box checking algorithm alternates between the incremental learning algorithm and a black box testing (VC algorithm) of the proposed transition system with the actual system. Applying the VC algorithm may be very expensive. In the adaptive model checking case, we try to guide the learning algorithm using the already existing (albeit inaccurate) model. We assume that the modified system has a nontrivial similarity with the model. This is due to the fact that changes that may have been made to the system were based on the old version of it. We can use the following:

1. A false negative counterexample σ found (i.e., a sequence σ that was considered to be a counterexample when checking the nonaccurate model, but has turned out not to be an actual execution of the actual system $\mathcal{S}$). We perform learning experiments with σ (and its prefixes).
2. The runs T of a spanning tree G of the model M as the initial set of access sequences V . We precede the learning algorithm by performing **for each** $v \in T$ **do** *add_rows*(v).
3. A set of separating sequences $DS(M)$ calculated [20] for the states of M as the initial value of the set W . Thus, we precede the learning algorithm by setting $W = DS(M)$.

Thus, we attempt to speed up the learning, using the existing model information, but with the learning experiments now done on the actual current system $\mathcal{S}$. We experimented with the choices 1 + 2 (in this case we set $W = \{\varepsilon\}$), 1 + 3 (in this case we set $V = \{\varepsilon\}$) and 1 + 2 + 3.

If the model M accurately models a system $\mathcal{S}$, starting with the aforementioned choices of V and W the above choices allow Angluin's algorithm to learn M accurately, without the assistance of the (time expensive) black box testing (the VC algorithm) [14]. Furthermore, the given initial settings do not prevent from learning correctly a finite representation of $\mathcal{S}$. Of course, when AMC is applied, the assumption is that the system $\mathcal{S}$ deviates from the model M . However, if the changes to the system are modest, the proposed initial conditions are designed to speed up the adaptive learning process.

5 Combination 3: Unit Checking

There are two main principles that guide testers in generating test cases. The first principle is *coverage* [26], where the tester attempts to exercise the code in a way that reveals maximal errors with minimal effort. The second principle is based on the tester's intuition; the tester *inspects* the code in pursuit of suspicious executions. In order to reaffirm or alleviate a suspicion, the tester attempts to exercise the code through these executions.

In unit testing, only a small piece of the code, e.g., a single procedure or a collection of related procedures, is checked. It is useful to obtain some automated help in generating a *test harness* that will exercise the appropriate executions. Generating a test condition can be done by calculating the path condition [11].

Unit checking [12] allows the symbolic verification of a unit of code and the generation of test cases. A common restriction of model checking that is addressed by unit checking is that model checking is usually applied to a fully initialized program, and assumes that all the procedures used are available. Unit checking is based on a combination of model checking and theorem proving principles. The user gives a specification for paths along which a trouble seems to occur. The paths in the program flow chart are searched for possible executions that satisfy the specification. Path conditions are symbolically calculated and instantiations that can derive the execution through them are suggested. We allow a temporal specification based on both program counters and program variables.

A unit of code needs to work in the presence of other parts of code: the program that calls it, and the procedures that are called from it. In order to check a unit of code, we need to provide some representation for these other parts. A *driver* for the checked unit of code is replaced by an assertion on the relation between the variables at the start of executing the unit. *Stubs* for procedures that were not provided are replaced by further assertions, which relate the values of the variables at the beginning of the execution of the procedure with their values at the end. This allows us to check parts of the code, rather than a complete system at once. The advantages of our approach are:

- Combating state space explosion by searching through paths in the flow chart rather than through the execution sequences. One path can correspond to multiple (even infinitely many) executions.
- Compositionality. Being able to check part of the code, rather than all of it.
- Parametric and infinite state space verification.
- The automatic generation of test cases, given as path conditions.

A flow chart of a program or a procedure is a graph, with nodes corresponding to the transitions, and edges reflecting the flow of control between the nodes. There are several kinds of nodes. Most common are a box containing an assignment, a diamond containing a condition, and an oval denoting the beginning or end of the program (procedure). Edges exiting from a diamond node are marked with either 'yes' or 'no' to denote the success or failure of the condition, respectively.

A *state* of a program is a function assigning values to the program variables, including the program counters. Each transition consists of a *condition* and a

transformation. Some of the conditions are implicit to the text of the flow chart node, e.g., a check that the program counter has a particular value in an assignment node. Similarly, part of the transformation is implicit, in particular, each transition includes the assignment of a new value to the program counter. The change of the program counter value corresponds to passing an edge out of one node and into another. An *execution* of a program is a finite sequence of states $s_1 s_2 \dots s_n$, where each state s_{i+1} is obtained from its predecessor s_i by executing a transition. This means that the condition for the transition to execute holds in s_i , and the transformation associated with the transition is applied to it.

A *path* of a program is a consecutive sequence of nodes in the flow chart. The projection of an execution sequence on the program counter values is a path through the nodes labeled with these values in the corresponding flow chart. Thus, in general, a path may correspond to multiple executions. A *path condition* is a first order predicate that expresses the condition to execute the path, starting from a given node. In deterministic code, when we start to execute the code from the first node in the path in a state that satisfies the path condition, we are guaranteed to follow that path.

Unit checking combines ideas from testing, verification and model checking. We first compile the program into a flow chart. We keep separately the structure of the flow chart, abstracting away all the variables. We also obtain a collection of *atomic transitions* that correspond to the basic nodes of the flow chart.

We specify the program paths that are suspected of having some problem (thus, the specification is given ‘in the negative’). The specification corresponds to the tester’s intuition about the location of an error. For example, a tester that observes the code may suspect that if the program progresses through a particular sequence of instructions, it may cause a division by zero. The tester can use a temporal specification to express paths. The specification can include assertions on both the program counter values (program location labels), and the program variables.

A model checker generates paths that fit the restrictions on the program counters appearing in the specification. Given a path, it uses the transitions generated from the code in order to generate the *path condition*. The assertions on the program variables that appear in the specification are integrated into the generated path condition, as will be explained below. The path condition describes values for the program variables that will *guarantee* (in the sequential case, or *allow*, in the nondeterministic case, e.g., due to concurrency) passing through the path. Given a path, we can then instantiate the path conditions with actual values so that they will form test cases. In this way, we can also generate test cases that consist of paths and their initial conditions.

There are two main possibilities in calculating path conditions: *forward* [17] and *backward* [8]. We describe here the backwards calculation. The details of the forward calculation can be found in [12].

An *accumulated path condition* is the condition to move from the current edge in the calculation to the end of the path. The current edge moves at each step of the calculation of the path condition backwards over one node to the previous edge. We start with the condition *true* at the end of the path (i.e.,

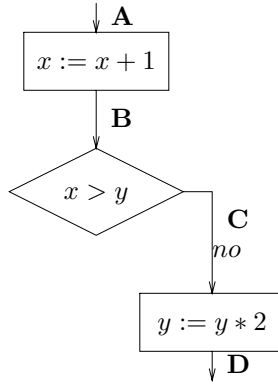


Fig. 1. A path

after its last node). When we pass (on our way back) over a diamond node, we either conjoin it as is, or conjoin its negation, depending on whether we exited this node with a *yes* or *no* edge, respectively. When we pass an assignment, we “relativize” the path condition φ with respect to it; if the assignment is of the form $x := e$, where x is a variable and e is an expression, we substitute e instead of each free occurrence of x in the path condition. This is denoted by $\varphi[e/x]$.

Calculating the path condition for the example in Figure 1 backwards, we start at the end of the path, i.e., the edge **D**, with a path condition *true*. Moving backwards through the assignment $y := y * 2$ to the edge **C**, we substitute every occurrence of y with $y * 2$. However, there are no such occurrences in the accumulated path condition *true*, so the accumulated path condition remains *true*. Progressing backwards to the edge **B**, we now conjoin the negation of the condition $x > y$ (since the edge **C** is labeled *no*), obtaining $\neg(x > y)$. This is now the condition to execute the path from **B** to **D**. Passing further back to the edge **A**, we have to relativize the accumulated path condition $\neg(x > y)$ with respect to the assignment $x := x + 1$, which means replacing the occurrence of x with $x + 1$, obtaining the same path condition as in the forward calculation, $\neg(x + 1 > y)$.

We limit the search by imposing a property of the paths we are interested in. The property may mention the labels that such paths pass through and some relationship between the program variables. It can be given in various forms, e.g., as an LTL formula. We are only interested in properties of finite sequences; checking for cycles in the symbolic framework is, in general, impossible, since we cannot identify repeated states.

We use LTL specification limited to finite executions. This means that $\bigcirc\varphi$ holds in a suffix of a sequence if we are not already in the last state. We also use $\bigcirc\varphi = \neg\bigcirc\neg\varphi$.

The LTL specification is translated into a finite state automaton. The algorithm is similar to the one described in [13], relativized to finite sequences, as in [11], with further optimizations to reduce the number of states generated.

The *property automaton* is $A = \langle S^A, \delta^A, I^A, L^A, F^A \rangle$. Each property automaton node is labeled by a set of negated or non-negated basic formulas. The flow chart can also be denoted as an automaton $B = \langle S^B, \delta^B, I^B, L^B, S^B \rangle$ (where all the nodes are accepting, hence $F^B = S^B$). Each node in S^B is labeled by (1) a single program counter value (2) a node shape, e.g., box or a diamond, respectively), and (3) an assignment or a condition, respectively.

The intersection $A \times B$ is $\langle S^{A \times B}, \delta^{A \times B}, I^{A \times B}, L^{A \times B}, F^{A \times B} \rangle$. The nodes $S^{A \times B} \subseteq S^A \times S^B$ have matching labels: the program counter of the flow chart must satisfy the program counter predicates labeling the property automaton nodes. The transitions are $\{(\langle a, b \rangle, \langle a', b' \rangle) \mid (a, a') \in \delta^A \wedge (b, b') \in \delta^B\} \cap (S^{A \times B} \times S^{A \times B})$. We also have $I^{A \times B} = (I^A \times I^B) \cap S^{A \times B}$, and $F^{A \times B} = (F^A \times S^B) \cap S^{A \times B}$. Thus, acceptance of the intersection automaton depends only on the A automaton component being accepting. The label on a matched pair $\langle a, b \rangle$ in the intersection contains the separate labels of a and b .

One intuition behind the use of a temporal formula to constrain the search is that a human tester that inspects the code usually has suspicion about some execution paths. The temporal formula specifies these paths. For example, a path that passes through label l_2 twice may be suspicious of leading to some incorrect use of resources. We may express such paths in LTL as

$$(\neg at\ l_2)\mathcal{U}(at\ l_2 \wedge \bigcirc((\neg at\ l_2) \wedge ((\neg at\ l_2)\mathcal{U}at\ l_2))). \quad (4)$$

This formula can be translated to the property automaton that appears on the left in Figure 2. The initial nodes are denoted with an incoming edge without a source node. The accepting nodes are denoted with a double circle.

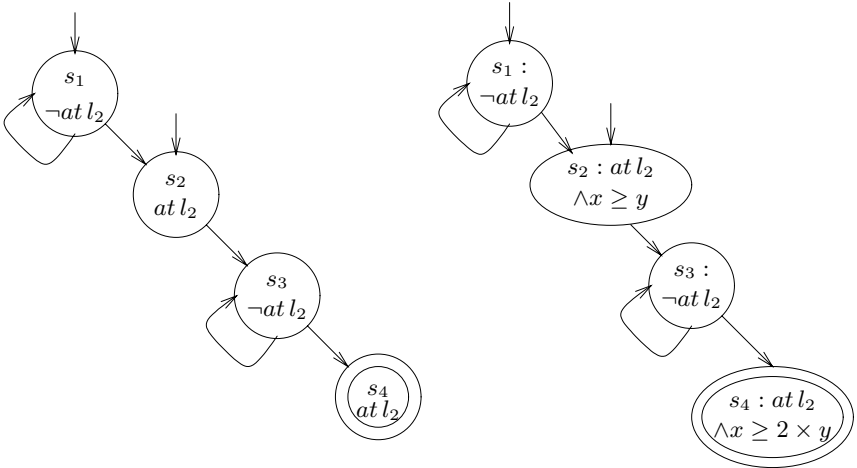


Fig. 2. A property automaton

The specification formula (4) is based only on the program counters. Suppose that we also want to express that when we are at the label l_2 for the first time, the value of x is greater or equal to the value of y , and that when we are at the label l_2 the second time, x is at least twice as big as y . We can write the specification as follows:

$$(\neg at\ l_2)\mathcal{U}(at\ l_2 \wedge x \geq y \wedge \bigcirc((\neg at\ l_2) \wedge ((\neg at\ l_2)\mathcal{U}(at\ l_2 \wedge x \geq 2 \times y)))) \quad (5)$$

An automaton obtained by the translation appears on the right in Figure 2. The translation from a temporal formula to an automaton results in the program variables assertions $x \geq y$ and $x \geq 2 \times y$ labeling the second and fourth nodes. They do not participate in the automata intersection, hence do not contribute further to limiting the paths. Instead, they are added to the path condition in the appropriate places. The conjunction of the program variables assertions labeling the property automaton are assumed to hold in the path condition *before* the effect of the matching flow chart node.

In order to take into account program variables assertions from the property automaton, we can transform the currently checked path as follows. Observe that each node in the intersection is a pair (a, b) , where a is a property automaton node, and b is a flow chart node in the current path. For each such pair, when the node a includes some program variables assertions, we insert a new diamond node to the current path, just before b . The inserted node contains as its condition the conjunction of the program variables assertions labeling the node a . The edge between the new diamond and b is labeled with ‘yes’ corresponding to the case where the condition in a holds. The edge that was formerly entering b now enters the new diamond.

In symbolic execution, we are often incapable of comparing states, consequently, we cannot check whether we reach the same state again. We may not assume that two nodes in the flow chart with the same program counter labels are the same, as they may differ because of the values of the program variables. We also may not assume that they are different, since the values of the program variables may be the same. One solution is to allow the user to specify a limit n on the number of repetitions that we allow each flow chart node, i.e., a node from S^B , to occur in a path. Repeating the model checking while incrementing n , we eventually cover any length of sequence. Hence, in the limit, we cover every path, but this is of course impractical.

In unit testing, when we want to check a unit of code, we may need to provide drivers for calling the checked procedure, and stubs simulating the procedures used by our checked code. Since our approach is logic based, we use a *specification* for drivers and stubs, instead of using their code. Instead of using a stub, our method prescribes replacing a procedure with an assertion that relates the program variables *before* and *after* its execution. We call such assertions *stub specifications*, and adapt the path condition calculation to handle nodes that include them [12].

6 Conclusions

We described several combinations of model checking and testing. In model checking, we assume a given model of the checked system. In black box checking, no model is given, and we can only observe whether the system allows some input from its current state or not. In adaptive model checking, we are given a model, but it may be inaccurate. Finally, in unit checking, we are given a description of a part of the code and may want to verify some of its properties in isolation.

References

1. D. Angluin, Learning Regular Sets from Queries and Counterexamples, *Information and Computation*, 75, 87–106 (1978).
2. J. R. Büchi. On a decision method in restricted second order arithmetic, *Proceedings of the International Congress on Logic, Method and Philosophy in Science* 1960, Stanford, CA, 1962. Stanford University Press, 1–12.
3. T. S. Chow, Testing software design modeled by finite-state machines, *IEEE transactions on software engineering*, SE-4, 3, 1978, 178–187.
4. E. M. Clarke, A. Biere, R. Raimi, Yunshan Zhu, Bounded Model Checking Using Satisfiability Solving, *Formal Methods in System Design* 19 (2001), 7–34.
5. E. M. Clarke, E. A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic. *Workshop on Logic of Programs*, Yorktown Heights, NY, *Lecture Notes in Computer Science* 131, Springer-Verlag, 1981, 52–71.
6. E.M. Clarke, O. Grumberg, D. Peled, *Model Checking*, MIT Press, 2000.
7. C. Courcoubetis, M. Y. Vardi, P. Wolper, M. Yannakakis, Memory efficient algorithms for the verification of temporal properties, *Formal Methods in System Design*, Kluwer, 1(1992), 275–288.
8. E.W. Dijkstra, Guarded commands, nondeterminacy and formal derivation of programs, *Communication of the ACM* 18(8), 1975, 453–457.
9. E. A. Emerson, E. M. Clarke, Characterizing correctness properties of parallel programs using fixpoints, *International Colloquium on Automata, Languages and Programming*, *Lecture Notes in Computer Science* 85, Springer-Verlag, July 1980, 169–181.
10. R. Floyd, Assigning meaning to programs, *Proceedings of symposium on applied mathematical aspects of computer science*, J.T. Schwartz, ed. American Mathematical Society,
11. E. L. Gunter, D. Peled, Temporal debugging for concurrent systems, *TACAS 2002*, Grenoble, France, *LNCS* 2280, Springer, 431–444.
12. E. L. Gunter, D. Peled, Unit checking: symbolic model checking for a unit of code, in N. Dershovitz (ed.), *Zohar Manna Festschrift*, *LNCS*, Springer-Verlag.
13. R. Gerth, D. Peled, M.Y. Vardi, P. Wolper, Simple On-the-fly Automatic Verification of Linear Temporal Logic, *PSTV95, Protocol Specification Testing and Verification*, 3–18, Chapman & Hall, 1995, 19–32.
14. A. Groce, D. Peled, M. Yannakakis, Adaptive Model Checking, *TACAS 2002*, *LNCS* 2280, 357–370.
15. C. A. R. Hoare, An axiomatic basis for computer programming, *Communication of the ACM* 12(1969), 576–580.

16. G. E. Hughes, M. J. Cresswell, *A New Introduction to Modal Logic*, Routledge, 1996.
17. J.C. King, Symbolic Execution and Program Testing, *Communication of the ACM*, 17(7), 1976, 385–395.
18. G.J. Myers, *The Art of Software Testing*, John Wiley and Sons, 1979.
19. R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1994.
20. D. Lee, M. Yannakakis, Principles and methods of testing finite state machines – a survey, *Proceedings of the IEEE*, 84(1996), 1090–1126.
21. Z. Manna, A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, 1991.
22. K. L. McMillan, *Symbolic Model Checking*, Kluwer Academic Press, 1993.
23. D. Peled, M. Y. Vardi, M. Yannakakis, Black Box Checking, *Black Box Checking, FORTE/PSTV 1999*, Beijing, China.
24. A. Pnueli, The temporal logic of programs, 18th IEEE symposium on Foundation of Computer Science, 1977, 46–57.
25. J. P. Quielle, J. Sifakis, Specification and verification of concurrent systems in CESAR, *Proceedings of the 5th International Symposium on Programming*, 1981, 337–350.
26. S Rapps, E. J. Weyuker, Selecting software test data using data flow information, *IEEE Transactions on software engineering*, SE-11 4(1985), 367–375.
27. W. Thomas, Automata on infinite objects, In *Handbook of Theoretical Computer Science*, vol. B, J. van Leeuwen, ed., Elsevier, Amsterdam (1990) 133–191.
28. R. E. Tarjan, Depth first search and linear graph algorithms, *SIAM Journal of computing*, 1(1972), 146–160.
29. M. Y. Vardi, P. Wolper, An automata-theoretic approach to automatic program verification, *Proceedings of the 1st Annual Symposium on Logic in Computer Science IEEE*, 1986, 332–344.
30. M. P. Vasilevskii, Failure diagnosis of automata, *Kibernetika*,

Logic and Automata: A Match Made in Heaven

Moshe Y. Vardi*

Rice University, Department of Computer Science, Houston, TX 77005-1892, USA

One of the most fundamental results connecting mathematical logic to computer science is the Büchi-Elgot-Trakhtenbrot Theorem [1,2,6], established in the early 1960s, which states that finite-state automata and monadic second-order logic (interpreted over finite words) have the same expressive power, and that the transformations from formulas to automata and vice versa are effective. In this talk, I survey the evolution of this beautiful connection and show how it provides an algorithmic tool set for automated reasoning. As a running example, I will use temporal-logic reasoning and show how one goes from standard nondeterministic automata on finite words to nondeterministic automata on infinite words [10] and trees [9], to alternating automata on infinite words [7] and trees [4], to two-way alternating automata on infinite words [3] and trees [8,5], all in the search of powerful algorithmic abstractions.

References

1. J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundle. Math.*, 6:66–92, 1960.
2. C. Elgot. Decision problems of finite-automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.
3. O. Kupferman, N. Piterman, and M.Y. Vardi. Extended temporal logic revisited. In *Proc. 12th International Conference on Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 519–535, August 2001.
4. O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.
5. U. Sattler and M.Y. Vardi. The hybrid μ -calculus. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. 1st Int'l Joint Conf. on Automated Reasoning*, Lecture Notes in Computer Science 2083, pages 76–91. Springer-Verlag, 2001.
6. B.A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math. J.*, 3:101–131, 1962. Russian; English translation in: AMS Transl. 59 (1966), 23–55.
7. M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, Berlin, 1996.

* Supported in part by NSF grants CCR-9988322, CCR-0124077, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation. URL: <http://www.cs.rice.edu/~vardi>.

8. M.Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th International Coll. on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer-Verlag, Berlin, July 1998.
9. M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–221, April 1986.
10. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

Pushdown Automata and Multicounter Machines, a Comparison of Computation Modes^{*}

(Extended Abstract)

Juraj Hromkovič¹ and Georg Schnitger²

¹ Lehrstuhl für Informatik I, Aachen University RWTH,
Ahornstraße 55, 52 074 Aachen, Germany

² Institut für Informatik, Johann Wolfgang Goethe University,
Robert Mayer Straße 11–15, 60054 Frankfurt am Main, Germany

Abstract. There are non-context-free languages which are recognizable by randomized pushdown automata even with arbitrarily small error probability. We give an example of a context-free language which cannot be recognized by a randomized pda with error probability smaller than $\frac{1}{2} - O(\frac{\log_2 n}{n})$ for input size n . Hence nondeterminism can be stronger than probabilism with weakly-unbounded error.

Moreover, we construct two deterministic context-free languages whose union cannot be accepted with error probability smaller than $\frac{1}{3} - 2^{-\Omega(n)}$, where n is the input length. Since the union of any two deterministic context-free languages can be accepted with error probability $\frac{1}{3}$, this shows that $\frac{1}{3}$ is a sharp threshold and hence randomized pushdown automata do not have amplification.

One-way two-counter machines represent a universal model of computation. Here we consider the polynomial-time classes of multicounter machines with a constant number of reversals and separate the computational power of nondeterminism, randomization and determinism.

Keywords: complexity theory, randomization, nondeterminism, pushdown automata, multicounter machines

1 Introduction

A separation of nondeterminism, randomization and determinism for polynomial-time computation is probably the central problem of theoretical computer science. Because of the enormous hardness of this problem many researchers consider restricted models of computations (see, for instance, [1,2,3,4,5,6,7,9,10,12,13,15,17,18,19]). This line of research has started with the study of simple models like one-way finite automata and two-party communication protocols and continues by investigating more and more complex models of computation.

^{*} The work of this paper has been supported by the DFG Projects HR 14/6-1 and SCHN 503/2-1.

The goal of this paper is to establish new results separating randomization from determinism and nondeterminism as well as to contribute to the development of proof techniques for this purpose. The computing models considered here are pushdown automata and multicounter machines.

1.1 Randomized Pushdown Automata

Pushdown automata (pda) are one of the classical models of computation presented in each theoretical computer science textbook. The main reason for this is that nondeterministic pushdown automata (npda) define the well-known class of context-free languages (CF) and that deterministic pushdown automata (dpda) define the class of deterministic context-free languages (DCF). Despite of these facts randomized versions of pushdown automata are barely investigated and so there are only a few papers [1,8,14] on randomized pushdown automata. This is in contrast to an intensive search for natural extensions of the classes DCF and CF motivated by compiler construction. But, as pointed out in [8], randomized pushdown automata with amplification provide a natural extension of dpda's and hence of deterministic context-free languages.

Definition 1. *We define a randomized pda P as a nondeterministic pda with a probability distribution over the next moves and demand that all computations are finite. We say that P recognizes a language L with error at most $\varepsilon(n)$, iff for each $x \in L$, $\text{Prob}(P \text{ accepts } x) \geq 1 - \varepsilon(|x|)$ and for each $x \notin L$, $\text{Prob}(P \text{ rejects } x) \geq 1 - \varepsilon(|x|)$.*

In [8] various modes of randomized pda are separated from deterministic and nondeterministic pda. For instance, it is shown that Las Vegas pda are more powerful than dpa (i.e., the class of languages recognized by Las Vegas pushdown automata is a natural extension of DCF), and randomized pda with arbitrarily small error probability can be more powerful than npda (i.e., randomized pda's with arbitrarily small error recognize non context-free languages). One of the main remaining open problems was to determine whether there is a context-free language that cannot be accepted by a bounded-error pda. We show that nondeterminism can be even stronger than probabilism with weakly-unbounded error by considering the context-free language $IP = \{ u \circ v^{\text{reverse}} \in \{0,1\}^* \mid |u| = |v| \text{ and } \sum_{i=1}^{|u|} u_i \cdot v_i \equiv 1 \pmod{2} \}$.

Theorem 1. *IP cannot be recognized by a randomized pda with error at most $\frac{1}{2} - c \cdot \frac{\log_2 n}{n}$, where n is the length of the input and c is a (suitably large) constant.*

A second open problem concerns the question of amplification: are randomized two-sided error pda capable of reducing the error probability? It is easy to observe that the union of any two deterministic context-free languages can always be accepted with error probability $\frac{1}{3}$: If $L = L(A_1) \cup L(A_2)$ for dfa's A_1, A_2 , then a randomized pda A decides to simulate A_1 (resp. A_2) by tossing a fair coin. If the input w is accepted by the corresponding dfa, then w is accepted with probability 1 and otherwise accepted with probability $\frac{1}{3}$. Thus the acceptance

probability for $w \in L$ is at least $\frac{1}{2} \cdot (1 + \frac{1}{3}) = \frac{2}{3}$ and for $w \notin L$ is at most $\frac{1}{2} \cdot (\frac{1}{3} + \frac{1}{3}) = \frac{1}{3}$. Observe that the language

$$\text{IP}^2 = \{ u\#x\#v\#y \mid (|u| = |v| \text{ and } u \circ v \in \text{IP}) \text{ or } (|x| = |y| \text{ and } x \circ y \in \text{IP}) \}$$

is a union of two deterministic context-free languages. We show that $\frac{1}{3}$ is a sharp threshold and hence randomized pushdown automata cannot be amplified.

Theorem 2. IP^2 cannot be recognized by a randomized pda with error at most $\frac{1}{3} - 2^{-n/8+c \cdot \log_2 n}$, where n is the length of the input and c is a (suitably large) constant.

We apply methods from communication complexity, but face a severe problem, since a traditional simulation of pda by communication cannot handle the large amount of information stored in the stack. Hence we have to design new communication models that are powerful enough to be applicable to pda, but also weak enough so that their power can be controlled. The resulting method for proving lower bounds on randomized pda is the main contribution of this paper.

1.2 Multicounter Machines

Here we consider the model of two-way multicounter machines with a constant number of reversals and polynomial running time. (A reversal is a reversal of the reading head on the input tape). Note that polynomial-time two-way deterministic (nondeterministic) multicounter machines define exactly DLOGSPACE (NLOGSPACE). But it is an open problem whether polynomial-time two-way randomized multicounter machines determine the corresponding randomized logarithmic space class, because $\text{LVLOGSPACE} = \text{NLOGSPACE}$ and the simulation of nondeterminism by Las Vegas randomization causes an exponential increase of time complexity [11,20,10,16]. Let **1DMC(poly)** [**1NMC(poly)**] be the class of languages accepted by polynomial-time one-way deterministic [nondeterministic] multicounter machines. Let **2cDMC(poly)** [**2cNMC(poly)**] denote the class of languages accepted by deterministic [nondeterministic] two-way mcm with a constant number of reversals. (*mcm* denotes a multicounter machine).

Definition 2. Let A be a randomized mcm with three final states q_{accept} , q_{reject} and q_{neutral} . We say that A is a **Las Vegas mcm (LVmcm)** recognizing a language L if for each $x \in L$, $\text{Prob}(A \text{ accepts } x) \geq \frac{1}{2}$ and $\text{Prob}(A \text{ rejects } x) = 0$ and for each $x \notin L$, $\text{Prob}(A \text{ rejects } x) \geq \frac{1}{2}$ and $\text{Prob}(A \text{ accepts } x) = 0$.

We say that A is a **one-sided-error Monte Carlo mcm, Rmcm** for L iff for each $x \in L$, $\text{Prob}(A \text{ accepts } x) \geq \frac{1}{2}$, and for each $x \notin L$, $\text{Prob}(A \text{ rejects } x) = 1$.

We say that A is a **bounded-error probabilistic mcm, BPmcm** for L , if there is a constant $\varepsilon > 0$ such that for each $x \in L$, $\text{Prob}(A \text{ accepts } x) \geq \frac{1}{2} + \varepsilon$ and for each $x \notin L$, $\text{Prob}(A \text{ rejects } x) \geq \frac{1}{2} + \varepsilon$.

We denote by **1LVMC(poly)**[**1RMC(poly)**, **1BPMC(poly)**] the class of languages accepted by a polynomial-time one-way LVmcm [Rmcm, BPmcm]. Let

$2cLVMC(poly)$ [$2cRMC(poly)$, $2cBPMC(poly)$] denote the class of languages accepted by polynomial-time two-way LVmcm [$Rmcm$, $BPmcm$] with a constant number of reversals.

All probabilistic classes possess amplification: We can reduce the error arbitrarily by simulating independent runs with an appropriately increased number of counters. Here the interesting question is whether an error probability tending to zero is reachable and we therefore consider the complexity class $\mathcal{C}^*$ of all languages from $\mathcal{C}$ recognizable with error probability tending towards 0 with machines of the same type as in $\mathcal{C}$. (In the case of Las Vegas randomization we consider the probability of giving the answer "?" as error probability.) We obtain the following separations.

Theorem 3. (a) *Bounded-error randomization and nondeterminism are incomparable, since $1NMC(poly) - 2cBPMC(poly) \neq \emptyset$ and $1BPMC^*(poly) - 2cNMC(poly) \neq \emptyset$. Thus, in particular, $1BPMC^*(poly) - 2cRMC(poly) \neq \emptyset$.*
 (b) *One-sided-error randomization is more powerful than Las Vegas randomization, since $1RMCM^*(poly) - 2cLVMC(poly) \neq \emptyset$.*
 (c) *Las Vegas is more powerful than determinism, since $2cDMC(poly)$ is a proper subset of $2cLVMC^*(poly)$ and $2cLVMC^*(2^{O(\sqrt{n} \cdot \log^2 n)}) - 2cDMC(2^{o(n)}) \neq \emptyset$.*

Theorem 3 shows a proper hierarchy between LVmcc, Rmcc and BPmcc resp. nondeterministic mcc, where the weaker mode cannot reach the stronger mode, even when restricting the stronger mode to 1-way computations and additionally demanding error probability approaching 0. The proof even shows that allowing $o(n/\log n)$ reversals on inputs of size n does not help the weaker mode.

It is not unlikely that determinism and Las Vegas randomization are equivalent for 1-way computations. However the separation $2cLVMC^*(2^{O(\sqrt{n} \cdot \log^2 n)}) - 2cDMC(2^{o(n)}) \neq \emptyset$ also holds for $o(n/\log n)$ reversals of the deterministic machine.

The paper is organized as follows. Theorems 1 and 2 are shown in section 2, where we also describe the non-standard two-trial communication model. Section 3 is devoted to the study of randomized multicounter machines.

2 Pushdown Automata

In this section we outline the proof idea of Theorems 1 and 2. Since we demand that all computations of a randomized pda are finite, we obtain:

Fact 1 *Every computation of a randomized pda on an input w runs in time $O(|w|)$.*

The class of languages recognizable by randomized pda with bounded error seems to have lost any resemblance of the pumping-property, since for instance the language $\{a^n \circ b^n \circ c^n \mid n \in \mathbb{N}\}$ is recognizable with even arbitrarily small error [8]. Thus structural reasons as limits on the computing power seem unlikely.

Therefore we try to apply methods from communication complexity, but are immediately confronted with the problem of dealing with a potentially large stack which may encode the entire input seen so far. Hence we develop the two-trial communication model, a non-standard model of communication which is tailor-made to handle pda.

2.1 Two-Trial Communication

Definition 3. Let P be a randomized pda and let C be a deterministic computation of P on input w . We define $\text{stack}_C(w)$ to equal the contents of the stack after reading w according to C and just before reading the next input letter. $\text{height}_C(w)$ denotes the height of $\text{stack}_C(w)$.

We say that C compresses u_2 relative to the partition (u_1, u_2, v_1) iff the lowest stack height h when reading u_2 is at least as large as the lowest stack height when reading v_1 . We demand that $h \leq \text{stack}_C(u_1)$ and $h \leq \text{stack}_C(u_1 \circ u_2)$.

We first introduce the two-trial communication model informally by describing a simulation of a randomized pda P on an input w . Two processors A and B participate. The input w is arbitrarily partitioned into four substrings $w = u_1 \circ u_2 \circ v_1 \circ v_2$ of P and accordingly A (resp. B) receives the pair (u_1, u_2) (resp. (v_1, v_2)). When reading v_1 , the deterministic computation C has the option to compress u_2 . Therefore we simulate P by a randomized two-round protocol which utilizes two *trials*. The protocol assumes public random bits and will determine whether w is to be accepted.

In **trial 1** the simulation will be successful, if C does not compress u_2 relative to the partition (u_1, u_2, v_1) . In particular, let h be the height of the lowest stack when reading u_2 and let T_1 be the last time¹ when the stack has height h . A sends

1. a pointer to the first unused random bit at time T_1 ,
2. the state and the topmost stack symbol at time T_1 ,
3. u_2 and a pointer to the first unread input symbols of u_2 at time T_1 .

Processor B will be able to simulate P , beginning at time T_1 , as long as the stack height is at least as large as h . If the stack height decreases to $h - 1$ when reading v_1 , then B stops the trial by sending a question mark. Otherwise B commits and we observe that B 's commitment decision does not depend on v_2 . If the stack height reaches height $h - 1$ at time T_2 , then B sends

4. a pointer to the first unused random bit at time T_2 ,
5. the current state at time T_2 ,
6. v_2 and a pointer to the first unread input symbol of v_2 at time T_2

and processor A can finish the simulation. Thus A sends u_2 , followed by B who sends v_2 . Moreover both processors exchange $O(\log_2(|w|))$ additional bits. The

¹ A configuration at time T is the configuration before executing the operation at time $T + 1$.

simulation is successful, provided P does not compress u_2 relative to (u_1, u_2, v_1) . Also remember that B can determine whether this trial is successful without consulting v_2 .

But trial 1 may fail, if C does compress u_2 relative to the partition (u_1, u_2, v_1) . Therefore **trial 2** assumes compression. Processor B begins by sending v_1 and A replies with a question mark if u_2 is not compressed. Otherwise A commits and continues the simulation which results in compressing u_2 . Assume that h is the height of the lowest stack when reading v_1 and that height h is reached at time T for the last time. Observe that $h \leq \text{height}_C(u_1)$, since u_2 is compressed. A sends

1. a pointer to the first unused random bit at time T ,
2. the state at time T and the height h ,
3. u_1 and a pointer to the first unread input symbols of v_1 at time T .

B first determines $\text{stack}_C(u_1)$ by simulating C on u_1 and then determines the stack at time T , which consists of the h bottommost stack elements of $\text{stack}_C(u_1)$. Then B finishes the computation by simulating C from time T onwards with the help of the remaining information. Observe that B sends v_1 , followed by A who sends u_1 and $O(\log_2(|w|))$ additional bits. The simulation is successful, provided C compresses u_2 relative to (u_1, u_2, v_1) .

Moreover A 's decision to commit can be based only on the lowest stack height h' when reading u_2 , the top portion of the stack after reading $u_1 \circ u_2$ (i.e., the stack elements with height larger than h'), the state after reading $u_1 \circ u_2$ and the string v_1 . To determine the top portion of the stack, A just has to know the state and stack element after visiting height h' for the last time t , the first unread position of u_2 and the first unused random bit at time t and u_2 . Thus knowledge of u_2, v_1 and additional information on u_1 and u_2 of logarithmic length is sufficient.

The following definition formalizes the two-trial communication model.

Definition 4. Let $c : \mathbb{N} \rightarrow \mathbb{N}$ be a given function. A two-trial randomized communication protocol $\mathcal{P}$ with communication at most $c(n)$ is defined as follows.

- (a) Processor A receives (u_1, u_2) and processor B receives (v_1, v_2) as input. We set $u = u_1 \circ u_2$, $v = v_1 \circ v_2$ and $w = u \circ v$. We assume public random bits throughout.
- (b) In trial 1 A sends u_2 and an additional message of length at most $c(|w|)$. Either B sends a question mark or B commits and replies by sending v_2 and an additional message of length at most $c(|w|)$. B 's decision to commit does not depend on v_2 .
- (c) In trial 2 B sends v_1 . Either A sends a question mark or A commits and replies by sending u_1 and an additional message of length at most $c(|w|)$. A 's commitment decision is based only on u_2, v_1 and a string s_{u_1, u_2} . The string s_{u_1, u_2} has length $O(\log_2(|u|))$ and depends only on u_1 and u_2 .
- (d) For every deterministic computation of $\mathcal{P}$ on input w exactly one of the two trial commits and one processor has to determine the output.

We summarize the main properties of the two-trial communication model. We consider exchanging u_2, v_2 in trial 1, resp. exchanging u_1, v_1 in trial 2 as free and charge only for the additional information. The decision to commit has become a powerful new feature of the new model and therefore it is demanded that commitment can be determined with restricted input access.

In the next definition we define acceptance of languages. We require the error probability for every input w and for every partition of w to be small. A question mark is not counted as an error, but property (d) demands that for every deterministic computation exactly one trial leads to commitment.

Definition 5. Let $L \subseteq \Sigma^*$ be a language and let $\mathcal{P}$ be a two-trial randomized communication protocol. For an input w and a partition $p = (u_1, u_2, v_1, v_2)$ with $w = u_1 \circ u_2 \circ v_1 \circ v_2$ we define the error probability of w relative to p to be

$$\varepsilon_p(w) = t_p^1(w) \cdot \varepsilon_p^1(w) + t_p^2(w) \cdot \varepsilon_p^2(w),$$

where $\varepsilon_p^i(w)$ is the error probability for w in trial i and $t_p^i(w)$ is the probability that the processors commit in trial i on input w relative to partition p . (Hence an error is a misclassification and a question mark is disregarded.)

We say that $\mathcal{P}$ recognizes L with error probability at most ε iff $\varepsilon_p(w^*) \leq \varepsilon$ for every input w and for every partition p of w .

We summarize our above simulation of a randomized pda.

Lemma 1. Let P be a randomized pda. Assume that P recognizes the language L with error probability at most ε . Then L can be recognized in the two-trial model with communication $O(\log_2 n)$ and error probability at most ε .

This simulation works also for pda's and dpda's. However the resulting lower bounds will not always be best possible. For instance $\{a^n \circ b^n \circ c^n \mid n \geq 0\}$ can be recognized in the deterministic two-trial model with communication $O(\log_2 n)$, since A can encode its entire input with logarithmically many bits. As a second example consider the language

$$\text{ND} = \{ u \# v \in \{0, 1\}^* \mid \text{there is } i \text{ with } u_i = v_i = 1 \}$$

of non-disjointness. ND can probably not be recognized with bounded-error by a randomized pushdown automata, however the following two-trial protocol recognizes ND with error at most $\frac{1}{3}$ without any (charged) communication: the processors commit with probability $\frac{1}{2}$. If a common element is determined after exchanging u_1, v_1 (resp. u_2, v_2), then accept with probability 1 and otherwise accept with probability $\frac{1}{3}$. Hence the error is $\frac{1}{3}$ for disjoint sets and otherwise the error is at most $\frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$.

Thus a separation of probabilism and nondeterminism remains non-trivial, since ND is the prime example for separating probabilism and nondeterminism within conventional two-party communication [12,17].

2.2 Discrepancy

Let X and Y be finite sets and let $L \subseteq X \times Y$ be a language. We say that R is a rectangle, if $R = X' \times Y'$ for subsets $X' \subseteq X$ and $Y' \subseteq Y$. The discrepancy $D_\mu(R, L)$ of L with respect to a rectangle R and a distribution μ is defined as

$$D_\mu(R, L) = \left| \sum_{(x,y) \in R \text{ and } (x,y) \notin L} \mu(x, y) - \sum_{(x,y) \in R \text{ and } (x,y) \in L} \mu(x, y) \right|.$$

$D_\mu(L) = \max_R D_\mu(R, L)$ is the discrepancy of L with respect to μ . Languages with small discrepancy force conventional randomized protocols to exchange correspondingly many bits, since large rectangles introduce too many errors.

Fact 2 (a) *Let P be a conventional deterministic protocol for L with expected error $\frac{1}{2} - \varepsilon$ w.r.t distribution μ . Then P has to exchange at least $\log_2(\frac{2 \cdot \varepsilon}{D_\mu(L)})$ bits.*

(b) *Set $\text{IP}_n = \{u \circ v \in \text{IP} : |u| = |v| = n\}$ and $X = Y = \{0, 1\}^n$. Then $D_{\text{uniform}}(R, \text{IP}_n) \leq 2^{-n/2}$ for every rectangle R and the uniform distribution.*

Part (a) is Proposition 3.28 in [13]. Part (b) is shown in example 3.29 of [13].

2.3 Proof of Theorem 2

We now show that our non-standard communication model allows us to sharply bound the error probability when recognizing IP^2 . We restrict our attention to $\text{IP}_N^2 = \{u_1 \# u_2 \# v_1 \# v_2 \in \text{IP}^2 \mid |u_1| = |v_1| = |u_2| = |v_2| = N\}$. Since the input size equals $4 \cdot N$, it suffices to show that IP_N^2 cannot be recognized for sufficiently large N in the two-trial model with communication $O(\log_2 N)$ and error probability at most $\varepsilon = \frac{1}{3} - 2^{-N/2 + c \cdot \log_2 N}$. Assume otherwise and let $\mathcal{P}$ be a randomized two-trial protocol with error less than ε and communication $O(\log_2 N)$. We define the distribution μ , where μ is the uniform distribution on all inputs (u_1, u_2, v_1, v_2) with $|u_1| = |u_2| = |v_1| = |v_2| = N$ and $u_1 \circ v_1^{\text{reverse}} \notin \text{IP}$ or $u_2 \circ v_2^{\text{reverse}} \notin \text{IP}$. By enumerating all coin tosses we find a deterministic protocol $\mathcal{P}^*$ with communication $O(\log_2 N)$ such that the expected error of $\mathcal{P}^*$ is less than ε for distribution μ .

We begin by investigating a committing *trial 2* message R of $\mathcal{P}^*$, since exploiting the feature of commitment is harder for trial 2 messages. R consists of all inputs for which identical additional information is sent from processor A to processor B ; additionally we require that processor B either accepts or rejects all inputs of R . Observe that R will in general not have the rectangle property, since A 's message also depends on v_1 . However, if we fix u_1 and v_1 , then $R(u_1, v_1) = \{(u_1, u_2, v_1, v_2) \in R \mid u_2, v_2 \in \{0, 1\}^N\}$ is a rectangle and thus R is the disjoint union of the rectangles $R(u_1, v_1)$.

We call an input (u, v) *dangerous*, if $u_1 \circ v_1^{\text{reverse}} \notin \text{IP}$ and *harmless* otherwise. Observe that a harmless input belongs to IP_N^2 . We define $D^+(R)$ (resp. $D^-(R)$) as the set of dangerous inputs of R belonging to IP_N^2 (resp. to the complement)

and $H(R)$ as the set of harmless inputs. Our first goal is to show that messages cannot differentiate between dangerous positive and dangerous negative inputs.

Claim 1 *For any message R , $|\mu(D^+(R)) - \mu(D^-(R))| \leq 2^{-N/2}$.*

Proof. We fix u_1 and v_1 with $u_1 \circ v_1^{\text{reverse}} \notin \text{IP}$ and observe that $(u_1, u_2, v_1, v_2) \in R$ belongs to IP_N^2 iff $u_2 \circ v_2^{\text{reverse}}$ belongs to IP_N . Therefore we obtain with Fact 2 (b) that

$$D_{\text{uniform}}(R(u_1, v_1), \text{IP}_N) \leq 2^{-N/2}. \quad (1)$$

The claim follows by summing inequality (1) over all pairs (u_1, v_1) with $u_1 \circ v_1^{\text{reverse}} \notin \text{IP}$ and afterwards rescaling to the measure μ . $\square$

Let $\mathcal{R}$ be the set of inputs for which a trial 2 message commits. Our second goal is to show that the μ -weights of $D^+(\mathcal{R})$, $D^-(\mathcal{R})$ and $H(\mathcal{R})$ are almost identical.

Claim 2 $|\frac{1}{3} \cdot \mu(\mathcal{R}) - \mu(H(\mathcal{R}))| \leq \text{poly}(N) \cdot 2^{-N/2}$.

Proof. According to Definition 4, processor A decides its commitment based on its knowledge of the string s_{u_1, u_2} , u_2 and v_1 , where the string s_{u_1, u_2} is of length $O(\log_2(|u_1| + |u_2|))$ and only depends on u_1 and u_2 . Thus we can view A 's commitment as the result of a message from a processor A' with input (u_1, u_2) to a processor B' with input (u_2, v_1) . We fix u_2 , apply Fact 2 (b) to this “commitment” message and obtain a discrepancy (of IP_N relative to the uniform distribution) of at most $2^{-N/2}$. Thus a commitment message cannot differentiate between $u_1 \circ v_1^{\text{reverse}} \notin \text{IP}$ and $u_1 \circ v_1^{\text{reverse}} \in \text{IP}$. Since there are polynomially many commitment messages, the overall discrepancy for fixed u_2 is at most $\text{poly}(N) \cdot 2^{-N/2}$. Hence, after considering all possible values of u_2 ,

$$\frac{1}{2^{4N}} \cdot ||D^+(\mathcal{R})| + |D^-(\mathcal{R})| - |H(\mathcal{R})|| \leq \text{poly}(N) \cdot 2^{-N/2} \quad (2)$$

follows. For a message R let $H^+(R)$ (resp. $H^-(R)$) be the set of harmless inputs of R with $u_2 \circ v_2^{\text{reverse}} \in \text{IP}$ (resp. with $u_2 \circ v_2^{\text{reverse}} \notin \text{IP}$). Then $||H^+(R)| - |H^-(R)|| \leq 2^{4N} \cdot 2^{-N/2}$, since the discrepancy of IP_N with respect to $R(u_1, v_1)$ is upper-bounded by $2^{-N/2}$ for every pair (u_1, v_1) with $u_1 \circ v_1^{\text{reverse}} \in \text{IP}$. Since we have only polynomially many messages, we obtain

$$\frac{1}{2^{4N}} \cdot ||H^+(\mathcal{R})| - |H^-(\mathcal{R})|| \leq \text{poly}(N) \cdot 2^{-N/2}.$$

The result follows from (2) and Claim 1, since $\mu(H(\mathcal{R})) = \frac{4}{3} \cdot \frac{1}{2^{4N}} \cdot |H^-(\mathcal{R})|$. $\square$

Let $(A_i \mid i \leq \text{poly}(N))$ (resp. $(R_i \mid i \leq \text{poly}(N))$) be the sequence of all accepting (resp. rejecting) messages of $\mathcal{P}^*$. Therefore Claim 1 and Claim 2 imply

$$\begin{aligned} D &:= \sum_i |\mu(D^+(R_i)) - \mu(D^-(R_i))| + \sum_i |\mu(D^+(A_i)) + \mu(H(A_i)) - \mu(D^-(A_i))| \\ &\leq \text{poly}(N) \cdot 2^{-N/2} + \frac{\mu(\mathcal{R})}{3}. \end{aligned}$$

Since harmless inputs belong to IP_N^2 , we may assume w.l.o.g. that $H(R_i) = \emptyset$ for all i . Thus D adds up the measure of the symmetric difference between the sets of correctly and incorrectly classified inputs over all messages of $\mathcal{P}^*$. Hence D is at least as large as the measure of the symmetric difference between the sets of inputs, which are correctly, respectively incorrectly classified by $\mathcal{P}^*$. Thus, if ε_2 is the expected error of trial-2 messages, then $\mu(\mathcal{R}) \cdot (1 - \varepsilon_2 - \varepsilon_2) \leq D$. We obtain:

Claim 3 *If $\mathcal{R}$ is the set of inputs for which trial-2 messages commit, then*

$$\mu(\mathcal{R}) \cdot (1 - 2 \cdot \varepsilon_2) \leq \text{poly}(N) \cdot 2^{-N/2} + \frac{\mu(\mathcal{R})}{3}.$$

The corresponding claim for trial-1 messages can be shown analogously. Thus, since $\mathcal{P}^*$ commits itself for each input in exactly one trial due to Definition 4 (d), we get $(1 - \mu(\mathcal{R})) \cdot (1 - 2 \cdot \varepsilon_1) \leq \text{poly}(N) \cdot 2^{-N/2} + \frac{1 - \mu(\mathcal{R})}{3}$, where ε_1 is the expected error of trial-1 messages.

Let ε be the expected error probability of $\mathcal{P}^*$. Then $\varepsilon = \varepsilon_1 \cdot (1 - \mu(\mathcal{R})) + \varepsilon_2 \cdot \mu(\mathcal{R})$ and we obtain $1 - 2 \cdot \varepsilon \leq \text{poly}(N) \cdot 2^{-N/2} + \frac{1}{3}$ after adding the inequalities for ε_1 and ε_2 : the claim $\varepsilon \geq \frac{1}{3} - \text{poly}(N) \cdot 2^{-N/2}$ follows. $\square$

2.4 Proof of Theorem 1

The argument for Theorem 1 needs a further ingredient besides two-trial communication. Let P be a randomized pda for IP. We set

$$f_P(v_1) = \sum_{u_1 \circ u_2 \in \Sigma^{2N}} \text{prob}[P \text{ compresses } u_2 \text{ for 1 partition } (u_1, u_2, v_1)]$$

and show that a string v_1 can be constructed such that the probability of compression w.r.t. (u_1, u_2, v_1) is, on the average, almost as high as the probability of compression w.r.t. $(u_1, u_2, v_1 \circ v_2)$ for strings $v_2 \in \Sigma^{2N}$. (Observe that the probability of compression does not decrease when appending suffices.)

We make v_1 known to both processors in a simulating two-trial protocol. If processor A receives (u_1, u_2, v_1) , then A can determine whether trial 1 fails. If it does, then A , already knowing v_1 , sends u_1 and a small amount of information enabling B to continue the simulation. If trial 1 succeeds, then A sends u_2 and again additional information for B to continue. But this time B will, with high probability, not have to respond, since trial 1 will remain successful with high probability for suffix $v_1 \circ v_2$. Thus the two-trial communication model “almost” turns one-way and the issue of commitment disappears.

We begin with the construction of $v = v_1$. For a string $x \in \Sigma^{2N}$ let x_1 be the prefix of the first N letters and let x_2 be the suffix of the last N letters of x .

Proposition 1. *Let $\Delta \in \mathbb{N}$ be given. Then there is a string $v \in \Sigma^*$ of length at most $2N \cdot \frac{|\Sigma|^{2N}}{\Delta}$ such that $f_P(v \circ w) \leq \Delta + f_P(v)$ for all $w \in \Sigma^{2N}$.*

Proof. We obtain $f_P(v) \leq f_P(v \circ w)$, since the probability of compression does not decrease when appending suffices. We now construct a string v incrementally as follows:

- (1) Set $i = 0$ and $v^0 = \lambda$, where λ is the empty string.
- (2) If there is a string $v' \in \Sigma^{2N}$ with $f_P(v^i \circ v') - f_P(v^i) \geq \Delta$, then set $v^{i+1} = v^i \circ v'$, $i = i + 1$ and go to (2). Otherwise stop and output $v = v^i$.

Observe that there are at most $|\Sigma|^{2N}/\Delta$ iterations, since the “ f -score” increases by at least Δ in each iteration and since the maximal f -score is $|\Sigma|^{2N}$. $\square$

We fix Δ and N and obtain a string v with the properties stated in Proposition 1. Finally define

$$L_{N,v} = \{ (u, w) \mid |u| = |w| = 2N \text{ and } u \circ v \circ w \in L \}.$$

We now utilize that the two-trial protocol of Lemma 1 collapses to a conventional one-way randomized protocol with public randomness and small expected error.

Lemma 2. *Fix the parameters $N, \Delta \in \mathbb{N}$. If L is recognized by a randomized pda P with error probability at most ε , then $L_{N,v}$ can be recognized by a conventional one-way randomized communication protocol in the following sense:*

- (1) String u is assigned to processor A and string w is assigned to processor B . Both processors know v .
- (2) The communication protocol achieves error probability at most $\varepsilon + p_{u,w}$ on input (u, w) , where

$$\sum_{u \in \Sigma^{2N}} \sum_{w \in \Sigma^{2N}} p_{u,w} \leq \Delta \cdot |\Sigma|^{2N}.$$

- (3) Processor A sends a message of $O(\log_2(|u| + |v|))$ bits and additionally either u_1 or u_2 is sent. u_1 (resp. u_2) is the prefix (resp. suffix) of u of length N .

Proof. Let u be the input of processor A and w the input of processor B . Let $p_{u,w}$ be the probability that P compresses u_2 relative to $(u_1, u_2, v \circ w)$, but not relative to (u_1, u_2, v) . By assumption on v we have

$$\sum_{u \in \Sigma^{2N}} p_{u,w} \leq \Delta$$

for each $w \in \Sigma^{2N}$. We now simulate P on $u \circ v \circ w$ along the lines of Lemma 1, however this time we only use conventional one-way communication.

Processor A simulates a computation C of P on input $u \circ v$. If the computation C does not compress u_2 relative to (u_1, u_2, v) , then A behaves exactly as in trial 1 and sends u_2 and $O(\log_2(|u| + |v|))$ additional bits. Now processor B will be able to reconstruct the relevant top portion of the stack obtained by P after reading $u \circ v$ and to continue the simulation as long as top portion is not emptied. If the the top portion is emptied, then B accepts all inputs from this point on. (Observe that this happens with probability at most $p_{u,w}$.)

If the computation C compresses u_2 relative to (u_1, u_2, v) , then processor A behaves exactly as in trial 2 and sends u_1 and $O(\log_2(|u| + |v|))$ additional bits.

Now processor B can finish the simulation without introducing an additional error. All in all the additional error is bounded by

$$\sum_{u \in \Sigma^{2N}} \sum_{w \in \Sigma^{2N}} p_{u,w} \leq \Delta \cdot |\Sigma|^{2N}$$

and this was to be shown. $\square$

We are now ready to show that IP, the language of inner products, has no randomized pda, even if we allow a weakly unbounded error computation. We set $\text{IP}_N = \{ u \circ v^{\text{reverse}} \in \text{IP} \mid |u| = |v| = N \}$ and observe that either $\text{IP}_{N,v}$ equals IP_{2N} or it equals the complement of IP_{2N} . Hence, if we assume that IP can be recognized by a randomized pushdown P with error probability δ , then we obtain a one-way randomized communication protocol that “almost” recognizes IP_{2N} with error probability “close” to δ .

We set $\varepsilon = \frac{1}{2} - \delta$ and $\Delta = \frac{\varepsilon}{2} \cdot 2^{2N}$. The randomized protocol induced by P introduces an additional total error of at most $\Delta \cdot 2^{2N}$ and hence the total error is at most

$$\delta \cdot 2^{4N} + \Delta \cdot 2^{2N} = \left(\delta + \frac{\varepsilon}{2}\right) \cdot 2^{4N} = \left(\frac{1}{2} - \varepsilon + \frac{\varepsilon}{2}\right) \cdot 2^{4N} = \left(\frac{1}{2} - \frac{\varepsilon}{2}\right) \cdot 2^{4N}.$$

Hence, by an averaging argument, we obtain a deterministic protocol with error $\frac{1}{2} - \frac{\varepsilon}{2}$ under the uniform distribution.

Next we derive a lower bound for such protocols. Our messages consist in either sending u_1 or u_2 plus additional bits and Fact 2 (b) implies that the discrepancy of such a message under the uniform distribution is upper-bounded by 2^{-N} . Hence we obtain with Fact 2 (a) that the distributional complexity (for the uniform distribution and error $\frac{1}{2} - \frac{\varepsilon}{2}$) is at least

$$\log_2\left(\frac{2 \cdot \varepsilon/2}{2^{-N}}\right) = \log_2\left(\frac{\varepsilon}{2^{-N}}\right) = N - \log_2 \frac{1}{\varepsilon}.$$

Therefore the deterministic protocol has to exchange at least $N - \log_2 \frac{1}{\varepsilon}$ bits. We set $b = O(\log_2(N + |v|))$ as the length of the additional messages and obtain

$$\log_2(N + |v|) = \Omega\left(N - \log_2 \frac{1}{\varepsilon}\right). \quad (3)$$

Finally we have $|v| \leq 2N \cdot \frac{2^{2N}}{\Delta} = 2N \cdot \frac{2^{2N}}{\frac{\varepsilon}{2} \cdot 2^{2N}} = \frac{4N}{\varepsilon}$ and (3) translates into

$$\log_2 \frac{4N}{\varepsilon} = \Omega\left(N - \log_2 \frac{1}{\varepsilon}\right).$$

Hence we get $\frac{1}{\varepsilon} = 2^{\Omega(N)}$ and $\frac{1}{\varepsilon} = \Omega\left(\frac{|v|}{\log_2 |v|}\right)$ follows. This establishes the theorem, since the error probability will be at least $\frac{1}{2} - O\left(\frac{\log |v|}{|v|}\right)$. $\square$

3 Multicounter Machines

Our first two results compare nondeterminism and bounded-error randomness.

Lemma 3. *Let $EQ = \{0^n \# w \# w \mid w \in \{0,1\}^n, n \in \mathbb{N}\}$ be the equality problem. Then $EQ \in 1BPMC^*(poly) - 2cNMC(poly)$.*

Proof Outline. First, we show $EQ \in 1BPMC^*(poly)$. For input $0^n \# w \# y$ a randomized mcm M works as follows. Reading 0^n it saves the value n in a counter and the value n^2 in another counter. Then it randomly picks a number from $\{1, \dots, n^2 - 1\}$ by tossing $\lceil \log_2 n^2 \rceil$ coins and adds the value 2^i to the contents of an appropriate counter if the i -th random bit is 1. Afterwards M deterministically checks in time $O(n^3)$ whether the random number is a prime. If it is not a prime, M generates a new random number. Since the number of primes smaller than n^2 is at least $n^2/(2 \ln n)$, M finds a prime p with probability arbitrarily close to 1 after sufficiently many attempts.

Let $\text{Number}(w)$ be the number with binary representation w . M computes $\text{Number}(w) \bmod p$ as well as $\text{Number}(y) \bmod p$ and stores the results in two separate counters. If $\text{Number}(w) \bmod p = \text{Number}(y) \bmod p$, then M accepts and rejects otherwise. Obviously, M always accepts, if $w = y$. If n and y are different, then the error probability (i.e., the probability of acceptance) is at most $2 \ln n/n$ [see for instance [6]]. Since M works in time polynomial in n we obtain that $EQ \in 1BPMC^*(poly)$.

To show that $EQ \notin 2cNMC(poly)$ we use an argument from communication complexity theory. Assume the opposite, i.e., that there is a polynomial-time nondeterministic mcm D that accepts EQ and uses at most c reversals in any computation. Let D have k counters, and let D work in time at most n^r for any input of length n . Consider the work of D on an input $0^n \# x \# y$ with $|x| = |y| = n$. D is always in a configuration where the contents of each counter is bounded by n^r . Each such configuration can be represented by a sequence of $O(k \cdot r \cdot \log_2 n)$ bits and so the whole crossing sequence on this position can be stored by $O(c \cdot k \cdot r \cdot \log_2 n)$ bits. Thus D can be simulated by a nondeterministic communication protocol that accepts EQ within communication complexity $O(\log_2 n)$. This contradicts the fact that the nondeterministic communication complexity of EQ is in $\Omega(n)$ [6,13]. $\square$

Lemma 4. (a) $NDIS = \{x \# y \mid x, y \in \{0,1\}^n \text{ for } n \in \mathbb{N} \text{ and } \exists j : x_j = y_j = 1\}$ is the non-disjointness problem. Then $NDIS \in 1NMC(poly) - 2cBPMC(poly)$.

(b) $NEQ = \{0^n \# x \# y \mid n \in \mathbb{N}, x, y \in \{0,1\}^n, x \neq y\}$ is the language of non-equality. Then $NEQ \in 1RMC^*(poly) - 2cLVMC(poly)$.

Proof Outline. (a) One can easily observe that $NDIS$ can be accepted by a nondeterministic mcm with one counter. Similarly as in the proof of Lemma 3, we simulate a polynomial-time BPmcm for $NDIS$ by a sequence of bounded-error protocols that accept $NDIS$ within communication complexity $O(\log_2 n)$. This

contradicts the result of [12,17] that the communication complexity of $NDIS$ is in $\Omega(n)$.

(b) We obtain a Rmcm for NEQ , with error probability tending towards 0, as in the proof of Lemma 3. But membership of NEQ in $2cLVMC(\text{poly})$ implies that the Las Vegas communication complexity for NEQ is in $O(\log_2 n)$ and this contradicts the lower bound $\Omega(n)$ [15]. $\square$

Observe, that the lower bounds of Lemmas 3 and 4 even work when allowing $o(n/\log n)$ reversals instead of a constant number of reversals.

Lemma 5. $2cLVMC^*(\text{poly}) - 2cDMC(\text{poly}) \neq \emptyset$ and $2cLVMC^*(2^{O(\sqrt{n} \cdot \log^2 n)}) - 2cDMC(2^{o(n)}) \neq \emptyset$.

Proof Outline. We only show the second separation. Consider the language

$$L = \{ w_1 \# \cdots \# w_m \# \# y_1 \# \cdots \# y_m \mid \forall i, j : w_i, y_i \in \{0, 1\}^m \text{ and } \exists j : w_j = y_j \}.$$

We outline how to construct a LVmcm M that accepts L in time $2^{O(\sqrt{n} \log^2 n)}$. Let $x \in \{0, 1, \#\}^*$ be an input of size n . M can check the syntactic correctness of x in one run from the left to the right in linear time. To check membership, M creates a random prime of size at most $\log_2(m+1)^3$ as in the proof of Lemma 3. If M does not succeed, then it will stop in the state $q_{neutral}$. If it succeeds, then M computes the m residues $a_i = \text{Number}(w_i) \bmod p$ and saves the vector $(a_1, \dots, a_m)$ in a counter of size $2^{O(m \cdot \log^2 m)}$. When reading $y_1 \# y_2 \# \dots \# y_m$, M determines $b_i = \text{Number}(y_i) \bmod p$, reconstructs the binary representation of a_i in time linear in $2^{O(m \cdot \log^2 m)}$ and checks whether $a_i \neq b_i$.

If all matching residues are different, then M rejects input x . If M determines two identical residues $a_j = b_j$, then M saves y_j in a designated counter in time 2^m . M reverses the direction of the head and moves to w_j in order to check whether $w_j = y_j$. If $w_j = y_j$, then M accepts x and finishes otherwise in the state $q_{neutral}$. Since $n = |x| = m \cdot (m+1)$, M works in time $2^{O(\sqrt{n} \cdot \log n)}$. Clearly, M never errs and the probability to commit approaches 1 with increasing input length. Thus, M is a LVmcm accepting L . Finally, $L \notin 2cDMC(2^{o(n)})$ follows from the communication result of [15]. $\square$

Acknowledgement. Many thanks to Jiri Sgall for helping us to improve the presentation of the paper.

References

1. J. Kaneps, D. Geidmanis, and R. Freivalds, “Tally languages accepted by Monte Carlo pushdown automata”, *RANDOM '97, Lecture Notes in Computer Science* 1269, pp. 187–195.
2. P. Ďuriš, J. Hromkovič, and K. Inone, “A separation of determinism, Las Vegas and nondeterminism for picture recognition”, *Proc. IEEE Conference on Computational Complexity*, IEEE 2000, pp. 214–228.

3. P. Ďuriš, J. Hromkovič, J.D.P. Rolim, and G. Schnitger, “Las Vegas versus determinism for one-way communication complexity, finite automata and polynomial-time computations”, *Proc. STACS’97, Lecture Notes in Computer Science* 1200, Springer, 1997, pp. 117–128.
4. M. Dietzfelbinger, M. Kutylowski, and R. Reischuk, “Exact lower bounds for computing Boolean functions on CREW PRAMs”, *J. Computer System Sciences* 48, 1994, pp. 231–254.
5. R. Freivalds, “Projections of languages recognizable by probabilistic and alternating multitape automata”, *Information Processing Letters* 13 (1981), pp. 195–198.
6. J. Hromkovič, *Communication Complexity and Parallel Computing*, Springer 1997.
7. J. Hromkovič, “Communication Protocols – An Exemplary Study of the Power of Randomness”, *Handbook on Randomized Computing*, (P. Pardalos, S. Kajasekaran, J. Reif, J. Rolim, Eds.), Kluwer Publisher 2001, to appear.
8. J. Hromkovič, and G. Schnitger, “On the power of randomized pushdown automata”, *5th Int. Conf. Developments in Language Theory*, 2001, pp. 262–271.
9. J. Hromkovič, and G. Schnitger, “On the power of Las Vegas for one-way communication complexity, OBDD’s and finite automata”, *Information and Computation*, 169, 2001, pp.284–296.
10. J. Hromkovič, and G. Schnitger, “On the power of Las Vegas II, Two-way finite automata”, *Theoretical Computer Science*, 262, 2001, pp. 1–24
11. Immermann, N, “Nondeterministic space is closed under complementation”, *SIAM J. Computing*, 17 (1988), pp. 935–938.
12. B. Kalyanasundaram, and G. Schnitger, “The Probabilistic Communication Complexity of Set Intersection”, *SIAM J. on Discrete Math.* 5 (4), pp. 545–557, 1992.
13. E. Kushilevitz, and N. Nisan, *Communication Complexity*, Cambridge University Press 1997.
14. I. Macarie, and M. Ogihara, “Properties of probabilistic pushdown automata”, *Technical Report TR-554*, Dept. of Computer Science, University of Rochester 1994.
15. K. Mehlhorn, and E. Schmidt, “Las Vegas is better than determinism in VLSI and distributed computing”, *Proc. 14th ACM STOC’82*, ACM 1982, pp. 330–337.
16. I.I. Macarie, and J.I. Seiferas, “Amplification of slight probabilistic advantage at absolutely no cost in space”, *Information Processing Letters* 72, 1999, pp. 113–118.
17. A.A. Razborov, “On the distributional complexity of disjointness”, *Theor. Comp. Sci.* 106 (2), pp. 385–390, 1992.
18. M. Sauerhoff, “On nondeterminism versus randomness for read-once branching programs”, *Electronic Colloquium on Computational Complexity*, TR 97 - 030, 1997.
19. M. Sauerhoff, “On the size of randomized OBDDs and read-once branching programs for k -stable functions”, *Proc. STACS ‘99, Lecture Notes in Computer Science* 1563, Springer 1999, pp. 488–499.
20. R. Szelepcsényi, “The method of forcing for nondeterministic automata”, *Ball. EATCS* 33, (1987), pp. 96–100.

Generalized Framework for Selectors with Applications in Optimal Group Testing

Annalisa De Bonis¹, Leszek Gąsieniec², and Ugo Vaccaro¹

¹ Dipartimento di Informatica ed Applicazioni, Università di Salerno, 84081 Baronissi (SA), Italy

² Department of Computer Science, The University of Liverpool, Liverpool, L69 7ZF, UK

Abstract. Group Testing refers to the situation in which one is given a set of objects $\mathcal{O}$, an unknown subset $\mathcal{P} \subseteq \mathcal{O}$, and the task is to determine $\mathcal{P}$ by asking queries of the type “does $\mathcal{P}$ intersect $\mathcal{Q}$?”, where $\mathcal{Q}$ is a subset of $\mathcal{O}$. Group testing is a basic search paradigm that occurs in a variety of situations such as quality control in product testing, searching in storage systems, multiple access communications, and software testing, among the others. Group testing procedures have been recently applied in Computational Molecular Biology, where they are used for screening library of clones with hybridization probes and sequencing by hybridization.

Motivated by particular features of group testing algorithms used in biological screening, we study the efficiency of two-stage group testing procedures. Our main result is the first optimal two-stage algorithm that uses a number of tests of the same order as the information theoretic lower bound on the problem. We also provide efficient algorithms for the case in which there is a Bernoulli probability distribution on the possible sets $\mathcal{P}$, and an optimal algorithm for the case in which the outcome of tests may be unreliable because of the presence of “inhibitory” items in $\mathcal{O}$. Our results depend on a combinatorial structure introduced in this paper. We believe that it will prove useful in other contexts too.

1 Introduction and Contributions

In group testing, the task is to determine the *positive* members of a set of objects $\mathcal{O}$ by asking subset queries of the form “does the subset $\mathcal{Q} \subseteq \mathcal{O}$ contain a positive object?”. Each query informs the tester whether or not the subset $\mathcal{Q}$ (in common parlance called a *pool*) has a nonempty intersection with the subset of positive members denoted by $\mathcal{P}$. A negative answer to this question gives information that all the items belonging to pool $\mathcal{Q}$ are negative, i.e., non-positive. The aim of group testing is to identify the unknown subset $\mathcal{P}$ using as few queries as possible.

Group testing was originally introduced as a potential approach to economical mass blood testing [22]. However, due to its basic nature, it has been proved to find application in a surprising variety of situations, including quality control

in product testing [44], searching files in storage systems [32], sequential screening of experimental variables [36], efficient contention resolution algorithms for multiple-access communication [32,46], data compression [28], and software testing [9,15]. Group testing has also exhibited strong relationships with several disciplines like Coding Theory, Information Theory, Complexity, Computational Geometry, Computational Learning Theory, among others.

Probably the most important modern applications of group testing are in the realm of Computational Molecular Biology, where it is used for screening library of clones with hybridization probes [4,10,8], and sequencing by hybridization [40, 42]. We refer to [5,23,26,29] for an account of the fervent development of the area. The applications of group testing to biological screening present some distinctive features that pose new and challenging research problems. For instance, in the biological setting screening one pool at the time is far more expensive than screening many pools in parallel. This strongly encourages the use of nonadaptive procedures for screening, that is, procedures in which all tests must be specified in advance without knowing the outcomes of other tests. Instead, in adaptive group testing algorithms the tests are performed one by one, and the outcomes of previous tests are assumed known at the time of determining the current test. Unfortunately, it is known that non-adaptive group testing strategies are inherently much more costly than adaptive algorithms. This can be shown by observing that non-adaptive group testing algorithms are essentially equivalent to *superimposed codes* [24,25,32] (equivalently, cover free families) and by using known non-existential results on the latter [27,24,43]. A nearly non-adaptive algorithm that is of considerable interest for screening problems is the so called *trivial two-stage algorithm* [33]. Such an algorithm proceeds in two stages: In the first stage certain pools are tested in parallel; in the second stage individual objects may be tested singly, depending on the outcomes of the first stage. Our first result is rather surprising: we prove that the best trivial two-stage algorithms are asymptotically as much efficient as the best *fully adaptive* group testing algorithms, that is, algorithms with arbitrarily many stages. More precisely, we prove that there are trivial two-stage algorithms that determine all the positives using a worst-case number of tests equal to the information theoretic lower bound on the problem that, of course, is a lower bound on the number of tests required by any algorithm, independently on the number of performed stages.

There is another feature that differentiate biologically motivated group testing problems from the traditional ones. In the classical scenario it is assumed that the presence of a single positive object in a pool is sufficient for the test to produce a positive result. However, recent work [26] suggests that classical group testing procedures should take into account the possibility of the existence of “inhibitory items”, that is, objects whose presence in the tested set could render the outcome of the test meaningless, as far as the detection of positive objects is concerned. In other words, if during the execution of an algorithm we tested a subset $Q \subseteq \mathcal{O}$ containing positive items *and* inhibitory items, we would get the same answer as Q did not contain any positive object. Similar issues were considered in [19] where further motivations for the problem were

given. Our contribution to the latter issue is an algorithm that determines all positives in a set of objects, containing also up to a certain number of inhibitory items, that uses the optimal worst-case number of tests, considerably improving on results of [20] and [26]. An interesting feature of our algorithm is that it can be implemented to run in only 4 stages.

We also consider the important situation in which a trivial two-stage strategy is used to find the set of positives, given that some prior information about them has been provided in terms of a Bernoulli probability distribution, that is, it is assumed that each object has a fixed probability q of being positive. Usually q is a function $q(n)$ of $n = |\mathcal{O}|$. This situation has received much attention [6, 7, 8, 39], starting from the important work [33]. The relevant parameter in this scenario is the average number of tests necessary to determine all positives. We prove that trivial two-stage strategies can asymptotically attain the information theoretic lower bound for a large class of probability functions $q(n)$. It should be remarked that there are values of $q(n)$ for which lower bounds on the average number of tests better than the information theoretic lower bounds exist [6, 33].

Our results depend on a combinatorial structure we introduce in this paper: (k, m, n) -selectors, to be formally defined in Section 2. Our definition of (k, m, n) -selectors includes as particular cases well known combinatorial objects like superimposed codes [32, 25] and k -selectors [13]. Superimposed codes and k -selectors are very basic combinatorial structures and find application in an amazing variety of situations, ranging from cryptography to data security [35, 45] to computational molecular biology [5, 20, 23, 29], from multi-access communication [23, 32] to database theory [32], from pattern matching [30] to distributed coloring [37], circuit complexity [12], broadcasting in radio networks [13, 14], and other areas in computer science. We believe that our (k, m, n) -selectors will prove be useful in several different areas as well.

1.1 Previous Results

We address the reader to the excellent monographs [1, 2, 23] for a survey of the vast literature on Group Testing. The papers [29, 33, 26] include a very nice account of the most important results on biologically motivated group testing problems. To the best of our knowledge, our paper is the first to address the problem of estimating the worst case complexity of trivial two-stage group testing algorithms. The problem of estimating the minimum expected number of tests of trivial two-stage group testing algorithms when it is known that any item has a probability $p = p(n)$ of being positive has been studied in [6, 7, 8, 33, 39]. The papers most related to our results are [33, 7]. In particular, the paper [33] proves that for several classes of probability functions $p(n)$ trivial two-stage group testing procedures are inherently more costly than fully adaptive group testing procedures (interestingly, we prove that this is not so in the worst case analysis). The paper [7], with a real *tour-de-force* of the probabilistic method, provides a sharp estimate of the minimum expected number of tests of trivial two-stage procedures for an ample class of probability functions $p(n)$. Our approach is simpler and still it allows to obtain the correct order of magnitude

of the minimum expected number of tests of the trivial two-stage group testing procedure for several classes of probability functions. A more detailed comparison of our results with those of [7] will be given at the end of Section 4. Finally, the study of group testing in presence of inhibitory items, the subject matter of our Section 5, was initiated in [26], continued in [20] and, under different models, also in [21] and [19].

1.2 Summary of the Results and Structure of the Paper

In Section 2 we formally define our main combinatorial tool, (k, m, n) -selectors, and give bounds on their size. These bounds will be crucial for all our subsequent results. In Section 3 we present a two-stage group testing algorithm with asymptotically optimal worst-case complexity. In Section 3 we also present some related results of independent interest. For instance, we prove an $\Omega(k \log n)$ lower bound on the size of k -selectors defined in [13], improving on the lower bound $\Omega(\frac{k}{\log k} \log n)$ mentioned in [31]. This bound shows that the construction in [13] is optimal. We also apply our results to solve the open problem mentioned in [26] of estimating the minimum number of different pools (not tests!) required by a two-stage group testing algorithm. Finally, we also establish an interesting link between our results and the problem of learning boolean functions in a constant number of rounds, in the sense of [16]. In Section 4 we present our results on two-stage procedures when a probability distribution on the possible set of positives is assumed. Finally, in Section 5 we present a worst-case optimal algorithm for group testing in presence of inhibitory items, improving on the algorithms given in [20] and [26].

2 (k, m, n) -Selectors and Bounds on Their Sizes

In this section we introduce our main combinatorial tool: (k, m, n) -selectors. We point out their relationships with other well known combinatorial objects and provide upper and lower bounds on their sizes.

Definition 1. *Given integers k , m , and n , with $1 \leq m \leq k \leq n$, we say that a boolean matrix M with t rows and n columns is a (k, m, n) -selector if any submatrix of M obtained by choosing k out of n arbitrary columns of M contains at least m distinct rows of the identity matrix I_k . The integer t is the size of the (k, m, n) -selector.*

Our notion of (k, m, n) -selector includes as particular cases well known combinatorial structures previously defined in the literature. It is possible to see that k -cover free families [25], disjunctive codes [23], superimposed codes [32], and strongly selective families [14,13] correspond to our notion of $(k+1, k+1, n)$ -selector. The k -selectors of [13] coincides with our definition of $(2k, 3k/2+1, n)$ -selectors.

We are interested in providing upper and lower bounds on the minimum size $t = t(k, m, n)$ of (k, m, n) -selectors. Upper bounds will be obtained by translating

the problem into the hypergraph language. Given a finite set X and a family $\mathcal{F}$ of subsets of X , an hypergraph is a pair $\mathcal{H} = (X, \mathcal{F})$. Elements of X will be called vertices of $\mathcal{H}$, elements of $\mathcal{F}$ will be called hyperedges of $\mathcal{H}$. A *cover* of $\mathcal{H}$ is a subset $T \subseteq X$ such that for any hyperedge $E \in \mathcal{F}$ we have $T \cap E \neq \emptyset$. The minimum size of a cover of $\mathcal{H}$ will be denoted by $\tau(\mathcal{H})$. A fundamental result by Lovász [38] implies that

$$\tau(\mathcal{H}) < \frac{|X|}{\min_{E \in \mathcal{F}} |E|} (1 + \ln \Delta), \quad (1)$$

where $\Delta = \max_{x \in X} |\{E : E \in \mathcal{F} \text{ and } x \in E\}|$.

Essentially, Lovász proves that, by greedily choosing vertices in X that intersect the maximum number of yet non-intersected hyperedges of $\mathcal{H}$, one obtains a cover of size smaller than the right-hand side of (1). Our aim is to show that (k, m, n) -selectors are covers of properly defined hypergraphs. Lovász's result (1) will then provide us with the desired upper bound on the minimum selector size.

We shall proceed as follows. Let X be the set of all binary vectors $\mathbf{x} = (x_1, \dots, x_n)$ of length n containing n/k 1's (the value n/k is a consequence of an optimized choice whose justification can be skipped here). For any integer i , $1 \leq i \leq k$, let us denote by $\mathbf{a}_i$ the binary vector of length k having all components equal to zero but that in position i , that is, $\mathbf{a}_1 = (1, 0, \dots, 0)$, $\mathbf{a}_2 = (0, 1, \dots, 0)$, $\dots$, $\mathbf{a}_k = (0, 0, \dots, 1)$. Moreover, for any set of indices $S = \{i_1, \dots, i_k\}$, with $1 \leq i_1 \leq i_2 < \dots < i_k \leq n$, and for any binary vector $\mathbf{a} = (a_1, \dots, a_k) \in \{\mathbf{a}_1, \dots, \mathbf{a}_k\}$, let us define the set of binary vectors $E_{\mathbf{a}, S} = \{\mathbf{x} = (x_1, \dots, x_n) \in X : x_{i_1} = a_1, \dots, x_{i_k} = a_k\}$. For any set $A \subseteq \{\mathbf{a}_1, \dots, \mathbf{a}_k\}$ of size r , $r = 1, \dots, k$, and any set $S \subseteq \{1, \dots, n\}$, with $|S| = k$, let us define $E_{A, S} = \bigcup_{\mathbf{a} \in A} E_{\mathbf{a}, S}$. For any $r = 1, \dots, k$ we define $\mathcal{F}_r = \{E_{A, S} : A \subset \{\mathbf{a}_1, \dots, \mathbf{a}_k\}, |A| = r, \text{ and } S \subseteq \{1, \dots, n\}, |S| = k\}$ and the hypergraph $\mathcal{H}_r = (X, \mathcal{F}_r)$. We claim that *any* cover T of $\mathcal{H}_{k-m+1}$ is a (k, m, n) selector, that is, any submatrix of k arbitrary columns of T contains at least m distinct rows of the identity matrix I_k . The proof is by contradiction. Assume that there exists a set of indices $S = \{i_1, \dots, i_k\}$ such that the submatrix of T obtained by considering only the columns of T with indices $i_1, \dots, i_k$ contains *at most* $m - 1$ distinct rows of I_k . Let such rows be $\mathbf{a}_{j_1}, \dots, \mathbf{a}_{j_s}$, with $s \leq m - 1$ and let A be *any* subset of $\{\mathbf{a}_1, \dots, \mathbf{a}_k\} \setminus \{\mathbf{a}_{j_1}, \dots, \mathbf{a}_{j_s}\}$ of cardinality $|A| = k - m + 1$ and $E_{A, S}$ be the corresponding hyperedge of $\mathcal{H}_{k-m+1}$. By construction, we have that $T \cap E_{A, S} = \emptyset$, contradicting the fact that T is a cover for $\mathcal{H}_{k-m+1}$.

The above proof that (k, m, n) -selectors coincide with the covers of $\mathcal{H}_{k-m+1}$ allows us to use Lovász's result (1) to give upper bounds on the minimum size of selectors.

Theorem 1. *For any integers k, m and n , with $1 \leq m \leq k < n$, there exists a (k, m, n) -selector of size t , with*

$$t < \frac{ek^2}{k - m + 1} \ln \frac{n}{k} + \frac{ek(2k - 1)}{k - m + 1},$$

where $e = 2.7182\dots$ is the base of the natural logarithm.

Remark Applying the above theorem to (k, k, n) -selectors, that is, to $k-1$ -cover free families, one recovers the usual upper bound of $O(k^2 \log n)$ on their size [24, 25]. Applying the above theorem to $(2k, 3k/2 + 1, n)$ -selectors, (that is, to k -selectors in the sense of [13]) one gets the same upper bound of $O(k \log n)$ on their size, with better constant (22 vs. 87). By concatenating $(k, \alpha k, n)$ -selectors, $\alpha < 1$, of suitably chosen parameter k one gets in a simple way the same combinatorial structure of [34], with the same asymptotic upper bound given therein, but our constants are much better (44 vs. $\sim 5 \cdot 10^5$, according to [11]).

In order to present our first lower bound on the size of (k, m, n) -selectors we need to recall the definition of (p, q) -superimposed codes [20, 24].

Definition 2. *Given integers p, q and n , with $p + q \leq n$, we say that a $t \times n$ boolean matrix M is a (p, q) -superimposed code if for any choice of two subsets P and Q of columns of M , where $P \cap Q = \emptyset$, $|P| = p$, and $|Q| = q$, there exists a row in M such that all columns in Q have a zero in correspondence to that row, and at least a column in P has a one in correspondence to the same row. The integers n and t are the size and the length of the (p, q) -superimposed code, respectively. The minimum length of a (p, q) -superimposed code of size n is denoted by $t_s(p, q, n)$.*

It can be shown that (k, m, n) -selectors are $(k - m + 1, m - 1)$ -superimposed codes. Therefore, lower bounds on the length of (p, q) -superimposed codes translates into lower bounds on selectors. The following theorem can be obtained by combining results of [24] and [27].

Theorem 2. *For any positive integers p, q and n , with $p \leq q$ and $n \geq p + q$, the minimum length $t_s(p, q, n)$ of a (p, q) -superimposed code of size n is at least*

$$t \geq \frac{p \lfloor q/p \rfloor^2}{4 \log(\lfloor q/p \rfloor) + O(1)} \log \frac{n}{p}.$$

By setting $p = k - m + 1$ and $q = m - 1$ in the above lower bound one obtains the following lower bound on the size of (k, m, n) -selectors.

Corollary 1. *For any integers k, m and n , with $1 \leq m \leq k \leq n$, $k < 2m - 2$ the minimum size $t(k, m, n)$ of a (k, m, n) -selector is at least*

$$t(k, m, n) \geq \frac{(k - m + 1) \lfloor (m - 1)/(k - m + 1) \rfloor^2}{4 \log(\lfloor (m - 1)/(k - m + 1) \rfloor) + O(1)} \log \frac{n}{k - m + 1}. \quad (2)$$

3 Application of (k, m, n) -Selectors to Optimal 2-Stage Group Testing

We have a set of objects $\mathcal{O}$, $|\mathcal{O}| = n$, and a subset $\mathcal{P} \subseteq \mathcal{O}$ of positives, $|\mathcal{P}| = p$. The task is to determine the members of $\mathcal{P}$ by asking subset queries of the form “does the subset $\mathcal{Q} \subseteq \mathcal{O}$ contain a positive object?”. We focus on the so called trivial two-stages algorithms. Recall that these algorithms consist of two stages:

in the first stage a certain set of pools are tested in parallel and in the second stage only individual objects are tested (always in parallel). Which individual objects are tested may depend on the outcomes of the first stage.

In the following we provide a 2-stage algorithm which uses an asymptotically optimal number of tests. We associate each item of the input set $\mathcal{O}$ to a distinct column of a $(k, p + 1, n)$ -selector $M = [M(i, j)]$. Let t denote the size of the $(k, p + 1, n)$ -selector. For $i = 1, \dots, t$, we define $T_i = \{j \in \{1, \dots, n\} : M(i, j) = 1\}$. The first stage of the algorithm consists of testing the t pools $T_1, \dots, T_t$ in parallel. Let $\mathbf{f}$ denote the binary vector collecting the answers of the t tests (here a “yes” answer to test T_i corresponds to a 1-entry in the i -th position of $\mathbf{f}$, and a “no” answer corresponds to a 0 entry). Notice that $\mathbf{f}$ is the boolean sum of the p columns associated with the p positives. It is easy to see that in addition to the columns associated with the p positives items, there are at most $k - p - 1$ columns which are “covered” by $\mathbf{f}$, that is, that have the 1’s in a subset of the positions in which also the vector $\mathbf{f}$ has 1’s. Let $y_1, \dots, y_p$ denote the p positives. Assume by contradiction that there are more than $k - p - 1$ columns, other than those associated with $y_1, \dots, y_p$, which are covered by $\mathbf{f}$. Let $z_1, \dots, z_{k-p}$ denote $k - p$ such columns and let us consider the submatrix of M consisting of $y_1, \dots, y_p, z_1, \dots, z_{k-p}$. By Definition 1 one has that this submatrix contains at least $p + 1$ rows of the identity matrix I_k . At least one of these $p + 1$ rows of I_k has a 1 in one of columns $z_1, \dots, z_{k-p}$. Let ℓ denote the index of such a row. Since the columns associated to $y_1, \dots, y_p$ have the ℓ -th entry equal to 0, then one has that the ℓ -th entry of $\mathbf{f}$ is 0 thus contradicting the hypothesis that $\mathbf{f}$ covers all columns $z_1, \dots, z_{k-p}$. Using this argument one concludes that if we discard all columns which are not covered by $\mathbf{f}$ then we are left with $k - 1$ columns p of which correspond to the p positives. Stage 2 consists of individually probing these $k - 1$ elements. The following theorem holds.

Theorem 3. *Let t be the size of a $(k, p + 1, n)$ -selector. There exists a 2-stage group testing algorithm to find p positives out of n items that uses a number of tests equal to $t + k - 1$.*

From Theorem 1 and Theorem 3 we get the following

Corollary 2. *For any integers k, p and n , with $1 \leq p < k \leq n$, there exists a 2-stage group testing algorithm to find p positives using a number of tests less than*

$$\frac{ek^2}{k-p} \ln \frac{n}{k} + \frac{ek(2k-1)}{k-p} + k - 1. \quad (3)$$

By optimizing the choice of k to $k = 2p$ in (3), we get the main result of this section.

Corollary 3. *For any integers p and n , with $1 \leq p \leq n$, there exists a 2-stage group testing algorithm to find p positives using a number of tests less than*

$$4ep \ln \frac{n}{2p} + p(8e + 2) - 2e - 1 < 7.54p \log_2 \frac{n}{p} + 16.21p - 2e - 1$$

The 2-stage algorithm of the above corollary is asymptotically optimal because of the information theoretic lower bound on the number of tests given by

$$\log_2 \binom{n}{p} > p \log_2 \frac{n}{p}, \quad (4)$$

that holds also for fully adaptive group testing algorithms.

Finally, we also remark that our algorithm can be easily modified to run in the same asymptotic complexity also when only an upper bound on the number of positives is known.

3.1 Deriving a Lower Bound on the Size of (k, m, n) -Selectors via 2-Stage Group Testing

Let $g(p, n)$ denote the minimum number of tests needed to identify p positive items out of n items by a group testing strategy. Theorem 3 and the information theoretic lower bound (4) give

$$\log_2 \binom{n}{p} \leq g(n, p) \leq t(k, p+1, n) + k - 1,$$

from which we get the following result that provides a lower bound on the size of (k, m, n) -selectors also for values of k and m not covered by (2).

Theorem 4. *For any integers k, m and n , with $1 \leq m \leq k < n$, the minimum size $t(k, m, n)$ of a (k, m, n) -selector satisfies*

$$t(k, m, n) \geq \log \binom{n}{m-1} - k + 1 \geq (m-1) \log \frac{n}{m-1} - k + 1.$$

Theorem 4 implies a lower bound of $\Omega(k \log \frac{n}{k})$ on the size of the k -selectors of [13] (that is, of our $(2k, 3k/2 + 1, n)$ -selectors), improving on the lower bound of $\Omega(\frac{k}{\log k} \log \frac{n}{k})$ mentioned in [31]. Our lower bound is optimal since it matches the upper bound on the size of k -selectors given in [13].

3.2 Estimating the Number of Pools in 2-Stage Algorithms

Classical group testing theory measures the cost of an algorithm to find the positives by the number of tests the algorithm requires. As stressed in [26], there are situations in which the number of *constructed pools* may be the dominant cost of an algorithm. Bearing this in mind, the authors of [26] proposed the following research problem. Denote by $N(v, h)$ the maximum size of a search space $\mathcal{O}$ such that any potential subset of up to p positives can be successfully identified by using a total of v different pools and at most h excess confirmatory tests in the second stage. Excess confirmatory tests are those individual tests that involve negative objects. The problem is to estimate

$$f(p, h) = \limsup_{v \rightarrow \infty} \frac{\log_2 N(v, h)}{v}.$$

The authors of [26] noted that classical results on superimposed codes [24] imply

$$\frac{\ln 2}{p^2}(1 + o(1)) \leq f(p, 0) \leq \frac{2 \log_2 p}{p}(1 + o(1)),$$

where the $o(1)$ is for $p \rightarrow \infty$, and posed as an open problem that of estimating $f(p, h)$, for $h > 0$. This estimation for $h \geq p$ can be obtained from our previous results. Notice that $f(p, h)$ is increasing in h . It is now possible to see that (4) and our Corollaries 2 and 3 allow us to determine $f(p, h)$ up to a constant, (the rather easy computations will be given in the full paper).

Theorem 5. *With the notation as above, we have*

$$\frac{1}{7.54p} \leq f(p, h) \leq \frac{1}{p}, \text{ for all } h \geq 2p,$$

$$\frac{\alpha - 1}{e\alpha^2 p \ln 2} \leq f(p, \alpha p) \leq \frac{1}{p}, \text{ for all } 1 < \alpha < 2.$$

3.3 A Remark on Learning Monotone Boolean Functions

We consider here the well known problem of exact learning an unknown boolean function of n variables by means of membership queries, provided that at most k of the variables (attributes) are relevant. This is known as attribute-efficient learning. With membership queries one means the following [3]: The learner chooses a 0-1 assignment x of the n variables and gets the value $f(x)$ of the function at x . The goal is to learn (identify) the unknown function f exactly, using a small number of queries. Typically, one assumes that the learner knows in advance that f belongs to a restricted class of boolean functions, since the exact learning problem in the full generality admits only trivial solutions. In this scenario, the group testing problem is equivalent to the problem of exactly learning an unknown function f , where it is known that f is an *OR* of at most p variables.

Recently, P. Damaschke in a series of papers [16,17,18] studied the power of adaptive vs. non adaptive attribute efficient learning. In this framework he proved that adaptive learning algorithms are more powerful than non adaptive ones. More precisely, he proved that in general it is impossible to learn monotone boolean functions with k relevant variables in less than $\Omega(k)$ stages, if one insists that the total number of queries be of the same order of that used by the best fully adaptive algorithm (i.e., an algorithm that may use an arbitrary number of stages, see [16,17] for details). In view of Damaschke's results, we believe worthwhile to state our Corollary 3 in the following form.

Corollary 4. *Boolean functions made by the disjunction of at most p variables are exactly learnable in only two stages by using a number of queries of the same order as that of the best fully adaptive learning algorithm.*

Above remark raises the interesting question of characterizing monotone boolean functions “optimally” learnable in a constant number of stages. Another example of class of functions optimally learnable in a constant number of stages will be given at the end of Section 5.

4 Two-Stage Algorithms for Probabilistic Group Testing

In this section we assume that objects in $\mathcal{O}$, $|\mathcal{O}| = n$, have some probability $q = q(n)$ of being positive, independently from each other. This means that the probability distribution on the possible subsets of positive is a binomial distribution, which is a standard assumption in the area (e.g., [6,7,33]). In this scenario one is interested in minimizing the average number of queries necessary to identify all positives. Shannon's source coding theorem implies that the minimum average number of queries is lower bounded by the entropy

$$n(-q(n) \log q(n) - (1 - q(n)) \log(1 - q(n))). \quad (5)$$

It is also known [6,33] that for some values of the probability $q(n)$ the lower bound (5) is not reachable, in the sense that better lower bounds exist. Our algorithm for the probabilistic case is very simple and is based on the following idea. Given the probability $q = q(n)$ that a single object in $\mathcal{O}$ be positive, we estimate the expected number of positives $\mu = nq(n)$. We now run the 2-stage algorithm described in Section 3, using a (k, m, n) -selector with parameters $m = (1 + \delta)\mu + 1$, with $\delta > 0$, and $k = 2(1 + \delta)\mu$. Denote by X the random variable taking value i if and only if the number of positives in $\mathcal{O}$ is exactly i . X is distributed according to a binomial distribution with parameter q and mean value μ . If the number of positives is at most $(1 + \delta)\mu$, and this happens with probability $\Pr[X \leq (1 + \delta)\mu]$, then by the result of Section 3 the execution of the queries of stage 1 will restrict our search to $2(1 + \delta)\mu$ elements which will be individually probed during stage 2. Stage 1 requires $O(m \log \frac{n}{m})$ queries. If, on the contrary, the number of positives is larger than $(1 + \delta)\mu$, then the feedback vector $\mathbf{f}$ might cover more than $2(1 + \delta)\mu$ columns of the selector. Consequently a larger number of elements, potentially *all* n elements, must be individually probed in stage 2. The crucial observation is that this latter unfavourable event happens with probability $\Pr[X > (1 + \delta)\mu]$. All together, the above algorithm uses an average number of queries E given by

$$E = O(m \log \frac{n}{m}) + n\Pr[X > (1 + \delta)\mu]. \quad (6)$$

Choosing $\delta \geq 2e$ and by recalling that $m = (1 + \delta)\mu + 1$, we get from (6) and by Chernoff bound ([41], p.72) that

$$E = O(nq(n) \log \frac{1}{q(n)}) + n2^{-(1+\delta)nq(n)}. \quad (7)$$

A similar idea was used in [7]. However, the authors of [7] used classical superimposed codes in the first stage of their algorithm, and since these codes have size much larger than our selectors, their results are worse than ours. Recalling now the information theoretic lower bound (5) on the expected number of queries, we get from (7) that our algorithm is asymptotically optimal whenever the probability function $q(n)$ satisfies the following condition

$$q(n) \geq \frac{1}{n}(\log \frac{1}{q(n)} - \log \log \frac{1}{q(n)} - O(1)). \quad (8)$$

For instance, $q(n) = c \frac{\log n}{n}$ for any positive constant c or $q(n)$ such that $\frac{q(n)n}{\log n} \rightarrow \infty$ satisfy (8). The previous two cases were explicitly considered in [6] where the authors obtain results similar to ours, with better constants. Nevertheless, our condition (8) is more general. The main difference between our results and those of [6] consists of the following. Here we estimate the average number of queries of our *explicitly defined algorithm*. Instead, the authors of [6] estimate the average number of queries performed by a two-stage algorithm where the boolean matrix used in the first stage is randomly chosen among all $m \times n$ binary matrices, where the choice of m depends on $q(n)$. Using a very complex and accurate analysis, they probabilistically show the *existence* of two stage algorithms with good performances. For several classes of probability functions $q(n)$ they are able to give asymptotic upper and lower bounds on the minimum average number of queries that differs in several cases only by a multiplicative constant.

5 An Optimal 4-Stage Group Testing Algorithm for the GTI Model

In this section we consider the group testing with inhibitors (GTI model) introduced in [26]. We recall that, in this model, in addition to positive items and regular items, there is also a category of items called inhibitors. The inhibitors are the items that interfere with the test by hiding the presence of positive items. As a consequence, a test yields a positive feedback if and only if the tested pool contains one or more positives and no inhibitor. We present an optimal worst case 4-stage group testing algorithm to find p positives in the presence of r inhibitors.

stage 1. The goal of this stage is to find a pool $Q \subseteq \mathcal{O}$ which tests positive. To this aim, we associate each item to a distinct column of a (p, r) -superimposed code $M = [M(i, j)]$. Let t be the length of the code. For $i = 1, \dots, t$ we construct the pool $T_i = \{j \in \{1, \dots, n\} : M(i, j) = 1\}$. If we test pools $T_1, \dots, T_t$, then the feedback vector has the i -th entry equal to 1 if and only if at least one the columns associated to the p positives has the i -th entry equal to 1, whereas none of the columns associated to the r inhibitors has the i -th entry equal to 1. It is easy to prove that such an entry i exists, by using the fact that the code M is (p, r) -superimposed. Stage 1 returns $Q = T_i$.

stage 2. The goal of this stage is to remove all inhibitors from the set $\mathcal{O}$. To this aim we associate each item not in Q to a distinct column of a $(k', r+1, n - |Q|)$ -selector M' . Let t' be the size of the selector. For $i = 1, \dots, t'$ we construct the pool $T'_i = \{j \in \{1, \dots, n\} : M'(i, j) = 1\}$. If we test pools $T'_1 \cup Q, \dots, T'_{t'} \cup Q$, then the feedback vector $\mathbf{f}'$ has the i -th entry equal to 0 if and only if T'_i contains one or more inhibitors. Hence, the feedback vector $\mathbf{f}'$ is equal to the intersection (boolean product) of the bitwise complement of the columns associated with the r inhibitors. Let $\bar{\mathbf{f}}'$ be the bitwise complement of $\mathbf{f}'$. The column $\bar{\mathbf{f}}'$ is equal to the boolean sum of the columns associated to the r inhibitors. Using an argument

similar to that used for the 2-stage group testing algorithm of Section 3, one has that $\vec{\mathbf{f}}'$ covers at most $k' - r$ columns in addition to those associated with the r inhibitor items. We put apart all k' items covered by $\vec{\mathbf{f}}'$. These k' items will be individually probed in stage 4 since some of them might be defective items.

stage 3. The goal of this stage is to discard a “large” number of regular items from the set of $n - k'$ items remaining after stage 2. The present stage is similar to stage 1 of our 2-stage algorithm of Section 3. We associate each of the $n - k'$ items to a distinct column of a $(k'', p + 1, n - k')$ -selector M'' . Let t'' be the size of the selector. For $i = 1, \dots, t''$ we construct the pool $T_i'' = \{j \in \{1, \dots, n\} : M''(i, j) = 1\}$ and test pools $T_1'', \dots, T_{t''}''$. Notice that after stage 2 there is no inhibitor among the searched set of items and consequently the feedback vector $\mathbf{f}''$ is equal to the boolean sum of the columns associated with the positive items in the set (those which have not been put apart in stage 2). After these t'' tests we discard all items but those corresponding to columns covered by the feedback vector $\mathbf{f}''$. Hence, we are left with k'' items.

stage 4. We individually probe the k' items returned by stage 2 and the k'' items returned by stage 3.

The above algorithm provides the following general result.

Theorem 6. *Let k' , k'' , n , p , and r be integers with $1 \leq r < k' < n$ and $1 \leq p < k'' < n - k'$. There exists a 4-stage group testing algorithm to find p positives in the presence of r inhibitors by*

$$t_s(p, r, n) + t(k', r + 1, n - |Q|) + t(k'', p + 1, n - k') + k' + k''$$

tests.

The following main corollary of Theorem 6 holds.

Corollary 5. *Let p , and r be integers with $1 \leq r < n$ and $1 \leq p < n - 2r$. There exists a 4-stage group testing algorithm to find p positives in the presence of r inhibitors by*

$$t_s(p, r, n) + O\left(r \log \frac{n}{r} + p \log \frac{n - r}{p}\right) \quad (9)$$

tests, and this upper bound is asymptotically optimal.

Proof. By setting $k' = 2r$ and $k'' = 2p$ in Theorem 6 and using the bound of Theorem 1 on the size of selectors, one gets the following upper bound on the number of tests performed by the 4-stage algorithm

$$t_s(p, r, n) + 4er \ln \frac{n - |Q|}{2r} + 2e(4r - 1) + 4ep \ln \frac{n - 2r}{2p} + 2e(4p - 1) + 2r + 2p. \quad (10)$$

We now prove that the above upper bound is asymptotically optimal. In [20] it has been proved a lower bound of

$$\Omega \left(t_s(p, r, n - p - 1) + \ln \binom{n}{p} \right) \quad (11)$$

on the number of tests required by *any* algorithm (using any number of stages) to find p defectives in the presence of r inhibitors. Since it is $t_s(p, r, n - p - 1) = \Theta(t_s(p, r, n))$, then lower bound (11) is

$$\Omega \left(t_s(p, r, n) + \ln \binom{n}{p} \right). \quad (12)$$

It is possible to see that expression (12) is $\Omega(t_s(p, r, n) + r \log \frac{n}{r} + p \log \frac{n}{p})$. If $p > r$, then this is immediate. If $p \leq r$, Theorem 2 implies the following lower bound on the length of a (p, r) -superimposed code of size n

$$t_s(p, r, n) \geq \frac{p \lfloor r/p \rfloor^2}{4 \log(\lfloor r/p \rfloor) + O(1)} \log \frac{n}{p}. \quad (13)$$

It is possible to see that the right hand side of (13) is $\Omega(r \log \frac{n}{r})$. Therefore, one has that expression (12) is $\Omega(t_s(p, r, n) + r \log \frac{n}{r} + p \log \frac{n}{p})$. It follows that the upper bound (10) on the number of tests performed by the 4-stage algorithm is tight with lower bound (11).

□

We can employ a $(p + r, r + 1, n)$ -selector in stage 1 of the algorithm and use the bound of Theorem 1 on the size of selectors to estimate the number of tests performed by this stage. Notice that the weight of the rows of the $(p + r, r + 1, n)$ -selector corresponds to the size of the pools tested during stage 1 and consequently to that of the set Q returned by this stage. By using the construction of Theorem 1 one has that the size of Q is $\frac{n}{r+1}$. Hence, the following result holds.

Corollary 6. *For any integers p, r and n , with $p \geq 1$, $r \geq 0$ and $p + r \leq n$, there exists a 4-stage group testing algorithm to find p positives in a set of n elements, r of which can be inhibitors, using a number of tests at most*

$$\frac{e(p+r)^2}{p} \ln \frac{n}{p+r} + 4er \ln \frac{n}{2(r+1)} + 4ep \ln \frac{n-2r}{2p} + (10e+2)p + (12e+2)r - 5e + \frac{er(2r-1)}{p}.$$

It is remarkable that for $r = O(p)$ Corollary 6 implies that our deterministic algorithm attains the same asymptotic complexity $O((r+p) \log n)$ of the randomized algorithm presented in [26].

In the same spirit of Section 3.3 we mention that the problem of finding p positives in the presence of r inhibitors is equivalent to the problem of learning an unknown boolean function of the form $(x_1 \vee \dots \vee x_p) \wedge (\overline{y_1 \vee \dots \vee y_r})$. Hence, above results can be rephrased as follows.

Corollary 7. *Boolean functions of the form $(x_1 \vee \dots \vee x_p) \wedge (\overline{y_1 \vee \dots \vee y_r})$ are exactly learnable in only four stages by using a number of queries of the same order as that of the best fully adaptive learning algorithm.*

References

1. R. Ahlswede and I. Wegener, *Search Problems*, John Wiley & Sons, New York, 1987.
2. M. Aigner, *Combinatorial Search*, Wiley-Teubner, New York-Stuttgart, 1988.
3. D. Angluin, “Queries and concept learning”, *Machine Learning*, vol. 2, 319–342, 1987.
4. E. Barillot, B. Lacroix, and D. Cohen, “Theoretical analysis of library screening using an n -dimensional pooling strategy”, *Nucleic Acids Research*, 6241–6247, 1991.
5. D.J. Balding, W.J. Bruno, E. Knill, and D.C. Torney, “A comparative survey of non-adaptive pooling design” in: Genetic mapping and DNA sequencing, *IMA Volumes in Mathematics and its Applications*, T.P. Speed & M.S. Waterman (Eds.), Springer-Verlag, 133–154, 1996.
6. T. Berger and V. I. Levenshtein, “Asymptotic efficiency of two-stage disjunctive testing”, *IEEE Transactions on Information Theory*, **48**, N. 7, 1741–1749, 2002.
7. T. Berger and V. I. Levenshtein, “Application of cover-free codes and combinatorial design to two-stage testing”, to appear in *Discrete Applied Mathematics*.
8. T. Berger, J.W. Mandell, and P. Subrahmanya, “Maximally efficient two-stage screening”, *Biometrics*, **56**, No. 3, 833–840, 2000.
9. A. Blass and Y. Gurevich, “Pairwise testing”, in: *Bullettin of the EATCS*, no. 78, 100–131, 2002.
10. W.J. Bruno, D.J. Balding, E. Knill, D. Bruce, C. Whittaker, N. Dogget, R. Stalling, and D.C. Torney, “Design of efficient pooling experiments”, *Genomics*, **26**, 21–30, 1995.
11. P. Bussbach, “Constructive methods to solve problems of s-surjectivity, conflict resolution, and coding in defective memories”, Ecole Nationale des Telecomm., ENST Paris, Tech. Rep. 84D005, 1984.
12. S. Chaudhuri and J. Radhakrishnan, “Deterministic restrictions in circuit complexity”, in *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC 96)*, 30–36, 1996.
13. M. Chrobak, L. Gasieniec, W. Rytter, “Fast Broadcasting and Gossiping in Radio Networks”, in: *Proc. of 42nd IEEE Annual Symp. on Found. of Computer Science (FOCS 2000)*, 575–581, 2000.
14. A.E.F. Clementi, A. Monti and R. Silvestri, “Selective families, superimposed codes, and broadcasting on unknown radio networks”, in *Proc. of Symp. on Discrete Algorithms (SODA’01)*, 709–718, 2001.
15. D.M. Cohen, S. R. Dalal, M. L. Fredman, G.C. Patton, “The AETG System: An Approach to Testing Based on Combinatorial Design”, *IEEE Trans. on Soft. Eng.*, vol. **23**, 437–443, 1997.
16. P. Damaschke, “Adaptive versus Nonadaptive Attribute-Efficient Learning”, in *Proceedings of the Tertieth Annual ACM Symposium on Theory of Computing (STOC 1998)*, 590–596, 1998.
17. P. Damaschke, “Parallel Attribute-Efficient Learning of Monotone Boolean Functions”, in: *Algorithm Theory – SWAT2000*, M. Halldorsson (Ed.), LNCS, vol. 1851, pp. 504–512, Springer-Verlag, 2000.
18. P. Damaschke, “Computational Aspects of Parallel Attribute-Efficient Learning”, in *Proc. of Algorithmic Learning Theory 98*, M. Richter et al. (Eds.), LNCS 1501, Springer-Verlag, 103–111, 1998,

19. P. Damaschke, "Randomized group testing for mutually obscuring defectives", *Information Processing Letters*, **67** (3), 131–135, 1998.
20. A. De Bonis and U. Vaccaro, "Improved algorithms for group testing with inhibitors", *Information Processing Letters*, **66**, 57–64, 1998.
21. A. De Bonis and U. Vaccaro, "Efficient constructions of generalized superimposed codes with applications to Group Testing and conflict resolution in multiple access channels", in *ESA'02*, R. Möring and R. Raman (Eds.), LNCS, vol. 2461, 335–347, Springer-Verlag, 2002.
22. R. Dorfman, "The detection of defective members of large populations", *Ann. Math. Statist.*, **14**, 436–440, 1943.
23. D.Z. Du and F.K. Hwang, *Combinatorial Group Testing and its Applications*, World Scientific, 2000.
24. A.G. Dyachkov, V.V. Rykov, "A survey of superimposed code theory", *Problems Control & Inform. Theory*, **12**, No. 4, 1–13, 1983.
25. P. Erdős, P. Frankl, and Z. Füredi, "Families of finite sets in which no set is covered by the union of r others", *Israel J. of Math.*, **51**, 75–89, 1985.
26. M. Farach, S. Kannan, E.H. Knill and S. Muthukrishnan, "Group testing with sequences in experimental molecular biology", in *Proceedings of Compression and Complexity of Sequences 1997*, B. Carpentieri, A. De Santis, U. Vaccaro, and J. Storer (Eds.), IEEE Computer Society, 357–367, 1997.
27. Z. Füredi, "On r -cover free families", *Journal of Combinatorial Theory*, vol. 73(1), 172–173, 1996.
28. E.H. Hong and R.E. Ladner, "Group testing for image compression", in *Proceedings of Data Compression Conference (DCC2000)*, IEEE Computer Society, 3–12, 2000.
29. Hung Q. Ngo and Ding-Zhu Du, "A survey on combinatorial group testing algorithms with applications to DNA library screening", in *Discrete Mathematical Problems with Medical Applications*, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., **55**, Amer. Math. Soc., 171–182, 2000.
30. P. Indyk, "Deterministic superimposed coding with application to pattern matching", *Proc. of Thirty-ninth Annual IEEE Annual Symp. on Foundations of Computer Science (FOCS 97)*, 127–136, 1997.
31. P. Indyk, "Explicit constructions of selectors and related combinatorial structures, with applications", *SODA 2002*: 697–704.
32. W.H. Kautz and R.R. Singleton, "Nonrandom binary superimposed codes", *IEEE Trans. on Inform. Theory*, **10**, 363–377, 1964.
33. E. Knill, "Lower bounds for identifying subset members with subset queries", in *Proceedings of Symposium on Discrete Algorithms 1995 (SODA 1995)*, 369–377.
34. J. Komlós and A.G. Greenberg, "An asymptotically fast non-adaptive algorithm for conflict resolution in multiple-access channels", *IEEE Trans. on Inform. Theory*, **31**, No. 2, 302–306, 1985.
35. R. Kumar, S. Rajagopalan, and A. Sahai, "Coding constructions for blacklisting problems without computational assumptions", in *Proc. of CRYPTO '99*, LNCS **1666**, Springer-Verlag, 609–623, 1999.
36. C.H. Li, "A sequential method for screening experimental variables", *J. Amer. Sta. Assoc.*, vol. 57, 455–477, 1962.
37. N. Linial, "Locality in distributed graph algorithms", *SIAM J. on Computing*, **21**, 193–201, 1992.
38. L. Lovász, "On the ratio of optimal integral and fractional covers", *Discrete Math.*, **13**, 383–390, 1975.

39. A.J. Macula, “Probabilistic Nonadaptive and Two-Stage Group Testing with Relatively Small Pools and DNA Library Screening”, *Journal of Combinatorial Optimization*, **2**, Issue: 4, 385–397, 1999.
40. D. Margaritis and S. Skiena, “Reconstructing strings from substrings in rounds”, *Proc. of Thirty-seventh IEEE Annual Symposium on Foundations of Computer Science (FOCS 95)*, 613–620, 1995.
41. R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University press, 1995.
42. P. A. Pevzner and R. Lipshutz, “Towards dna sequencing chips”, in: *19th International Conference on Mathematical Foundations of Computer Science*, LNCS vol. 841, Springer Verlag, 143–158, 1994.
43. M. Ruszinkó, “On the upper bound of the size of the r -cover-free families”, *J. of Combinatorial Theory, Series A*, **66**, 302–310, 1994.
44. M. Sobel and P.A. Groll, “Group testing to eliminate efficiently all defectives in a binomial sample”, *Bell Syst. Tech. J.*, vol. 38, 1179–1252, 1959.
45. D.R. Stinson, T. van Trung and R. Wei, “Secure frameproof codes, key distribution patterns, group testing algorithms and related structures”, *J. of Statistical Planning and Inference*, **86**, 595–617, 2000.
46. J. Wolf, “Born again group testing: Multiaccess Communications”, *IEEE Trans. Information Theory*, vol. IT-31, 185–191, 1985.

Decoding of Interleaved Reed Solomon Codes over Noisy Data

Daniel Bleichenbacher¹, Aggelos Kiayias², and Moti Yung³

¹ Bell Laboratories,
Murray Hill, NJ, USA
`bleichen@bell-labs.com`

² Department of Computer Science and Engineering,
University of Connecticut, Storrs, CT, USA
`aggelos@cse.uconn.edu`

³ Department of Computer Science,
Columbia University, New York, NY, USA
`moti@cs.columbia.edu`

Abstract. We consider error-correction over the Non-Binary Symmetric Channel (NBSC) which is a natural probabilistic extension of the Binary Symmetric Channel (BSC). We propose a new decoding algorithm for interleaved Reed-Solomon Codes that attempts to correct all “interleaved” codewords simultaneously. In particular, interleaved encoding gives rise to multi-dimensional curves and more specifically to a variation of the Polynomial Reconstruction Problem, which we call Simultaneous Polynomial Reconstruction. We present and analyze a novel probabilistic algorithm that solves this problem. Our construction yields a decoding algorithm for interleaved RS-codes that allows efficient transmission arbitrarily close to the channel capacity in the NBSC model.

1 Introduction

Random noise assumptions have been considered extensively in the coding theory literature with substantial results. One prominent example is Forney Codes [For66] that were designed over the binary symmetric channel (BSC). The BSC suggests that when transmitting binary digits, errors are independent and every bit transmitted has a fixed probability of error. The BSC provides a form of a random noise assumption, which allows probabilistic decoding for message rates that approach the capacity of the channel.

Worst-case non-ambiguous decoding (i.e., when only a bound on the number of faults is assumed and a unique solution is required) has a natural limitation of correcting a number of errors that is up to half the distance of the code. Going beyond this natural bound, either requires re-stating the decoding problem (e.g. consider list-decoding: output all possible decodings for a corrupted codeword), or assuming some “noise assumption” that will restrict probabilistically the combinatorial possibilities for a multitude of possible solutions. Typically, such assumptions are associated with physical properties of given channels (e.g.,

bursty noise, etc.). Recent breakthrough results by Guruswami and Sudan in list-decoding ([Sud97,GS98]) showed that decoding beyond the natural error-correction bound is possible in the worst-case, by outputting all possible decodings. Naturally, there are still limitations in the case of worst-case decoding that prohibit the decoding of very high error-rates.

In this work, motivated by the above, we investigate a traditional channel model that is native to the non-binary setting. The channel is called “Non-Binary Symmetric Channel” (NBSC), presented in figure 1.

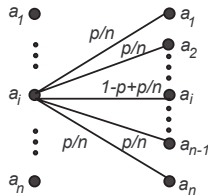


Fig. 1. A non-binary symmetric channel over an alphabet of n symbols. The probability of successful transmission is $1 - p + p/n$. We will refer to p as the error-rate of the NBSC.

As a channel model for bit-level transmission the Non-Binary Symmetric Channel model usually applies to settings where aggregates of bits are sent and errors are assumed to be bursty. Thus, in contrast with the Binary Symmetric Channel, errors in consecutive bits are assumed from a Coding Theoretic perspective to be correlated. There are additional situations that have been considered in a number of Computer Science settings where the NBSC describes the transmission model. For example, consider the case of Information Dispersal Algorithms (IDA) introduced by Rabin in [Rab89] for omission errors, and extended by Krawczyk [Kra92] to deal with general errors. In this setting, a word is encoded into a codeword and various portions of the codeword are sent over different radio network channels, some of which may introduce errors. In the case where the channels are operating in different frequencies, errors may be introduced by jammed channels which emit white noise. Namely, they randomize the transmitted symbol. As a result the communication model in this case approximates the NBSC. Another setting which approximates the NBSC is the transmission of encrypted data where each sub codeword is sent encrypted with what is called “error propagation encryption mode.” These popular modes (e.g. the CBC mode), over noisy channels, will produce a transmission that also approximates the NBSC model ([MOV96], page 230). Moreover the NBSC model has been used in the cryptographic setting as a way to hide information in schemes that employ intractability assumptions related to the hardness of decoding, see e.g. [KY01].

In this work we concentrate on Reed-Solomon Codes. The decoding problem of Reed-Solomon Codes (aka the Polynomial Reconstruction problem — PR) has

been studied extensively, see e.g. [Ber96,Sud97,GS98]. Here, we present a variation of the PR, which we call “Simultaneous Polynomial Reconstruction” and we present a novel probabilistic algorithm that solves it for settings of the parameters that are beyond the currently known solvability bounds for PR (without any effect on the solvability of the latter problem). Our algorithm is probabilistic and is employed in settings where errors are assumed to be random.

Next we concentrate on the “code interleaving” encoding schema, see e.g. section 7.5, [VV89], which is a technique used to increase the robustness of a code in the setting of burst errors. We consider the problem of decoding interleaved Reed-Solomon Codes and we discover the relationship of this problem to the problem of Simultaneous Polynomial Reconstruction. In particular we show that the two problems are equivalent when interleaved Reed-Solomon Codes are applied over a channel that satisfies the NBSC model.

Subsequently using our algorithm for Simultaneous Polynomial Reconstruction we present a novel decoding algorithm for interleaved Reed-Solomon Codes in the NBSC model that is capable of correcting any error-rate up to $\frac{r}{r+1}(1-\kappa)$ where r is the “amount of interleaving” and κ is the message rate.

We observe that traditional decoding of interleaved RS-Codes does not improve the error-rate that can be corrected. In fact, error-rates only up to $\frac{1-\kappa}{2}$ can be corrected (uniquely) in the worst-case, and in the NBSC model list-decoding algorithms ([GS98]) for unique decoding can be also employed thus correcting error-rates up to $1 - \sqrt{\kappa}$.

Nevertheless using our algorithm for Simultaneous Polynomial Reconstruction we correct error-rates up to $\frac{r}{r+1}(1-\kappa)$ (with high probability). An immediate corollary is that we can correct any error-rate bounded away from $(1-\kappa)$ provided that the alphabet-size is selected to be large enough. In other words, interleaved RS-Codes reach the channel’s capacity as the amount of interleaving $r \rightarrow \infty$ (something that requires that the alphabet-size n over which the NBSC model is employed to also satisfy $n \rightarrow \infty$).

Organization. In section 2 we present our variation of the Polynomial Reconstruction problem and we describe and analyze a probabilistic algorithm that solves this problem. Subsequently in section 3 we describe the relation of this problem to the decoding of Interleaved Reed-Solomon codes and we show how our algorithm is employed in this domain. We use the notation $[n]$ to denote the set $\{1, \dots, n\}$.

2 The Algorithm

In this section we present a probabilistic algorithm that solves efficiently the following problem, which we call the Simultaneous Polynomial Reconstruction:

Definition 1. (*Simultaneous Polynomial Reconstruction — SPR*) For $n, k, t, r \in \mathbb{N}$, an instance of SPR is a set of tuples $\{\langle z_i, y_{i,1}, \dots, y_{i,r} \rangle\}_{i=1}^n$ over a finite field $\mathbb{F}$ with $i \neq j \rightarrow z_i \neq z_j$ that satisfies the following:

1. There exists an $I \subseteq [n]$ with $|I| = t$, and polynomials $p_1, \dots, p_r \in \mathbb{F}[x]$ of degree less than k , such that $p_\ell(z_i) = y_{i,\ell}$ for all $i \in I$ and $\ell \in [r]$.

2. For all $i \notin I, \ell \in [r]$ it holds that $y_{i,\ell}$ are uniformly distributed over $\mathbb{F}$.

Goal: Recover $p_1, \dots, p_r$.

We remark that the goal of Simultaneous Polynomial Reconstruction, assuming a large underlying finite-field $\mathbb{F}$, is well-defined (in other words the probability that another tuple of r polynomials $p'_1, \dots, p'_r$ exists that would fit the data in the same way $p_1, \dots, p_r$ do, is very small). Taking this into account, the SPR problem with parameters n, k, t, r reduces easily to the Polynomial Reconstruction Problem with parameters n, k, t , (by simply reducing the n tuples to pairs by discarding $r - 1$ coordinates — it follows easily that the recovery of p_1 would reveal the remaining polynomials). Thus, we would be interested in algorithmic solutions for the SPR problem when the parameters n, k, t are selected to be beyond the state-of-the-art solvability of the PR problem.

2.1 Description of the Algorithm

The algorithmic construction that we present amends the prototypical decoding paradigm (fitting the data through an error-locator polynomial, see e.g. [BW86, Ber96]) to the setting of Simultaneous Polynomial Reconstruction. More specifically our algorithm can be seen as a generalization of the Berlekamp-Welch algorithm for Reed-Solomon Decoding, [BW86]. The parameter settings where our algorithm works is

$$t \geq \frac{n + rk}{r + 1}$$

observe that for $r = 1$ the above bound on t coincides with the bound of the [BW86]-algorithm, whereas when $r > 1$ less agreement is required (t is allowed to be smaller).

Let $\{\langle z_i, y_{i,1}, \dots, y_{i,r} \rangle\}_{i=1}^n$ be an instance of the SPR problem with parameters n, k, t, r . Further observe that the condition on t above implies that $r \geq \frac{n-t}{t-k}$.

Define the following system of rn equations:

$$[m_1(z_i) = y_{i,1}E(z_i)]_{i=1}^n \dots [m_r(z_i) = y_{i,r}E(z_i)]_{i=1}^n \quad (*)$$

where the unknowns are the coefficients of the polynomials $m_1, \dots, m_r, E$. Each m_ℓ is a polynomial of degree less than $n - t + k$ and E is a polynomial of degree at most $n - t$ with constant term equal to 1. It follows that the system has $r(n - t + k) + n - t$ unknowns and thus it is not underspecified (i.e., the number of equations is at least as large as the number of unknowns); this follows from the condition on r .

Our algorithm for SPR simply solves system $(*)$ to recover the polynomials $m_1, \dots, m_r, E$ and outputs $m_1/E, \dots, m_r/E$ as the solution to the given SPR instance. This is accomplished by selecting an appropriate square sub-system of $(*)$ defined explicitly in section 2.3.

This completes the description of our algorithm. We argue about its correctness in the following two sections. We remark that the novelty of our approach relies on the probabilistic method that is employed to ensure the uniqueness of the error-locator polynomial E .

2.2 Feasibility

In this section we argue that for a given SPR instance $\{\langle z_i, y_{i,1}, \dots, y_{i,r} \rangle\}_{i=1}^n$, one of the possible outputs of the algorithm of section 2.1 is the solution of the SPR instance. Observe that due to item 1 of definition 1, there exists $I \subseteq [n]$ with $|I| = t$ such that $p_\ell(z_i) = y_{i,\ell}$ for $i \in I$ and all $\ell \in [r]$ for some polynomials $p_1, \dots, p_r \in \mathbb{F}[x]$ (which constitute the solution of the SPR instance).

Let $\tilde{E}(x) = (-1)^{n-|I|} \prod_{i \notin I} (x/z_i - 1)$. Observe that $\tilde{E}$ has constant term 1 and degree $n - t$. Further, if $\tilde{m}_\ell(x) := p_\ell(x) \tilde{E}(x)$ it holds that $\tilde{m}_\ell(z_i) = p_\ell(z_i) \tilde{E}(z_i) = y_{i,\ell} \tilde{E}(z_i)$, for all $i = 1, \dots, n$. The degree of $\tilde{m}_\ell$ is less than $n - t + k$. Observe that the polynomials $\tilde{E}, \tilde{m}_1, \dots, \tilde{m}_r$ constitute a possible solution of the system (*). Moreover (by construction) $\tilde{m}_\ell(x)/\tilde{E}(x) = p_\ell(x)$ for $\ell = 1, \dots, r$ and as a result one of the possible outputs of the algorithm of section 2.1 is indeed the solution of the given SPR instance.

2.3 Uniqueness

The crux of the analysis of our algorithm is the technique we introduce to show the uniqueness of the solution constructed in the previous section.

In a nutshell we will present a technique for constructing a minor for the matrix of system (*) that is non-singular with high probability. It is exactly at this point that item 2 of definition 1 will be employed in a non-trivial manner. We present the technique as part of the proof of the theorem below. The reader is also referred to figure 2 for a graphical representation of the method.

Theorem 1. *The matrix of the linear system (*) has a minor of order $r(n - t + k) + n - t$ denoted by $\hat{A}$ that is non-singular with probability at least $1 - \frac{n-t}{|\mathbb{F}|}$.*

Proof. Consider the following matrices, for $\ell = 1, \dots, r$:

$$M = \begin{pmatrix} 1 & z_1 & z_1^2 & \dots & z_1^{n-t+k-1} \\ 1 & z_2 & z_2^2 & \dots & z_2^{n-t+k-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & z_n & z_n^2 & \dots & z_n^{n-t+k-1} \end{pmatrix} \quad M_\ell = \begin{pmatrix} y_{1,\ell} z_1 & y_{1,\ell} z_1^2 & \dots & y_{1,\ell} z_1^{n-t} \\ y_{2,\ell} z_2 & y_{2,\ell} z_2^2 & \dots & y_{2,\ell} z_2^{n-t} \\ \vdots & \vdots & \dots & \vdots \\ y_{n,\ell} z_n & y_{n,\ell} z_n^2 & \dots & y_{n,\ell} z_n^{n-t} \end{pmatrix}$$

Given these definitions, it follows that the matrix of the system (*) is the following (where $\mathbf{0}$ stands for a $n \times (n - t + k)$ -matrix with 0's everywhere):

$$A = \begin{pmatrix} M & \mathbf{0} & \dots & \mathbf{0} & -M_1 \\ \mathbf{0} & M & \dots & \mathbf{0} & -M_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & M & -M_r \end{pmatrix}$$

We index each row of A by the pair $\langle i, \ell \rangle$ with $i \in \{1, \dots, n\}$ and $\ell \in \{1, \dots, r\}$. The ℓ -th block row of A contains the rows $\langle 1, \ell \rangle, \dots, \langle n, \ell \rangle$.

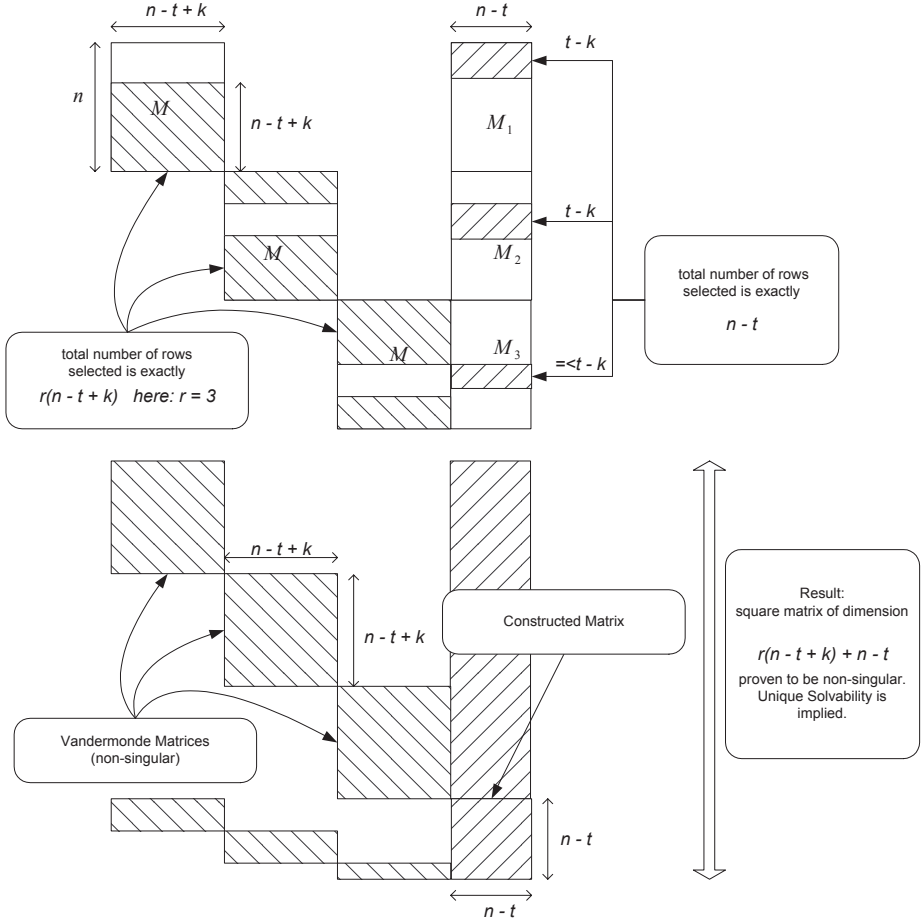


Fig. 2. Constructing the matrix $\hat{A}^*$ from the matrix of the system (*). Refer to the proof of theorem 1 for the definitions of the matrices shown above.

Now we select a square sub-matrix $\hat{A}$ of A by removing $r(t-k) - (n-t)$ rows as follows: starting from the r -th block row we remove a number of rows $x \in \{0, \dots, t-k\}$ indexed by $\langle n, r \rangle, \dots, \langle n-t+k+1, r \rangle$ (in this order) until $\hat{A}$ becomes square or x reaches $t-k$. Then we repeat the same procedure for the block-row $(r-1)$ and so on, until $\hat{A}$ becomes square. Next, we will show that $\hat{A}$ is non-singular with high probability.

Without loss of generality we assume that $I = \{n-t+1, \dots, n\}$. The proof is identical for any other choice of I .

Now let us denote by N_ℓ a $(n-t+k)$ -Vandermonde matrix over the elements $\{z_1, \dots, z_n\} - \{z_{1+(\ell-1)(t-k)}, \dots, z_{\ell(t-k)}\}$. Also we define M'_ℓ to be the sub-matrix of M_ℓ with the rows $\langle x + (\ell-1)(t-k), \ell \rangle$ removed for $x = 1, \dots, t-k$. Finally let

V_ℓ be a $(n-t) \times (n-t+k)$ -matrix that is 0 everywhere except for the rows u that satisfy the property that there is an $x \in [t-k]$ such that $u = x + (\ell-1)(t-k) \leq n-t$; such a row of V_ℓ will be equal to the tuple $\langle 1, z_u, \dots, z_u^{n-t+k-1} \rangle$. The matrix $\hat{A}^*$ defined below is a rearrangement of the rows of $\hat{A}$.

$$\hat{A}^* = \begin{pmatrix} N_1 & \mathbf{0} & \dots & \mathbf{0} & -M'_1 \\ \mathbf{0} & N_2 & \dots & \mathbf{0} & -M'_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & N_r & -M'_r \\ V_1 & V_2 & \dots & V_r & -\hat{M} \end{pmatrix}$$

where the right low corner matrix $\hat{M}$ is defined below:

$$\hat{M} = \begin{pmatrix} y_{1,1}z_1 & y_{1,1}z_1^2 & \dots & y_{1,1}z_1^{n-t} \\ y_{2,1}z_2 & y_{2,1}z_2^2 & \dots & y_{2,1}z_2^{n-t} \\ \vdots & \vdots & \dots & \vdots \\ y_{t-k,1}z_{t-k} & y_{t-k,1}z_{t-k}^2 & \dots & y_{t-k,1}z_{t-k}^{n-t} \\ y_{t-k+1,2}z_{t-k+1} & y_{t-k+1,2}z_{t-k+1}^2 & \dots & y_{t-k+1,2}z_{t-k+1}^{n-t} \\ y_{t-k+2,2}z_{t-k+2} & y_{t-k+2,2}z_{t-k+2}^2 & \dots & y_{t-k+2,2}z_{t-k+2}^{n-t} \\ \vdots & \vdots & \dots & \vdots \\ y_{2(t-k),2}z_{2(t-k)} & y_{2(t-k),2}z_{2(t-k)}^2 & \dots & y_{2(t-k),2}z_{2(t-k)}^{n-t} \\ \vdots & \vdots & \dots & \vdots \\ y_{n-t,\ell}z_{n-t} & y_{n-t,\ell}z_{n-t}^2 & \dots & y_{n-t,\ell}z_{n-t}^{n-t} \end{pmatrix}$$

We will argue that $\hat{A}^*$ is non-singular. First observe that the determinant of $\hat{A}^*$ can be seen as a multivariate polynomial over the variables $y_{i,\ell}$ where $i \in [n]$ and $\ell \in [r]$ (taking into account the fact that $y_{i,\ell}$ for $i \in I$ are only k -wise independent — note that without loss of generality we assume that the solution of a SPR instance is uniformly random: indeed given a SPR instance we can easily randomize its solution by adding a random polynomial of degree less than k to each of the r coordinates; naturally, if a solution is found we will have to subtract the randomization polynomial from each coordinate).

Suppose now we want to eliminate V_1 . In particular to eliminate the first non-zero row of V_1 we should find $\lambda_{t-k+1}, \dots, \lambda_n$ such that $\sum_{j=t-k+1}^n \lambda_j z_j^m = -z_1^m$ for each $m \in [n-t+k-1] \cup \{0\}$.

Now let us choose some assignment for the values $y_{1,1}, \dots, y_{n,1}$; we set $y_{1,1} = \dots = y_{t-k,1} = 2$ and $y_{t-k+1,1} = \dots = y_{n,1} = 1$. It follows that the first row of $\hat{M}$ is rewritten as $\langle 2z_1, \dots, 2z_1^{n-t} \rangle$. It follows that after the elimination of the first row of V_1 the first row of $\hat{M}$ becomes equal to $\langle z_1, \dots, z_1^{n-t} \rangle$.

Regarding the step above, observe the following: (i) the assignment we did for the $y_{i,\ell}$ values is consistent with their dependency condition: $y_{n-t+1,\ell}, \dots, y_{n,\ell}$ must be k -wise independent; (ii) by applying the same elimination method to the remaining non-zero rows of $V_1, V_2, \dots, V_r$ and for each $\ell \in [r]$ making the assignment $y_{i,\ell} = 2$ for each $i \in \{x + (\ell-1)(t-k) \leq n-t \mid x = 1, \dots, t-k\}$ and

$y_{i,\ell} = 1$ otherwise, it follows that we will eliminate all $V_1, \dots, V_r$. After this is accomplished observe that in place of the matrix $\hat{M}$ there will be a Vandermonde-like matrix of order $n - t$ that is non-singular.

It follows that $\det(\hat{A}^*)$ (seen as a multivariate-polynomial) is not the zero-polynomial and thus, by Schwartz's Lemma [Sch80], it cannot be 0 in more than a $\frac{n-t}{|\mathbb{F}|}$ -fraction of its domain (where $n - t$ is the total degree of the polynomial $\det(\hat{A}^*)$). As a result $\det(\hat{A}^*)$ will be 0 with probability at most $\frac{n-t}{|\mathbb{F}|}$. ■

It follows easily from the above theorem that the system $(*)$ accepts at most one solution. Naturally the non-singularity of $\hat{A}$ is not sufficient to ensure the existence of a solution. Nevertheless we know that $(*)$ accepts at least one solution (as constructed explicitly in section 2.2). It follows that system $(*)$ has a unique solution (that coincides with the solution constructed in section 2.2) and this solution can be found by solving the system that has $\hat{A}$ as its matrix.

To improve the efficiency of our algorithm observe that it is not necessary to solve the linear-system with matrix $\hat{A}$ directly; instead, we can derive easily a system of $n - t$ equations that completely determines the polynomial E ; it is obvious that the recovery of E will reveal all solutions of the given SPR instance. This is so, since finding all roots of E will reveal the error-locations of the given SPR-instance and then the recovery of $p_1, \dots, p_r$ can be done by interpolation. A system of $n - t$ equations that determines E completely can be found by eliminating all variables that correspond to the polynomials m_ℓ from at most $t - k$ rows of the ℓ -th block row of matrix $\hat{A}$, for $\ell = 1, \dots, r$. Such elimination will be possible for exactly $n - t$ rows.

3 Decoding Interleaved RS-Codes in the NBSC Model

In this section we present a coding theoretic application of our algorithm of section 2 to the case of interleaved Reed-Solomon Decoding. First we recall the notion of interleaved codes.

3.1 Interleaved Codes

Interleaved codes are not an explicit family of codes, but rather an encoding mode that can be instantiated over any concrete family of codes. In The mode can be applied to any family of codes; in this section we give a code independent description.

Let Σ' be an alphabet with $|\Sigma'| = \sqrt[r]{|\Sigma|}$. Let $\phi : \Sigma \rightarrow (\Sigma')^r$ be some 1-1 mapping. We will denote $\phi(x)$ by the string $x^\phi[1]x^\phi[2] \dots x^\phi[r]$, where $x^\phi[\ell] \in \Sigma'$, for $\ell = 1, \dots, r$, for any $x \in \Sigma$.

Now let $enc : (\Sigma')^k \rightarrow (\Sigma')^n$ be an encoding function. An interleaved code w.r.t. ϕ for enc is a function $enc_\phi : (\Sigma)^k \rightarrow (\Sigma)^n$ that is defined as follows: Let $m_0 m_1 \dots m_{k-1} \in (\Sigma')^k$. First the following strings of $(\Sigma')^n$ are computed:

$$\begin{aligned}
c_{1,1} \dots c_{n,1} &= \text{enc}(m_0^\phi[1] \dots m_{k-1}^\phi[1]) \\
&\vdots \\
c_{1,r} \dots c_{n,r} &= \text{enc}(m_0^\phi[r] \dots m_{k-1}^\phi[r])
\end{aligned}$$

The interleaved encoding is defined as follows:

$$\text{enc}_\phi(m_0 m_1 \dots m_{k-1}) = \phi^{-1}(c_{1,1} \dots c_{1,r}) \dots \phi^{-1}(c_{n,1} \dots c_{n,r})$$

A graphical representation of code interleaving is presented in figure 3.

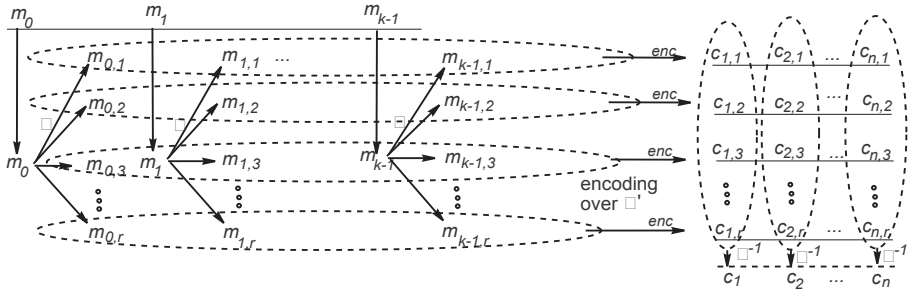


Fig. 3. Encoding schema for an interleaved code. Single subscript symbols (m_i, c_i) belong to the “outer” alphabet Σ ; double subscript symbols $(m_{i,j}, c_{i,j})$ belong to the “inner” alphabet Σ' .

Such interleaved encodings will be said to be of degree r over the alphabet Σ' (we will also call it “amount of interleaving”).

The common way to use an interleaved code, is simply decode each of the code words $(c_{1,i} \dots c_{n,i})$ separately. Such a decoding does not increase the error correction rate. The advantage is the fact that burst errors are distributed over several code words, and therefore employing interleaving over bursty channels increases the chances of error-correction.

We emphasize here that under reasonable channel assumptions it might be possible to take advantage of interleaving and attempt to correct all code words simultaneously. Indeed, in contrast to the standard approach of decoding each one of the codewords individually, we will present a decoding technique that attempts to correct all codewords simultaneously assuming that the NBSC model describes the transmission channel. This methodology will increase the possible error-rates that the interleaved code can withstand.

3.2 Interleaved Reed-Solomon Codes

Let $\Sigma = GF(2^B)$ be the alphabet for the encoding function (without loss of generality we will focus only on binary extension fields — all our results hold

also for general finite fields). The parameters are $n, k \in \mathbb{N}$ where $\kappa := k/n$ is the message rate. We assume additionally a parameter $r \in \mathbb{N}$ with the property $br = B$ (we remark here that a similar scheme is also possible when B is prime, however, for notational simplicity we do not deal with this case in this abstract). Let $z_1, \dots, z_n \in GF(2^b)$ be fixed distinct constants.

We now describe the case of interleaved Reed-Solomon Codes. First, observe that there exists a straightforward bijection mapping $\phi : GF(2^B) \rightarrow (GF(2^b))^r$. Given $m_0 \dots m_{k-1} \in GF(2^B)$ we define the following polynomials over $GF(2^b)$, for $\ell = 1, \dots, r$:

$$p_\ell(x) := m_0^\phi[\ell] + m_1^\phi[\ell]x + \dots + m_{k-1}^\phi[\ell]x^{k-1}$$

The encoding of $m_0 \dots m_{k-1}$ is set to be the string

$$\phi^{-1}(p_1(z_1) \dots p_r(z_1)) \dots \phi^{-1}(p_1(z_n) \dots p_r(z_n))$$

The common way to decode RS-interleaved-codes is to concentrate to each of the r coordinates individually and employ the decoding algorithm of the underlying RS-Code over Σ' . This can be done as follows: given a (partially corrupted) codeword $c_1 \dots c_n \in (\Sigma)^n$ we treat the string $c_1^\phi[1] \dots c_n^\phi[1] \in (\Sigma')^n$ as a partially corrupted RS-codeword over Σ' and we employ the RS-Decoding of Berlekamp-Welch to recover p_1 . Observe that the recovery of p_1 will imply the recovery of $p_2, \dots, p_r$ immediately, provided that the error-rate is at most $\frac{1-\kappa}{2}$ (the error-rate is taken over the channel that transmits $GF(2^B)$ symbols; it is easy to verify that in the NBSC model all codewords $c_1^\phi[\ell] \dots c_n^\phi[\ell]$, $\ell = 1, \dots, r$ will have identical error-pattern with very high probability).

Moreover, due to assured unique solution with high probability in our case, one can further employ the Guruswami-Sudan list-decoding algorithm that will produce a unique solution with high probability for error-rates up to $1 - \sqrt{\kappa}$. The main focus of the next section is to go beyond this bound.

3.3 The Decoding Algorithm

In this section we reduce the problem of decoding interleaved Reed-Solomon Codes in the NBSC model to the problem of Simultaneous Polynomial Reconstruction. Given this result, our algorithm for the latter problem yields a decoding algorithm for interleaved RS-codes.

Consider interleaved RS-Codes with parameters $r, n, k, t \in \mathbb{N}$, where r is the amount of interleaving. Also let $\phi : GF(2^B) \rightarrow GF(2^b)^r$ be the bijection mapping employed for the interleaving.

Let $c_1 \dots c_n \in (GF(2^B))^n$ be the received codeword. Let $y_{i,1} \dots y_{i,r} = \phi(c_i)$, with $y_{i,\ell} \in GF(2^b)$ for all $i = 1, \dots, n, \ell = 1, \dots, r$.

Suppose now that $i \in \{1, \dots, n\}$ is an error-location for the codeword $c_1 \dots c_n$. It follows that c_i is uniformly distributed over $GF(2^B)$ (because of the employment of the NBSC model). Since ϕ is a bijection it follows easily that each of $y_{i,1}, \dots, y_{i,r}$ are uniformly distributed over $GF(2^b)$.

On the other hand there exist polynomials $p_1, \dots, p_r \in GF(2^b)[x]$ of degree less than k such that for all $i \in \{1, \dots, n\}$ with i not an error-location, it holds that $y_{i,1} = p_1(z_i), \dots, y_{i,r} = p_r(z_i)$. The following proposition is immediate:

Proposition 1. *Let $c_1 \dots c_n \in GF(2^B)^n$ be an encoding of a message $m_0 \dots m_{k-1} \in GF(2^B)^k$ using the interleaved Reed-Solomon encoding scheme with parameters n, k, r that has e errors (over the NBSC model). Then the tuples $\{\langle z_i, y_{i,1}, \dots, y_{i,r} \rangle\}_{i=1}^n$ as defined above constitute an instance of the SPR problem with parameters $n, k, t := n - e, r$ over the field $GF(2^b)$, $b = B/r$.*

Based on our algorithm of section 2 we deduce:

Corollary 1. *There exists a decoding algorithm for interleaved Reed-Solomon codes over parameters n, k, r that corrects any error-rate ϵ up to*

$$\epsilon \leq \frac{r}{r+1}(1 - \kappa)$$

with probability $1 - \frac{n-t}{2^b}$.

Example: Suppose that the message-rate is $1/4$ and the error-rate is $11/16$. We employ the interleaved RS-schema for $r = 11$ with alphabets $\Sigma = GF(2^B) = GF(2^{440})$ and $\Sigma' = GF(2^b) = GF(2^{40})$. Observe that such error-rates are not correctable by considering the interleaved codewords individually (indeed, even list-decoding algorithms, e.g. the [GS98]-method would work only for error-rates up to $1/2$). Suppose now that the block-size is $n = 64$. Our probabilistic decoding algorithm for such interleaved RS-codes corresponds to solving the SPR problem on parameters $n = 64, k = 16, t = 20, r = 11$ over the finite-field $GF(2^{40})$ and thus we will succeed in decoding with probability least $1 - 2^{-34}$.

Remark: We note that employing our methodology, setting and analysis techniques in other cases (i.e. simultaneous decoding of all interleaved codewords for other families of interleaved codes in the NBSC model) is an interesting research direction.

An independent solution of the Simultaneous Polynomial Reconstruction Problem was presented recently by Coppersmith and Sudan in [CS03]. Their solution requires $t > \sqrt[r+1]{nkr} + k + 1$ which improves on our bound $t \geq \frac{n+rk}{r+1}$ in cases where $t > 2k$.

Acknowledgement. The authors wish to thank Alexander Barg for helpful discussions.

References

- [Ber96] Elwyn R. Berlekamp, *Bounded distance+1 soft-decision Reed-Solomon decoding*, IEEE Trans. Info. Theory, vol. IT-42, pp. 704–720, May 1996.
- [BW86] Elwyn R. Berlekamp and L. Welch, *Error Correction of Algebraic Block Codes*. U.S. Patent, Number 4,633,470, 1986.

- [CS03] Don Coppersmith and Madhu Sudan, *Reconstructing Curves in Three (and higher) Dimensional Space from Noisy Data*, to appear in the proceedings of the 35th ACM Symposium on Theory of Computing (STOC), June 9–11, 2003, San Diego, California.
- [For66] G. David Forney, *Concatenated Codes*, MIT Press, Cambridge, MA, 1966
- [GS98] Venkatesan Guruswami and Madhu Sudan, *Improved Decoding of Reed-Solomon and Algebraic-Geometric Codes*. In the Proceedings of the 39th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, pp. 28–39, 1998.
- [KY01] Aggelos Kiayias and Moti Yung, *Secure Games with Polynomial Expressions*, In the Proceedings of the 28th International Colloquium in Algorithms, Languages and Programming (ICALP), 2001, LNCS Vol. 2076, pp. 939–950.
- [Kra92] Hugo Krawczyk, *Distributed Fingerprints and Secure Information Dispersal*, PODC 1992, pp. 207–218.
- [MS77] F. J. MacWilliams and N. Sloane, *The Theory of Error Correcting Codes*. North Holland, Amsterdam, 1977.
- [MOV96] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
- [Rab89] Michael O. Rabin, *Efficient dispersal of information for security, load balancing, and fault tolerance*, J. ACM 38, pp. 335–348, 1989.
- [Sch80] J. T. Schwartz, *Fast Probabilistic Algorithms for Verifications of Polynomial Identities*, Journal of the ACM, Vol. 27(4), pp. 701–717, 1980.
- [Sud97] Madhu Sudan, *Decoding of Reed Solomon Codes beyond the Error-Correction Bound*. Journal of Complexity 13(1), pp. 180–193, 1997.
- [VV89] S. A. Vanstone and P. C. VanOorshot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic Publishers, 1989.

On the Axiomatizability of Ready Traces, Ready Simulation, and Failure Traces

Stefan Blom¹, Wan Fokkink^{1,2}, and Sumit Nain³

¹ CWI, Department of Software Engineering, PO Box 94079, 1090 GB Amsterdam, The Netherlands, {sccblom,wan}@cwi.nl

² Vrije Universiteit Amsterdam, Department of Theoretical Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, wanf@cs.vu.nl

³ IIT Delhi, Department of Computer Science and Engineering, Hauz Khas, New Delhi-110 016, India, nain@cse.iitd.ernet.in

Abstract. We provide an answer to an open question, posed by van Glabbeek [4], regarding the axiomatizability of ready trace semantics. We prove that if the alphabet of actions is finite, then there exists a (sound and complete) finite equational axiomatization for the process algebra BCCSP modulo ready trace semantics. We prove that if the alphabet is infinite, then such an axiomatization does not exist. Furthermore, we present finite equational axiomatizations for BCCSP modulo ready simulation and failure trace semantics, for arbitrary sets of actions.

1 Introduction

Labeled transition systems constitute a fundamental model of concurrent computation, which is widely used in light of its flexibility and applicability. They model processes by explicitly describing their states and their transitions from state to state, together with the actions that produced them. Several notions of behavioral equivalence have been proposed, with the aim to identify those states of labeled transition systems that afford the same observations. The lack of consensus on what constitutes an appropriate notion of observable behavior for reactive systems has led to a large number of proposals for behavioral equivalences for concurrent processes.

Van Glabbeek [4] presented the linear time - branching time spectrum of 15 behavioral equivalences for finitely branching, concrete, sequential processes. For 12 equivalences in this spectrum, van Glabbeek gave an axiomatization that is sound and complete for the process algebra BCCSP modulo such an equivalence. BCCSP is built from the nil 0 , alternative composition $- + -$, and prefixing $a-$, where a ranges over a nonempty set Act of actions.

For three equivalences, based on ready simulation [3,7], failure traces [10] and ready traces [2,11], the axiomatization in [4] includes a conditional equation. For example, for failure trace and ready trace equivalence, the axiomatizations include the conditional equation

$$I(x) = I(y) \Rightarrow a(x + y) \approx ax + ay \quad (1)$$

where $I(p)$ is the set of possible initial actions of process p . In [4, p. 78] it is remarked that for finite alphabets, ready simulation and failure trace equivalence do allow a finite equational axiomatization.

“As observed by Stefan Blom, if Act is finite, ready simulation equivalence can be finitely axiomatized without using conditional equations or auxiliary operators. [...] If Act is finite also failure trace equivalence has a finite equational axiomatization. *However, it is unknown whether the same holds for ready trace equivalence.*”

We present formal proofs of the observations regarding ready simulation and failure trace equivalence, for arbitrary sets of actions. The main part of this paper is devoted to answering the open question regarding ready trace equivalence.

Groote [5] introduced an infinite family of (unconditional) equations that, in the case of finitely branching processes, captures the conditional equation (1):

$$a(\sum_{i=1}^n (b_i x_i + b_i y_i) + z) \approx a(\sum_{i=1}^n b_i x_i + z) + a(\sum_{i=1}^n b_i y_i + z) \quad (2)$$

for $n \in \mathbb{Z}_{>0}$. We prove that if Act consists of k elements, then actually only equation (2) for the case $n = k$ is needed, together with the equations for ready trace equivalence from [4] (excluding (1)), to obtain a (sound and complete) finite equational axiomatization for BCCSP modulo ready trace equivalence. This provides an affirmative answer to van Glabbeek’s question in the case of a finite alphabet.

Van Glabbeek considers occurrences of actions in axioms as concrete action names, so that in the case of an infinite alphabet Act , an axiom such as (1) actually represents an infinite number of conditional equations, one for each $a \in Act$. In this paper we take such an occurrence of a in an axiom to represent a variable of type Act , so that (1) represents a single conditional equation. With the latter interpretation of occurrences of actions in axioms, the equational axiomatizations for 11 of the equivalences in the linear time - branching time spectrum remain finite in the case of an infinite alphabet. However, the finite equational axiomatization for ready trace equivalence given in this paper works only in the case of a finite alphabet, due to the fact that for an infinite alphabet it no longer suffices to select only one equation from the family of equations (2). We prove that in the case of an infinite alphabet, BCCSP modulo ready trace equivalence does not allow a finite equational axiomatization.

Related work: For BCCSP modulo 2-nested simulation [6], which is part of the linear time-branching time spectrum, there does not exist a finite equational axiomatization [1]; not even in the case of a finite alphabet.

Acknowledgement. Rob van Glabbeek is thanked for useful discussions and comments.

2 Preliminaries

Syntax of BCCSP. $\text{BCCSP}(\text{Act})$ is a basic process algebra to express finite process behavior. Its syntax consists of (process) terms that are constructed from a constant $\mathbf{0}$, a binary operator $+$ called *alternative composition*, unary *prefixing* operators $c\_,$ where c ranges over some nonempty set Act of *actions*, and countably infinite disjoint sets of variables $TVar$ of type term (with typical elements x, y, z) and $AVar$ of type action (with typical elements a, b). We shall use t, u, v to range over process terms and c, d, e, f to range over Act . A term is *closed* if it does not contain any variables. Closed terms will be denoted by p, q, r . A (closed) substitution maps variables in $TVar$ to (closed) $\text{BCCSP}(\text{Act})$ terms and variables in $AVar$ to $\text{Act} \cup AVar$ (resp. Act). For every term t and substitution σ , the term obtained by replacing every occurrence of a variable x or a in t with $\sigma(x)$ or $\sigma(a)$, respectively, will be written $\sigma(t)$.

Transition rules. Intuitively, closed terms represent finite process behaviors, where $\mathbf{0}$ does not exhibit any behavior, $p + q$ is the nondeterministic choice between the behaviors of p and q , and cp can execute action c to transform into p . This intuition for the operators of $\text{BCCSP}(\text{Act})$ is captured, in the style of Plotkin [9], by the transition rules below, which give rise to Act -labeled transitions between closed terms.

$$\frac{}{ax \xrightarrow{a} x} \quad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \quad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$$

For a closed term p , $I(p)$ denotes the set of actions c for which there exists a transition $p \xrightarrow{c} p'$ for some process term p' .

Axiomatization. An (equational) *axiomatization* E over $\text{BCCSP}(\text{Act})$ is a collection of equations $t \approx u$. We write $E \vdash t \approx u$ if this equation can be derived from the axioms in E using the standard rules of equational logic. An axiomatization E is *sound* modulo an equivalence $\sim$ over closed terms if $E \vdash p \approx q \Rightarrow p \sim q$, and it is *complete* modulo $\sim$ if $p \sim q \Rightarrow E \vdash p \approx q$, for all closed terms p and q . An axiomatization E is ω -*complete* if for any equation $t \approx u$ such that $E \vdash \sigma(t) \approx \sigma(u)$ for all closed substitutions σ , we have $E \vdash t \approx u$.

The core equations for $\text{BCCSP}(\text{Act})$ are axioms A1-4 below, which are sound and complete modulo bisimulation equivalence [8].

$$\begin{array}{ll} \text{A1} & x + y \approx y + x \\ \text{A2} & (x + y) + z \approx x + (y + z) \\ \text{A3} & x + x \approx x \\ \text{A4} & x + \mathbf{0} \approx x \end{array}$$

BA denotes the set of equations $\{\text{A1}, \text{A2}, \text{A3}, \text{A4}\}$. In the remainder of this paper, process terms are considered modulo associativity and commutativity of $+$, and modulo absorption of $\mathbf{0}$ summands (i.e., modulo A1,2,4). We use *summation* $\sum_{i=1}^n t_i$, with $n \in \mathbb{N}$, to denote $t_1 + \dots + t_n$, where the empty sum denotes $\mathbf{0}$. As binding convention, alternative composition binds weaker than summation, which in turn binds weaker than prefixing.

Ready trace semantics. A sequence $X_0c_1X_1 \cdots c_nX_n$ (with $n \in \mathbb{N}$), where $X_i \subseteq \text{Act}$ and $c_i \in \text{Act}$, is a *ready trace* of p_0 if $p_0 \xrightarrow{c_1} p_1 \cdots \xrightarrow{c_n} p_n$ and $I(p_i) = X_i$ for $i = 0, \dots, n$. Two closed terms p and q are *ready trace equivalent*, denoted by $p \sim_{\text{RT}} q$, if they have exactly the same ready traces. Baeten, Bergstra and Klop [2] proved that *BA* together with one conditional equation C1,

$$\text{C1} \quad I(x) = I(y) \Rightarrow a(x + y) \approx ax + ay$$

is sound and complete for $\text{BCCSP}(\text{Act})$ modulo ready trace equivalence; see also [4]. C1 gives rise to an equality $a(p + q) \approx ap + aq$ if its condition $I(p) = I(q)$ is satisfied.

Theorem 1. *BA $\cup$ {C1} is sound and complete for $\text{BCCSP}(\text{Act})$ modulo ready trace equivalence.*

Failure trace semantics. A sequence $X_0c_1X_1 \cdots c_nX_n$ (with $n \in \mathbb{N}$), where $X_i \subseteq \text{Act}$ and $c_i \in \text{Act}$, is a *failure trace* of p_0 if $p_0 \xrightarrow{c_1} p_1 \cdots \xrightarrow{c_n} p_n$ and $I(p_i) \cap X_i = \emptyset$ for $0 = 1, \dots, n$. Two closed terms p and q are *failure trace equivalent*, denoted by $p \sim_{\text{FT}} q$, if they have exactly the same failure traces. *BA* and C1 together with one equation,

$$\text{FT} \quad ax + ay \approx ax + ay + a(x + y)$$

is sound and complete for $\text{BCCSP}(\text{Act})$ modulo failure trace equivalence; see [4].

Theorem 2. *BA $\cup$ {FT, C1} is sound and complete for $\text{BCCSP}(\text{Act})$ modulo failure trace equivalence.*

Ready simulation. A binary relation R on closed terms is a *simulation* if whenever pRq and $p \xrightarrow{a} p'$, then there is a transition $q \xrightarrow{a} q'$ such that $p'Rq'$. A simulation R is a *ready simulation* if pRq implies $I(p) = I(q)$. Two closed terms p and q are *ready simulation equivalent*, denoted by $p \sim_{\text{RS}} q$, if pR_1q and qR_2p for ready simulations R_1 and R_2 . *BA* together with one conditional equation C2,

$$\text{C2} \quad I(y) \subseteq I(x) \Rightarrow a(x + y) \approx a(x + y) + ax$$

is sound and complete for $\text{BCCSP}(\text{Act})$ modulo ready simulation equivalence; see [4].

Theorem 3. *BA $\cup$ {C2} is sound and complete for $\text{BCCSP}(\text{Act})$ modulo ready simulation equivalence.*

We take occurrences of actions in axioms (such as the a in C1 and C2) to represent variables in AVar .

3 Ready Traces

Groote [5] noted that C1 can be replaced by an infinite family of (unconditional) equations RT_n for $n \in \mathbb{Z}_{>0}$:

$$\text{RT}_n \quad a\left(\sum_{i=1}^n (b_i x_i + b_i y_i) + z\right) \approx a\left(\sum_{i=1}^n b_i x_i + z\right) + a\left(\sum_{i=1}^n b_i y_i + z\right)$$

3.1 Finite Alphabets

We prove that for a finite alphabet Act , consisting of n actions, $BA \cup \{RT_n\}$ is complete for $BCCSP(Act)$ modulo ready trace equivalence.

Lemma 1. $\{RT_n, A3\} \vdash RT_m$ for $m, n \in \mathbb{Z}_{>0}$ with $m \leq n$.

Proof. (Sketch) Substitute b_m for b_i , x_m for x_i and y_m for y_i in RT_n , for $i = m+1, \dots, n$. Next, apply A3 to eliminate multiple occurrences of $b_m x_m$ and $b_m y_m$ in summations. $\square$

Proposition 1. $BA \cup \{RT_n\} \vdash$

$$a\left(\sum_{i=1}^n \left(\sum_{j=1}^{\ell} b_i x_{ij} + \sum_{k=1}^m b_i y_{ik}\right) + z\right) \approx a\left(\sum_{i=1}^n \sum_{j=1}^{\ell} b_i x_{ij} + z\right) + a\left(\sum_{i=1}^n \sum_{k=1}^m b_i y_{ik} + z\right)$$

for $\ell, m, n \in \mathbb{Z}_{>0}$.

Proof. We take n fixed, and prove the equation by induction on $\ell + m$. The base case $\ell = m = 1$ is an instance of RT_n . We proceed with the inductive case, where $\ell + m > 2$; without loss of generality we can assume that $\ell > 1$. IH is shorthand for the induction hypothesis.

$$\begin{aligned} & a\left(\sum_{i=1}^n \left(\sum_{j=1}^{\ell} b_i x_{ij} + \sum_{k=1}^m b_i y_{ik}\right) + z\right) \\ \approx & a\left(\sum_{i=1}^n \left(\sum_{j=1}^{\ell-1} b_i x_{ij} + \sum_{k=1}^m b_i y_{ik}\right) + \left(\sum_{i=1}^n b_i x_{i\ell} + z\right)\right) \\ \approx & a\left(\sum_{i=1}^n \sum_{j=1}^{\ell-1} b_i x_{ij} + \sum_{i=1}^n b_i x_{i\ell} + z\right) + a\left(\sum_{i=1}^n \sum_{k=1}^m b_i y_{ik} + \sum_{i=1}^n b_i x_{i\ell} + z\right) \quad (\text{IH}) \end{aligned}$$

$$\begin{aligned} \approx & a\left(\sum_{i=1}^n \sum_{j=1}^{\ell-1} b_i x_{ij} + \sum_{i=1}^n b_i x_{i\ell} + z\right) \\ & + a\left(\sum_{i=1}^n \sum_{k=1}^m b_i y_{ik} + z\right) + a\left(\sum_{i=1}^n b_i x_{i\ell} + z\right) \quad (\text{IH}) \end{aligned}$$

$$\begin{aligned} \approx & a\left(\sum_{i=1}^n \sum_{j=1}^{\ell-1} b_i x_{ij} + \sum_{i=1}^n b_i x_{i\ell} + z\right) \\ & + a\left(\sum_{i=1}^n b_i x_{i\ell} + \sum_{i=1}^n b_i x_{i\ell} + z\right) + a\left(\sum_{i=1}^n \sum_{k=1}^m b_i y_{ik} + z\right) \quad (\text{A3}) \end{aligned}$$

$$\approx a\left(\sum_{i=1}^n \sum_{j=1}^{\ell-1} b_i x_{ij} + \sum_{i=1}^n b_i x_{i\ell} + \sum_{i=1}^n b_i x_{i\ell} + z\right) + a\left(\sum_{i=1}^n \sum_{k=1}^m b_i y_{ik} + z\right) \quad (\text{IH})$$

$$\approx a\left(\sum_{i=1}^n \sum_{j=1}^{\ell} b_i x_{ij} + z\right) + a\left(\sum_{i=1}^n \sum_{k=1}^m b_i y_{ik} + z\right) \quad (\text{A3})$$

□

Theorem 4. *Let Act consist of n actions. Then $BA \cup \{RT_n\}$ is sound and complete for $BCCSP(Act)$ modulo ready trace equivalence.*

Proof. Let $I(p) = I(q) = \{d_1, \dots, d_m\}$ where $0 \leq m \leq n$. If $m = 0$, then $p \approx \mathbf{0} \approx q$ can be derived using A4. Suppose $m > 0$. Then by applying A3, p and q can be equated to closed terms of the form $\sum_{i=1}^m \sum_{j=1}^{\ell} d_i p_{ij}$ and $\sum_{i=1}^m \sum_{j=1}^{\ell} d_i q_{ij}$, respectively, for some $\ell \in \mathbb{Z}_{>0}$. Hence, by Proposition 1, for each $c \in Act$, $c(p + q) = cp + cq$ can be derived from $BA \cup \{RT_m\}$. So in view of Lemma 1, each closed instantiation of C1 can be derived from $BA \cup \{RT_n\}$.

By Theorem 1, $BA \cup \{C1\}$ is complete for $BCCSP(Act)$ modulo ready trace equivalence. Hence, $BA \cup \{RT_n\}$ is also complete for $BCCSP(Act)$ modulo ready trace equivalence. □

3.2 Infinite Alphabets

We prove that for an infinite alphabet Act , there does not exist a sound and complete finite equational axiomatization for $BCCSP(Act)$ modulo ready trace equivalence. Let RT denote the set of equations $\{RT_n \mid n \in \mathbb{Z}_{>0}\}$.

Corollary 1. *For any Act , $BA \cup RT$ is complete for $BCCSP(Act)$ modulo ready trace equivalence.*

Proof. Let $p \sim_{RT} q$. We take a nonempty, finite set $S \subseteq Act$ containing all actions that occur in p or q ; clearly, p and q are $BCCSP(S)$ -terms. Let S contain n elements. According to Theorem 4, $p \approx q$ can be derived from $BA \cup \{RT_n\}$. □

The following theorem is due to Groote [5]. It does not hold for finite alphabets (cf. the example on p. 321 in [5]).

Theorem 5. *If Act is infinite, then $BA \cup RT$ is ω -complete.*

The proposition below expresses that RT_{n+1} cannot be derived from $BA \cup \{RT_n\}$, for $n \in \mathbb{Z}_{>0}$. First we state without proof a simple lemma.

Lemma 2. Let $\ell, m \in \mathbb{Z}_{>0}$ and $d_1, \dots, d_m, e \in \text{Act}$. For closed substitutions σ ,

$$\begin{aligned} & \sigma\left(\sum_{i=1}^{\ell}(b_i x_i + b_i y_i) + z\right) \sim_{\text{RT}} \sum_{i=1}^m d_i e \mathbf{0} \\ \Leftrightarrow & \sigma\left(\sum_{i=1}^{\ell} b_i x_i + z\right) \sim_{\text{RT}} \sum_{i=1}^m d_i e \mathbf{0} \quad \wedge \quad \sigma\left(\sum_{i=1}^{\ell} b_i y_i + z\right) \sim_{\text{RT}} \sum_{i=1}^m d_i e \mathbf{0} \end{aligned}$$

Proposition 2. Let $n \in \mathbb{Z}_{>0}$. Let $d_1, \dots, d_{n+1} \in \text{Act}$ be distinct, and let $e, f \in \text{Act}$ be distinct. Then

$$BA \cup \{\text{RT}_n\} \not\models c\left(\sum_{i=1}^{n+1}(d_i e \mathbf{0} + d_i f \mathbf{0})\right) \approx c\left(\sum_{i=1}^{n+1} d_i e \mathbf{0}\right) + c\left(\sum_{i=1}^{n+1} d_i f \mathbf{0}\right)$$

Proof. Let p be of the form $\sum_{j=1}^{\ell} c p_j + \sum_{k=1}^m c p'_k$ where $\ell, m \in \mathbb{Z}_{>0}$, $p_j \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i e \mathbf{0}$ for $j = 1, \dots, \ell$ and $p'_k \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i f \mathbf{0}$ for $k = 1, \dots, m$. We write $P_{n+1}(p)$ to express that p is of this particular form.

We prove that if $P_{n+1}(p)$ and $p \approx q$ can be derived by one application of an axiom in $BA \cup \{\text{RT}_n\}$, then $P_{n+1}(q)$. We distinguish seven cases.

1. $p \approx q$ is derived by an application of A3.

Then trivially $P_{n+1}(q)$.

2. $p \approx q$ is derived by an application of RT_n within a p_j or p'_k .

Then, by the soundness of RT_n , $P_{n+1}(q)$.

3. The left-hand side $a(\sum_{i=1}^n (b_i x_i + b_i y_i) + z)$ of RT_n is applied to a $c p_j$.

Then $\sigma(\sum_{i=1}^n (b_i x_i + b_i y_i) + z) = p_j \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i e \mathbf{0}$ for a closed substitution σ . By Lemma 2, $\sigma(\sum_{i=1}^n b_i x_i + z) \sim_{\text{RT}} \sigma(\sum_{i=1}^n b_i y_i + z) \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i e \mathbf{0}$. This implies $P_{n+1}(q)$.

4. The left-hand side $a(\sum_{i=1}^n (b_i x_i + b_i y_i) + z)$ of RT_n is applied to a $c p'_k$.

Then $\sigma(\sum_{i=1}^n (b_i x_i + b_i y_i) + z) = p'_k \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i f \mathbf{0}$ for a closed substitution σ . By Lemma 2, $\sigma(\sum_{i=1}^n b_i x_i + z) \sim_{\text{RT}} \sigma(\sum_{i=1}^n b_i y_i + z) \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i f \mathbf{0}$. This implies $P_{n+1}(q)$.

5. The right-hand side $a(\sum_{i=1}^n b_i x_i + z) + a(\sum_{i=1}^n b_i y_i + z)$ of RT_n is applied to a pair of summands $c p_{j_1} + c p_{j_2}$.

Without loss of generality we can assume that $\sigma(\sum_{i=1}^n b_i x_i + z) = p_{j_1}$ and $\sigma(\sum_{i=1}^n b_i y_i + z) = p_{j_2}$ for a closed substitution σ . Then $\sigma(\sum_{i=1}^n b_i x_i + z) \sim_{\text{RT}} \sigma(\sum_{i=1}^n b_i y_i + z) \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i e \mathbf{0}$. By Lemma 2, $\sigma(\sum_{i=1}^n (b_i x_i + b_i y_i) + z) \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i e \mathbf{0}$. This implies $P_{n+1}(q)$.

6. The right-hand side $a(\sum_{i=1}^n b_i x_i + z) + a(\sum_{i=1}^n b_i y_i + z)$ of RT_n is applied to a pair of summands $c p'_{k_1} + c p'_{k_2}$.

Without loss of generality we can assume that $\sigma(\sum_{i=1}^n b_i x_i + z) = p'_{k_1}$ and $\sigma(\sum_{i=1}^n b_i y_i + z) = p'_{k_2}$ for a closed substitution σ . Then $\sigma(\sum_{i=1}^n b_i x_i + z) \sim_{\text{RT}} \sigma(\sum_{i=1}^n b_i y_i + z) \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i f \mathbf{0}$. By Lemma 2, $\sigma(\sum_{i=1}^n (b_i x_i + b_i y_i) + z) \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i f \mathbf{0}$. This implies $P_{n+1}(q)$.

7. The right-hand side $a(\sum_{i=1}^n b_i x_i + z) + a(\sum_{i=1}^n b_i y_i + z)$ of RT_n is applied to a pair of summands $cp_j + cp'_k$. This case leads to a contradiction.

Without loss of generality we can assume that $\sigma(\sum_{i=1}^n b_i x_i + z) = p_j$ and $\sigma(\sum_{i=1}^n b_i y_i + z) = p'_k$ for a closed substitution σ . Since $p_j \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i e \mathbf{0}$,

and $d_1, \dots, d_{n+1}$ are distinct, the first identity yields $\sigma(z) \xrightarrow{d_{i_0}} r \xrightarrow{e} r'$ for some $i_0 \in \{1, \dots, n+1\}$. Then the second identity yields $p'_1 \xrightarrow{d_{i_0}} r \xrightarrow{e} r'$.

Since $e \neq f$, this contradicts the fact that $p'_1 \sim_{\text{RT}} \sum_{i=1}^{n+1} d_i f \mathbf{0}$.

Concluding, if $P_{n+1}(p)$, and $p \approx q$ can be derived by an application of an axiom in $BA \cup \{\text{RT}_n\}$, then $P_{n+1}(q)$. Since $\neg P_{n+1}(c(\sum_{i=1}^{n+1} (d_i e \mathbf{0} + d_i f \mathbf{0}))$ and $P_{n+1}(c(\sum_{i=1}^{n+1} d_i e \mathbf{0}) + c(\sum_{i=1}^{n+1} d_i f \mathbf{0}))$, this proves the proposition. $\square$

Theorem 6. *If Act is infinite, then there does not exist a sound and complete finite equational axiomatization for $\text{BCCSP}(\text{Act})$ modulo ready trace equivalence.*

Proof. Let E be a finite equational axiomatization that is sound for $\text{BCCSP}(\text{Act})$ modulo ready trace equivalence. According to Corollary 1, $BA \cup \text{RT}$ is complete for $\text{BCCSP}(\text{Act})$ modulo ready trace equivalence. Hence, all closed instantiations of equations in E can be derived from $BA \cup \text{RT}$. By Theorem 5, $BA \cup \text{RT}$ is ω -complete, so the equations in E can be derived from $BA \cup \text{RT}$. Since each of these derivations requires only a finite number of applications of axioms in $BA \cup \text{RT}$, and E is finite, the equations in E can be derived from a finite subset of $BA \cup \text{RT}$. In view of Lemma 1, this means that the equations in E can be derived from $BA \cup \{\text{RT}_n\}$ for some $n \in \mathbb{Z}_{>0}$. So by Proposition 2,

$$c(\sum_{i=1}^{n+1} (d_i e \mathbf{0} + d_i f \mathbf{0})) \approx c(\sum_{i=1}^{n+1} d_i e \mathbf{0}) + c(\sum_{i=1}^{n+1} d_i f \mathbf{0})$$

with $d_1, \dots, d_{n+1}$ distinct and e, f distinct, cannot be derived from E . Hence, E is not complete for $\text{BCCSP}(\text{Act})$ modulo ready trace equivalence. $\square$

4 Ready Simulation

We prove that the conditional axiom C2 can be replaced by a single unconditional equation

$$\text{RS} \quad a(x + by + bz) \approx a(x + by + bz) + a(x + by)$$

It is not hard to see that RS is sound modulo ready simulation equivalence.

Theorem 7. $BA \cup \{\text{RS}\}$ is sound and complete for $\text{BCCSP}(\text{Act})$ modulo ready simulation equivalence.

Proof. Let $I(q) \subseteq I(p)$, where q is of the form $\sum_{i=1}^m b_i q_i$. We prove that $a(p+q) \approx a(p+q) + ap$ can be derived from $BA \cup \{\text{RS}\}$, by induction on m . The base case $m = 0$ is trivial, using A3,4. We focus on the inductive case, where $m > 0$.

Since $I(q) \subseteq I(p)$, p contains a summand $b_m p'$. Hence,

$$\begin{aligned} a(p+q) &\approx a(p+q) + a(p + \sum_{i=1}^{m-1} b_i q_i) && (\text{RS}) \\ &\approx a(p+q) + a(p + \sum_{i=1}^{m-1} b_i q_i) + ap && (\text{IH}) \\ &\approx a(p+q) + ap && (\text{RS}) \end{aligned}$$

This completes the inductive argument. Concluding, each closed instance of C2 can be derived from $BA \cup \{\text{RS}\}$.

By Theorem 3, $BA \cup \{\text{C2}\}$ is complete for $\text{BCCSP}(\text{Act})$ modulo ready simulation equivalence. Hence, $BA \cup \{\text{RS}\}$ is also complete for $\text{BCCSP}(\text{Act})$ modulo ready simulation equivalence. $\square$

5 Failure Traces

Theorem 8. $BA \cup \{\text{FT}, \text{RS}\}$ is sound and complete for $\text{BCCSP}(\text{Act})$ modulo failure trace equivalence.

Proof. Let $I(p) = I(q)$. Then $a(p+q) \sim_{\text{RS}} a(p+q) + ap + aq$, so according to Theorem 7, $a(p+q) \approx a(p+q) + ap + aq$ can be derived from $BA \cup \{\text{RS}\}$. By FT, $a(p+q) + ap + aq \approx ap + aq$. Concluding, each closed instance of C1 can be derived from $BA \cup \{\text{FT}, \text{RS}\}$.

By Theorem 2, $BA \cup \{\text{FT}, \text{C1}\}$ is complete for $\text{BCCSP}(\text{Act})$ modulo failure trace equivalence. Hence, $BA \cup \{\text{FT}, \text{RS}\}$ is also complete for $\text{BCCSP}(\text{Act})$ modulo failure trace equivalence. $\square$

References

1. L. Aceto, W.J. Fokkink, and A. Ingólfssdóttir. 2-nested simulation is not finitely equationally axiomatizable. In A. Ferreira and H. Reichel, eds, *Proceedings 18th Symposium on Theoretical Aspects of Computer Science (STACS'2001)*, Dresden, LNCS 2010, pp. 39–50. Springer-Verlag, 2001.
2. J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Ready-trace semantics for concrete process algebra with the priority operator. *The Computer Journal*, 30(6):498–506, 1987.
3. B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.
4. R.J. van Glabbeek. The linear time – branching time spectrum I. The semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, eds, *Handbook of Process Algebra*, pp. 3–99. Elsevier, 2001.

5. J.F. Groote. A new strategy for proving ω -completeness with applications in process algebra. In J.C.M. Baeten and J.W. Klop, eds., *Proceedings 1st Conference on Concurrency Theory (CONCUR'90)*, Amsterdam, LNCS 458, pp. 314–331. Springer-Verlag, 1990.
6. J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.
7. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
8. D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, ed., *Proceedings 5th GI (Gesellschaft für Informatik) Conference*, Karlsruhe, LNCS 104, pp. 167–183. Springer-Verlag, 1981.
9. G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.
10. I.C.C. Phillips. Refusal testing. *Theoretical Computer Science*, 50(3):241–284, 1987.
11. A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In W. Brauer, ed., *Proceedings 12th Colloquium on Automata, Languages and Programming (ICALP'85)*, Nafplion, LNCS 194, pp. 15–32. Springer-Verlag, 1985.

Resource Access and Mobility Control with Dynamic Privileges Acquisition^{*}

Daniele Gorla and Rosario Pugliese

Dipartimento di Sistemi e Informatica, Università di Firenze
`{gorla,pugliese}@dsi.unifi.it`

Abstract. μ KLAIM is a process language that permits programming distributed systems made up of several mobile components interacting through multiple distributed tuple spaces. We present the language and a type system for controlling the activities, e.g. access to resources and mobility, of the processes in a net. By dealing with privileges acquisition, the type system enables dynamic variations of security policies. We exploit a combination of static and dynamic type checking, and of in-lined reference monitoring, to guarantee absence of run-time errors due to lack of privileges and state two type soundness results: one involves whole nets, the other is relative to subnets of larger nets.

1 Introduction

Process mobility is a fundamental aspect of global computing; however it gives rise to a lot of relevant security problems. Recently, a number of languages for mobile processes have been designed that come equipped with security mechanisms (at compilation and/or at run-time) based on, e.g., type systems, control and data flow analysis and proof carrying code.

Our starting point is KLAIM [9], an experimental language specifically designed to program distributed systems made up of several mobile components interacting through multiple distributed tuple spaces, and its capability-based type system [10] for controlling access to resources and mobility of processes. KLAIM has been implemented [2] by exploiting Java and has proved to be suitable for programming a wide range of distributed applications with agents and code mobility. In KLAIM, the network infrastructure is clearly distinguishable from user processes and explicitly modelled, which we believe gives a proper description of the computer systems we are interested to. KLAIM communication mechanism rests on an extension of the basic Linda coordination model [13] with multiple distributed tuple spaces. General evidence of the success gained by the tuple space paradigm is given by the many tuple space based run-time systems, both from industries, e.g. SUN JavaSpaces [1] and IBM TSpaces [22], and from universities, e.g. PageSpace [8], WCL [21], Lime [19] and TuCSon [18].

^{*} Work partially supported by EU FET - Global Computing initiative, project MIKADO IST-2001-32222, and by MIUR project NAPOLI. The funding bodies are not responsible for any use that might be made of the results presented here.

KLAIM programming paradigm enjoys a number of properties, such as time uncoupling, destination uncoupling, space uncoupling, modularity, scalability and flexibility, that make the language appealing for open distributed systems and network computing environments (see, e.g., [11,14]), where, in general, connections are not stable and host machines are heterogenous. In conclusion, we think it is worthwhile to investigate the KLAIM paradigm, also because its peculiar aspects about interprocess communication and network modelling distinguish it from the most popular and studied process languages.

The major contribution of this paper is the introduction of a calculus, called μ KLAIM, with process distribution and mobility, and of a relative type system for controlling process activities. μ KLAIM is at the core of KLAIM and has a simpler syntax (without higher-order communication, with only one kind of addresses, without allocation environments, and without parameterized process definitions) and operational semantics. Moreover, it has a cleaner and powerful type system (types only record local information), that enables dynamic modifications of security policies and process privileges, and run-time type checking of programs, or part of them. In fact, static verification is useful in many circumstances since it avoids the use of dynamic mechanisms, thus improving system performances. However, it is hardly sufficient in highly dynamic systems, like e.g. open systems and the Internet, where it could restrict privileges and capabilities more than needed, thus unnecessarily reducing the expressive power (and the capabilities) of mobile processes. To deal with open systems, a certain amount of dynamic checking is needed (e.g. mobile processes should be dynamically checked at run-time when they migrate), also for taking into account that in these environments typing information could be partial, inaccurate or missing. Furthermore, extensive dynamic checking along with mechanisms supporting modifications at run-time of security policies and process privileges turn out to be essential for dealing with pervasive network applications, like e.g. those for e-commerce.

The μ KLAIM type system allows processes to be first partially verified and then executed in a more efficient and flexible way, rather than to run inefficiently because of massive run-time checks. Each network node has its own security policy that affects the evolution of the overall system and, thus, must be taken into account when defining the operational semantics. Types are used to express security policies in terms of capabilities (there is one capability for each process operation), hence they are part of the language for configuring the underlying net architecture. Moreover, types are used to record processes intended operations, but programmers are relieved from typing processes because this task is carried on by a static type inference system.

Because of lack of space, we shall omit from this extended abstract several details and all proofs, and present a version of the calculus where communications exchange tuples with only one field; a thorough presentation can be found in [14].

Table 1. μ KLAIM Syntax

$N ::= l ::_{\Sigma}^{\delta} P$	(single node)	$P ::= \mathbf{nil}$	(null process)
$ N_1 \parallel N_2$	(net composition)	$ a.P$	(action prefixing)
$a ::=$	(process actions)	$ P_1 \mid P_2$	(parallel composition)
$\mathbf{in}(T)@l$		$ A$	(process invocation)
$ \mathbf{read}(T)@l$		$T ::= t \mid !x \mid !u : \pi$	(templates)
$ \mathbf{out}(t)@l$		$t ::= e \mid \ell : \mu$	(tuples)
$ \mathbf{eval}(P)@l$		$e ::= V \mid x \mid \dots$	(expressions)
$ \mathbf{newloc}(u : \delta)$			

2 μ KLAIM Syntax

The syntax of μ KLAIM, given in Table 1, is parameterized with respect to the following syntactic sets, which we assume to be countable and pairwise disjoint: $\mathcal{A}$, of *process identifiers*, ranged over by $A, B, \dots$; $\mathcal{L}$, of *localities*, ranged over by l ; $\mathcal{U}$, of *locality variables*, ranged over by u . We use ℓ to range over $\mathcal{L} \cup \mathcal{U}$, V over basic values, x over value variables, π over sets of *capabilities*, δ over *types*, and μ over *capability specifications*.

The exact syntax of *expressions*, e , is deliberately not specified; we just assume that expressions contain, at least, basic values and variables. *Localities*, l , are the addresses (i.e. network references) of nodes. *Tuples*, t , contain expressions, localities or locality variables. In particular, $\ell : \mu$ points out a capability specification μ that permits dynamically determining the set of capabilities granted along with address ℓ . *Templates*, T , are used to select tuples. In particular, parameters $!x$ or $!u : \pi$ (the set of capabilities π constraints the use of the address dynamically bound to u) are used to bind variables to values.

Processes are the μ KLAIM active computational units and can perform a few basic operations over tuple spaces and nodes: retrieve/place (evaluated) tuples from/into a tuple space, send processes for execution on (possibly remote) nodes, and create new nodes. Processes are built up from the stuck process **nil** and from the basic operations by using action prefixing, parallel composition and process definition. It is assumed that each process identifier A has a *single* defining equation $A \triangleq P$. Of course, process defining equations should migrate along with invoking processes; however, for the sake of simplicity, we do not explicitly model migration of defining equations (that could be implemented like class code migration in [2]) and assume that they are available at any locality of a net.

Variables occurring in process terms can be *bound* by action prefixes **in/read/newloc**. For example, **in**($!u : \pi$) $@l$.. and **newloc**($u : \delta$).. bind u , while **in**($!x$) $@l$.. binds x . In process $a.P$, P is the scope of the binding made by a ; we call *free* the variables in P that are not bound and accordingly define α -*conversion*. In the sequel, we shall assume that bound variables in processes are all distinct and different from the free variables (by possibly applying α -

conversion, this requirement can always be satisfied). Moreover, we shall consider only *closed* processes, i.e. processes without free variables.

Nets are finite collections of nodes where processes and tuple spaces can be allocated. A *node* is a quadruple $l ::_{\Sigma}^{\delta} P$, where locality l is the address of the node, P is the (parallel) process located at l , Σ is the set of process defining equations $\{A_1 \triangleq P_1, \dots, A_n \triangleq P_n\}$ (with $A_i \neq A_j$ if $i \neq j$) that are valid at l , and δ is the type of the node, i.e. the specification of its access control policy. The *tuple space* (*TS*) located at l is part of P because, as we will see in Section 4, evaluated tuples are represented as special processes. In the sequel, we shall omit Σ whenever it plays no role.

We will identify nets which intuitively represent the same net. We therefore define *structural congruence* $\equiv$ to be the smallest congruence relation over nets equating α -convertible nets, stating that ‘ $\parallel$ ’ is commutative and associative and that **nil** is the identity for ‘ $\mid$ ’. If not differently specified, in the sequel we shall only consider *well-formed nets*, i.e. nets where pairwise distinct nodes have different addresses.

Capabilities are elements of set $\{r, i, o, e, n\}$, where each symbol corresponds to the operation whose name begins with it; e.g. r denotes the capability of executing a **read** operation. We use Π , ranged over by π , to denote the set formed by the subsets of $\{r, i, o, e, n\}$.

Types, ranged over by δ , are functions of the form $\delta : \mathcal{L} \cup \mathcal{U} \rightarrow_{\text{fin}} \Pi$, where $\rightarrow_{\text{fin}}$ means that the function maps only a finite subset of its domain to non-empty sets. Notation $[\ell_i \mapsto \pi_i]_{\ell_i \in D}$ stands for the type δ such that $\delta(\ell)$ is π_i if $\ell = \ell_i \in D$ and is $\emptyset$ otherwise. The *extension* of δ_1 with δ_2 , written $\delta_1[\delta_2]$, is the type δ' such that $\delta'(\ell) = \delta_1(\ell) \cup \delta_2(\ell)$ for each $\ell \in \mathcal{L} \cup \mathcal{U}$.

Capability specifications, ranged over by μ , are partial functions with finite non-empty domain of the form $\mu : \mathcal{L} \cup \mathcal{U} \rightarrow \Pi \cup \overline{\Pi}$, where $\overline{\Pi} \triangleq \{\bar{\pi} : \pi \in \Pi\}$. For capability specifications, we adopt a notation similar to that used for types, but now $[\ell_i \mapsto p_i]_{\ell_i \in D}$ (where $p_i \in \Pi \cup \overline{\Pi}$) stands for the capability specification μ such that $\text{dom}(\mu) = D$ and $\mu(\ell_i) = p_i$. Capability specifications are used, mainly in **out** operations, to identify sets of capabilities depending on the type at run-time of the node where processes run. In fact, when a process P running, say, at l wants to output a location l' along with some privileges, it is important to guarantee that P cannot grant larger privileges over l' than those owned by l . Since, in general, the latter can be determined only at run-time (because they depend on the privileges acquired by l over l' during the computation), capability specifications provide a way to statically express this fact.

3 A Capability-Based Type System

We start introducing a subtyping relation, $\preceq$. It relies on an ordering over sets of capabilities stating that, if $\pi_1 \sqsubseteq_{\Pi} \pi_2$, then π_1 enables at least the actions enabled by π_2 . The type theory we develop is parametric with respect to the used capability ordering; here, for the sake of simplicity, we let $\sqsubseteq_{\Pi}$ to be the reverse subset inclusion. Now, we define $\preceq$ by letting $\delta_1 \preceq \delta_2$ whenever $\delta_2(\ell) \sqsubseteq_{\Pi} \delta_1(\ell)$

for each $\ell \in \mathcal{L} \cup \mathcal{U}$ (which is the standard preorder over functions). Thus, if $\delta_1 \preceq \delta_2$, then δ_1 is less permissive than δ_2 .

Let us now present the static inference system. Informally, for each node, say $l ::_{\Sigma}^{\delta} P$, of a net, the inference system checks that all process identifiers occurring in P are defined in Σ and determines whether the actions that P intends to perform when running at l are enabled by the access policy δ or not. For example, capability e can be used to control process mobility: P can migrate to l' only if $[l' \mapsto \{e\}]$ is a subtype of δ . However, because l can dynamically acquire privileges when P performs **in/read** actions, some actions that can be permissible at run-time could be statically illegal. For this reason, if P intends to perform an action not allowed by δ , the static inference system cannot reject the process since the capability necessary to perform the action could in principle be dynamically acquired by l . In such cases, the inference system simply *marks* the action to require its dynamic checking. The marking mechanism never applies to actions whose targets are locality variables bound by **in/read**, because such actions can be statically checked, thus alleviating the burden of dynamic checking and improving system performance. In fact, according to the syntax, whenever a locality variable u is bound by an action **in/read**, u is annotated with a set of capabilities π that specifies the operations that the continuation process is allowed to perform by using u as the target address.

We therefore extend the μ KLAIM syntax to include marked actions, where a marked action is a normal μ KLAIM action which is underlined to require a dynamic checking of the corresponding capability. Formally, we extend the syntactic category of processes as $P ::= \dots \mid \underline{a}.P$. We will write $\underline{P}$ ($\underline{N}$, resp.) to emphasize that process P (net N , resp.) may contain marked actions.

A *type context* Γ is a type. To update a type context with the type annotations specified within a template, we use the auxiliary function *upd* that behaves like the identity function for all templates but for those binding locality variables. In this case, we have $\text{upd}(\Gamma, !u : \pi) = \Gamma[u \mapsto \pi]$. Hence, if T is a tuple, then $\text{upd}(\Gamma, T) = \Gamma$. To have more compact inference rules for judgments, we found it convenient to extend function *upd* to encompass the case that the second argument is a process and let $\text{upd}(\Gamma, P) = \Gamma$.

Type judgments for processes take the form $\Gamma \vdash_{\Sigma}^l P \triangleright \underline{P}$. In Γ , the bindings from localities to non-empty sets implement the access policy of the node with address l , while the bindings from locality variables to non-empty sets record the type annotations for the variables that are free (i.e. have been freed) in P . Intuitively, the judgment $\Gamma \vdash_{\Sigma}^l P \triangleright \underline{P}$ states that, within the context Γ , when P is located at l , the unmarked actions in $\underline{P}$ are admissible w.r.t. Γ and all process identifiers occurring in P are defined in Σ .

Type judgments are inferred by using the rules in Table 2. Given an action a , we use $\text{arg}(a)$ to denote its argument, $\text{tgt}(a)$ its target location and $\text{cap}(a)$ the capability corresponding to a . Moreover, we mark actions by using function

$$\text{mark}_{\Gamma}(a) = \begin{cases} a & \text{if } \Gamma(\text{tgt}(a)) \sqsubseteq_{\pi} \{\text{cap}(a)\} \\ \underline{a} & \text{if } \Gamma(\text{tgt}(a)) \not\sqsubseteq_{\pi} \{\text{cap}(a)\} \text{ and } \text{tgt}(a) \in \mathcal{L} \end{cases}$$

Table 2. μKLAIM Type Inference Rules

(1) $\Gamma \vdash_{\Sigma} \mathbf{nil} \triangleright \mathbf{nil}$	(2) $\frac{\Gamma \vdash_{\Sigma} P \triangleright \underline{P} \quad \Gamma \vdash_{\Sigma} Q \triangleright \underline{Q}}{\Gamma \vdash_{\Sigma} P \mid Q \triangleright \underline{P} \mid \underline{Q}}$
(3) $\frac{\Sigma = \Sigma' \cup \{A \triangleq P\}}{\Gamma \vdash_{\Sigma} A \triangleright A}$	(4) $\frac{\text{cap}(a) \neq n \quad \text{upd}(\Gamma, \text{arg}(a)) \vdash_{\Sigma} P \triangleright \underline{P}}{\Gamma \vdash_{\Sigma} a.P \triangleright \text{mark}_{\Gamma}(a).\underline{P}}$
(5) $\frac{\Gamma(l) \sqsubseteq_{\Pi} \{n\} \quad \Gamma[u \mapsto (\Gamma(l) - \{n\})] \vdash_{\Sigma} P \triangleright \underline{P}}{\Gamma \vdash_{\Sigma} \mathbf{newloc}(u : \delta).P \triangleright \mathbf{newloc}(u : \delta).\underline{P}}$	

where $\neg_{\Pi}$ denotes the negation of $\sqsubseteq_{\Pi}$. Condition $\text{tgt}(a) \in \mathcal{L}$ distinguishes actions using localities as target from those using variables, marking the formers and rejecting the latters (as previously explained). The rules in Table 2 should be quite explicative, we only remark a few points. Rule (3) says that a process identifier always successfully passes the static type checking provided that it is defined in Σ . Rule (4) deals with action prefixing. Notice that, in case of action **eval**, the argument process is not statically checked because the locality where the process will be sent for execution, and hence the access policy against which the process has to be checked, cannot be, in general, statically known. Action **newloc** is dealt with differently from the other actions by rule (5) and is always statically checked (i.e. it is never marked). Indeed, **newloc** is always performed locally and the corresponding capability cannot be dynamically acquired. Finally, notice that the creating node owns over the created one all the privileges it owns on itself (except, obviously, for the n capability).

Definition 1. A net is well-typed if for each node $l ::_{\Sigma}^{\delta} P$, with $\Sigma = \{A_1 \triangleq P_1, \dots, A_n \triangleq P_n\}$, there exist $P', P'_1, \dots, P'_n$ such that $\delta \vdash_{\Sigma}^{\delta} P \triangleright P'$ and $\delta \vdash_{\Sigma}^{\delta} P_i \triangleright P'_i$, for each $i \in \{1, \dots, n\}$.

4 μKLAIM Operational Semantics

An important ingredient we need for defining the operational semantics is a way to represent evaluated tuples and TSs. Like in [9], we model tuples as processes. To this aim, we extend the μKLAIM syntax with processes of the form $\langle et \rangle$ (et stands for *evaluated tuple*), that similarly to process **nil** perform no action (and, thus, need no capability). Well-typedness of these auxiliary processes is stated by the axiom

$$(\star) \quad \Gamma \vdash_{\Sigma}^{\delta} \langle et \rangle \triangleright \langle et \rangle$$

that is added to the rules in Table 2.

Only evaluated tuples can be added to a TS and, similarly, templates must be evaluated before being used to retrieve tuples. Hence we define the *tuple/template evaluation* function $\mathcal{T}[\![\cdot]\!]_{\delta}$ as the identity, except for

$$\mathcal{T}[\![e]\!]_{\delta} = \mathcal{E}[\![e]\!] \quad \mathcal{T}[\![l : \mu]\!]_{\delta} = l : [\![\mu]\!]_{\delta(l) - \{n\}}$$

Table 3. Capability Specifications Evaluation Function

$\llbracket [l' \mapsto \pi'] \rrbracket_\pi = [l' \mapsto \pi \cap \pi']$
$\llbracket [l' \mapsto \bar{\pi}'] \rrbracket_\pi = [l' \mapsto (\pi - \pi')]$
$\llbracket \mu_1[\mu_2] \rrbracket_\pi = (\llbracket \mu_1 \rrbracket_\pi)(\llbracket \mu_2 \rrbracket_\pi)$

where function $\mathcal{E}\llbracket \cdot \rrbracket$ evaluates expressions (thus it depends on the kind of allowed expressions and, hence, is left unspecified). $\mathcal{T}\llbracket \cdot \rrbracket_\delta$ takes as a parameter the type (i.e. access policy specification) of the node where the evaluation will take place and accordingly evaluates the contained capability specifications by using function $\llbracket \cdot \rrbracket_\pi$ (defined by the rules in Table 3). The latter is parameterized with respect to the set of capabilities owned by the node where the evaluation takes place over the locality which the capability specification being interpreted is associated to. Notice that, since actions **newloc** are always performed locally, the corresponding capability n is never transmitted. For this reason, the parameter of the interpretation function for capability specification does never contain n . The first rule ensures that no more privileges over a given l' than those owned by l are passed, while the second rule replaces $\bar{\pi}'$ with the complement of π' with respect to π , the set of capabilities used as a parameter of the evaluation function.

The *matching* function $match_l^\delta$, used to select evaluated tuples from a TS according to evaluated templates, is defined by the rules in Table 4. Function $match_l^\delta$ is parameterized with the locality l and the security policy δ of the node where it is invoked. A successful matching returns a type, used to extend the type of the node executing the matching with the capabilities granted by the (producer of the) tuple, and a substitution, used to assign values to variables in the (continuation of the) process invoking the matching. The first two rules say that two values match only if identical and that a value parameter match any value. Rule (3) requires that, for a matching to take place, the locality of the node where the **read/in** is executing must occur in the type specification associated to the locality being accessed. Rule (4) ensures that if a **read/in** executing at l looks for a locality where to perform the actions enabled by π , then, for selecting locality l' , it must hold that the union of the privileges over l' owned by l and of the privileges over l' granted to l by the tuple enables the actions enabled by π . The privileges granted by the tuple are then used to enrich the capabilities of l over l' . Notice that (4) succeeds only if $l \in \text{dom}(\mu)$; this requirement, like that in the premise of rule (3), permits controlling immediate access to tuples (see Section 6).

Finally, the μ KLAIM operational semantics is given by a net reduction relation, $\succrightarrow$, which is the least relation induced by the rules in Table 5. Net reductions are defined over configurations of the form $L \vdash N$, where L is such that $\text{loc}(N) \subseteq L \subset_{fin} \mathcal{L}$ and function $\text{loc}(N)$ returns the set of localities occurring in N . In a configuration $L \vdash N$, L keeps track of the localities in N and is needed to ensure global freshness of new addresses and, thus, to guarantee that well-formedness is preserved along reductions. For the sake of readability, when

Table 4. Matching Rules

(1) $match_i^\delta(V, V) = \langle [], \epsilon \rangle$	(2) $match_i^\delta(!x, V) = \langle [], [V/x] \rangle$
(3) $\frac{l \in dom(\mu_2)}{match_i^\delta(l' : \mu_1, l' : \mu_2) = \langle [], \epsilon \rangle}$	(4) $\frac{\delta(l') \cup \mu(l) \sqsubseteq_{\Pi} \pi}{match_i^\delta(!u : \pi, l' : \mu) = \langle [l' \mapsto \pi], [l'/u] \rangle}$

a reduction does not generate any fresh addresses we write $N \succrightarrow N'$ instead of $L \vdash N \succrightarrow L \vdash N'$; moreover, we also omit the sets of process defining equations from the nodes in N when they are irrelevant.

Let us now comment on the most significant rules in Table 5. Rule (EVAL) says that a process is allowed to migrate only if it successfully passes a type checking against the access policy of the target node. During this preliminary check, some process actions could be marked to be effectively checked when being executed. Rules (IN) and (READ) say that the process performing the operation can proceed only if matching succeeds. In this case, the access policy of the receiving node is enriched with the type returned by the matching mechanism and the substitution returned along with the type is applied to the continuation (and the type annotations therein) of the process performing the operation. In rule (NEW) the set L of localities already in use is exploited to choose a fresh address l' for naming the new node. Notice that, once created, the address of the new node is not known to any other node in the net. Thus, it can be used by the creating process as a sort of *private* resource. In order to enable the creation, the specified access policy δ' , after modification with substitution $[l'/u]$, must be in agreement with the access policy δ of the node executing the operation (δ^{-n} denotes the access policy defined as follows: $\delta^{-n}(l) = \delta(l) - \{n\}$ and $\delta^{-n}(l'') = \delta(l'')$ for every $l'' \neq l$) extended with the ability to perform over l' all the operations allowed locally (a part for **newloc**, of course). This is needed to prevent a malicious node l from forging capabilities by creating a new node with powerful privileges where sending a malicious process that takes advantage of capabilities not owned by l . Hereafter, we write $\overline{\Sigma}$ to denote the set Σ of process defining equations where all marks have been removed. Thus, notation $\delta'[l'/u] \vdash_{\overline{\Sigma}} \Sigma' \triangleright \Sigma'$ means that the set of process defining equations is checkable under the access policy of the new node and returns Σ' . Rule (MARK) says that the in-lined security monitor stops execution whenever the privilege for performing a is missing. Rule (SPLIT) is used to split the parallel processes running at a node thus enabling the application of the rules previously mentioned that, in fact, can only be used when there is only one process running at l .

5 Type Soundness

We start introducing the notion of *executable nets* that, intuitively, are nets already containing all necessary marks (as if they have already passed a static type checking phase). The second clause of the definition accounts for the assumption that all process defining equations are available everywhere (but, in general, are differently marked because checked against different access policies).

Table 5. μ KLAIM Operational Semantics

(OUT)	$\frac{et = \mathcal{T}[t]_\delta}{l ::^\delta \text{out}(t)@l'.P \parallel l' ::^{\delta'} P' \multimap l ::^\delta P \parallel l' ::^{\delta'} P' \langle et \rangle}$
(EVAL)	$\frac{\delta' _{\frac{\Sigma'}{l'}} Q \triangleright Q'}{l ::^\delta \text{eval}(Q)@l'.P \parallel l' ::^{\delta'} P' \multimap l ::^\delta P \parallel l' ::^{\delta'} P' Q'}$
(IN)	$\frac{\text{match}_l^\delta(\mathcal{T}[T]_\delta, et) = \langle \delta'', \sigma \rangle}{l ::^\delta \text{in}(T)@l'.P \parallel l' ::^{\delta'} \langle et \rangle \multimap l ::^{\delta[\delta'']} P\sigma \parallel l' ::^{\delta'} \text{nil}}$
(READ)	$\frac{\text{match}_l^\delta(\mathcal{T}[T]_\delta, et) = \langle \delta'', \sigma \rangle}{l ::^\delta \text{read}(T)@l'.P \parallel l' ::^{\delta'} \langle et \rangle \multimap l ::^{\delta[\delta'']} P\sigma \parallel l' ::^{\delta'} \langle et \rangle}$
(NEW)	$\frac{l' \notin L \quad \delta'[l'/u] \preceq \delta^{-n}[l' \mapsto \delta(l)] \quad \delta'[l'/u] _{\frac{\Sigma'}{l'}} \bar{\Sigma} \triangleright \Sigma'}{L \vdash l ::^\delta \text{newloc}(u : \delta').P \multimap L \cup \{l'\} \vdash l ::^{\delta[l' \mapsto \delta(l)] - \{n\}} P[l'/u] \parallel l' ::^{\delta'[l'/u]} \text{nil}}$
(CALL)	$l ::^\delta A \multimap l ::^\delta P \quad \text{if } \Sigma = \Sigma' \cup \{A \triangleq P\}$
(MARK)	$\frac{l' = \text{tgt}(a) \quad \delta(l') \sqsubseteq_{\pi} \{cap(a)\} \quad l ::^\delta a.P \parallel l' ::^{\delta'} Q \multimap N}{l ::^\delta \underline{a}.P \parallel l' ::^{\delta'} Q \multimap N}$
(SPLIT)	$\frac{L \vdash l ::^\delta P \parallel l ::^\delta Q \parallel N \multimap L' \vdash l ::^{\delta'} P' \parallel l ::^{\delta''} Q' \parallel N'}{L \vdash l ::^\delta P Q \parallel N \multimap L' \vdash l ::^{\delta'[\delta'']} P' Q' \parallel N'}$
(PAR)	$\frac{L \vdash N_1 \multimap L' \vdash N'_1}{L \vdash N_1 \parallel N_2 \multimap L' \vdash N'_1 \parallel N_2}$
(STRUCT)	$\frac{N \equiv N_1 \quad L \vdash N_1 \multimap L' \vdash N_2 \quad N_2 \equiv N'}{L \vdash N \multimap L' \vdash N'}$

Definition 2. A net is executable if the following conditions hold:

- (i) for each node $l ::^\delta_\Sigma P$, with $\Sigma = \{A_1 \triangleq P_1, \dots, A_n \triangleq P_n\}$, it holds that $\delta |_{\frac{\Sigma}{l}} P \triangleright P$ and $\delta |_{\frac{\Sigma}{l}} P_i \triangleright P_i$, for each $i \in \{1, \dots, n\}$,
- (ii) for any pair of nodes $l ::^\delta_\Sigma P$ and $l' ::^{\delta'}_{\Sigma'} P'$, it holds that $\bar{\Sigma} = \bar{\Sigma}'$,

where for inferring the type judgements, in addition to the rules in Table 2 and to axiom $(\star)$ for $\langle et \rangle$, one can also use the rule

$$(\star\star) \quad \frac{\text{upd}(\Gamma, \arg(a)) |_{\frac{\Sigma}{l}} P \triangleright \underline{P}}{\Gamma |_{\frac{\Sigma}{l}} \underline{a}.P \triangleright \underline{a}.P}$$

that allows a process to already contain marked actions.

Notice that executable nets are well-typed. Our main results will be stated in terms of executable nets; indeed, due to the dynamic acquisition of privileges, well-formed nets that are statically deemed well-typed can still give rise to run-time errors. However, by marking those actions that should be checked at run-time, well-typed (and well-formed) nets can be transformed into executable nets that, instead, cannot give rise to run-time errors (see Corollary 1).

It can be easily seen that the property of being executable is preserved by structural congruence. The following theorem states that it is also preserved by the reduction relation.

Theorem 1 (Subject Reduction). *If N is executable and $\text{loc}(N) \vdash N \multimap L' \vdash N'$ then N' is executable and $\text{loc}(N') \subseteq L'$.*

Now, we introduce the notion of *run-time error*, defined in terms of predicate $N \uparrow l$ that holds true when, within N , a process P running at node $l ::_{\delta}^{\Sigma}$ – attempts to perform an action that is not allowed by δ or invokes a process that is not in Σ . The key rules are

$$\frac{\delta(\text{tgt}(a)) \not\sqsubseteq_{\Pi} \{\text{cap}(a)\}}{l ::_{\Sigma}^{\delta} a.P \uparrow l} \qquad \frac{\not\exists \Sigma' : \Sigma = \Sigma' \cup \{A \triangleq P\}}{l ::_{\Sigma}^{\delta} A \uparrow l}$$

We can now state type safety, i.e. that executable nets do not give rise to run-time errors.

Theorem 2 (Type Safety). *If N is executable then $N \uparrow l$ for no $l \in \text{loc}(N)$.*

By combining together Theorem 1 and 2, and by denoting with $\multimap^*$ the reflexive and transitive closure of $\multimap$, we obtain the following result.

Corollary 1 (Global Type Soundness). *If N is executable and $\text{loc}(N) \vdash N \multimap^* L' \vdash N'$ then $N' \uparrow l$ for no $l \in \text{loc}(N')$.*

Type soundness is one of the main goal of any type system. However, in our framework it is formulated in terms of a property requiring the typing of whole nets. While this could be acceptable for LANs, where the number of hosts usually is relatively small, it is unreasonable for WANs, where in general hosts are under the control of different authorities. When dealing with larger nets, it is certainly more realistic to reason in terms of parts of the whole net. Hence, we put forward a more *local* formulation of our main result. To this aim, we define the *restriction* of a net N to a set of localities S , written N_S , as the subnet obtained from N by deleting all nodes whose addresses are not in S . The wanted local type soundness result can be formulated as follows.

Theorem 3 (Local Type Soundness). *Let N be a net and $S \subseteq \text{loc}(N)$. If N_S is executable and $\text{loc}(N) \vdash N \multimap^* L' \vdash N'$ then for no $l \in S$ it holds that $N' \uparrow l$.*

6 Example: Subscribing Online Publications

By means of a simple example, here we show the μKLAIM programming style and illustrate how to exploit its type system. For programming convenience, we will use the full version of the calculus [14], assume integers and strings to be basic values of the language, and omit trailing occurrences of process **nil** and the process defining equations.

Suppose that a user U wants to subscribe a ‘licence’ to enable accessing on-line publications by a given publisher P . To model this scenario we use three localities, l_U , l_P and l_C , respectively associated to U , P and to the repository containing P ’s on-line accessible publications. First of all, U sends a subscription request to P including its address (together with an ‘out’ capability) and credit card number; then, U waits for a tuple that will grant it the ‘read’ privilege needed to access P ’s publications and proceeds with the rest of its activity. The behaviour described so far is implemented by the process

$$U \triangleq \mathbf{out}(\text{“Subscr”}, l_U : [l_P \mapsto \{o\}], CrCard) @ l_P. \mathbf{in}(\text{“Acc”}, l_C : \{r\}) @ l_U. R$$

where process R may contain operations like $\mathbf{read}(\dots) @ l_C$. P , once it has received the subscription request and checked (by possibly using a third party authority) the validity of the payment information, gives U a ‘read’ capability over l_C . P ’s behaviour is modelled by the following process.

$$P \triangleq \mathbf{in}(\text{“Subscr”}, !x : \{o\}, !y) @ l_P. \text{check credit card } y \text{ of } x \text{ and require the payment.} \\ \mathbf{out}(\text{“Acc”}, l_C : [x \mapsto \{r\}]) @ x \mid P$$

For processes U and P to behave in the expected way, the underlying net architecture, namely distribution of processes and security policies, must be appropriately configured. A suitable net is:

$$l_U :: [l_U \mapsto \{o, i, r, e, n\}, l_P \mapsto \{o\}] \underline{U} \parallel l_P :: [l_P \mapsto \{o, i, r, e, n\}, l_C \mapsto \{o, i, r\}] P \parallel \\ l_C :: [] \langle paper1 \rangle \mid \langle paper2 \rangle \mid \dots$$

where we have intentionally used $\underline{U}$ to emphasize the fact that the static type checking might have marked some actions occurring in U , e.g. the $\mathbf{read}(\dots) @ l_C$ actions in R . Upon completion of the protocol, the net will be

$$l_U :: [l_U \mapsto \{o, i, r, e, n\}, l_P \mapsto \{o\}, l_C \mapsto \{r\}] \underline{R} \parallel l_P :: [l_P \mapsto \{o, i, r, e, n\}, l_C \mapsto \{o, i, r\}, l_U \mapsto \{o\}] P \parallel \\ l_C :: [] \langle paper1 \rangle \mid \langle paper2 \rangle \mid \dots$$

Notice that knowledge of address l_C is not enough for reading papers: the ‘read’ capability is needed. Indeed, security in the μ KLAIM framework does not rely on name knowledge but on security policies. Moreover, once the ‘read’ capability over l_C has been acquired, all processes eventually spawned at l_U can access P ’s on-line publications. In other terms, U obtains a sort of ‘site licence’ valid for all processes running at l_U . This is different from [10], where, by using the same protocol, U would have obtained a sort of ‘individual licence’. Notice also that the licence passed by P to U can be used only at l_U since the capability specification associated to l_C only grants l_U privilege r over l_C . Finally, no denial-of-service attack could be mounted through the access of tuple $\langle \text{“Acc”}, l_C : [l_U \mapsto \{r\}] \rangle$ located at l_U by processes running at sites of the network different from those explicitly mentioned because only processes running at l_U can retrieve the tuple (see rules (3) and (4) in Table 4).

Variants. We now touch upon a few variants (thoroughly presented in [14]) of the μ KLAIM framework and use the example for motivating their introduction. The variants differ in simple technical details and, mainly, in the burden charged to the static inference.

In real situations, a (mobile) process could dynamically acquire some privileges and, from time to time, decide whether it wants to keep them for itself or to share them with other processes running at the same node. In our example, U might just buy an ‘individual licence’. The μ KLAIM framework can smoothly fit for this feature, by associating privileges also to processes and letting them decide whether an acquisition must enrich their hosting node or themselves. Moreover, the subscription could have an expiration date, e.g., it could be an annual subscription. Timing information can easily be accommodated in the μ KLAIM framework by simply assigning privileges a validity duration and by updating these information for taking into account time passing. ‘Acquisition of privileges’ can be thought of as ‘purchase of services/goods’; hence it is natural that a process will lose the acquired privilege once it uses the service or passes the good to another process. In our running example, this corresponds to purchasing the right of accessing P ’s publications a given number of times. A simple modification of the μ KLAIM framework, for taking into account multiplicities of privileges and their consumption (due, e.g., to execution of the corresponding action or to cession of the privilege to another process), can permit to deal with this new scenario. Finally, the granter of a privilege could decide to revoke the privilege previously granted. In our example, P could prohibit U from accessing its publications because of, e.g., a misbehaviour or expiry of the subscription time (in fact, this is a way of managing expiration dates without assigning privileges a validity duration). Again, by annotating privileges dynamically acquired with the granter identity and enabling processes to use a new ‘revoke’ operation, the μ KLAIM framework can be extended to also manage privileges revocation.

7 Related Work

By now, there is a lot of work on type systems for security in calculi with process distribution and mobility; however, to the best of our knowledge, the type system we have presented in this paper is the first one that permits dynamic modification of security policies. We conclude by touching upon more strictly related work.

The research line closest to ours is that on the $D\pi$ -calculus [16], a distributed version of the π -calculus equipped with a type system to control privileges of mobile processes over located communication channels. [15,20] present two improved type systems for the $D\pi$ -calculus that, by relying on both local type information and on dynamic checking of incoming processes, permit establishing well-typedness of part of a network (like our local type soundness result). Like μ KLAIM, the $D\pi$ -calculus relies on a flat network architecture; however, differently from μ KLAIM, the network infrastructure is not independent of the processes running over it and communication is local and channel-based. Moreover, node types describe permissions to use local channels. This is in sharp contrast with μ KLAIM types that aim at controlling the remote operations that a network node can perform over the other network nodes.

[23] presents $D\pi\lambda$, a process calculus resulting from the integration of the call-by-value λ -calculus and the π -calculus, together with primitives for process distribution and remote process creation. Apart from the higher order and channel-based communication, the main difference with μ KLAIM is that $D\pi\lambda$ localities are not explicitly referrable by processes and just used to express process distribution. In [24], a fine-grained type system for $D\pi\lambda$ is defined that permits controlling the effect over local channels of transmitted processes parameterized w.r.t. channel names. Processes are assigned fine-grained types that, like interfaces, record the channels to which processes have access together with the corresponding capabilities, and parameterized processes are assigned dependent functional types that abstract from channel names and types. This use of types is akin to μ KLAIM one, though the differences between the underlying languages still remain.

Finally, we want to mention some proposals for the Mobile Ambients calculus and its variants, albeit their network models and mobility mechanisms are very different from those of μ KLAIM. Among the type systems more strictly related to security we recall those disciplining the types of the values exchanged in communications [5,4], those for controlling ambients mobility and ability to be opened [6,17,12,7] and that for controlling resources access via policies for mandatory access control based on ambients security levels [3].

Acknowledgements. We thank the anonymous referees for their useful comments.

References

1. K. Arnold, E. Freeman, and S. Hupfer. *JavaSpaces Principles, Patterns and Practice*. Addison-Wesley, 1999.
2. L. Bettini, R. De Nicola, and R. Pugliese. KLAVA: a Java Package for Distributed and Mobile Applications. *Software – Practice and Experience*, 32:1365–1394, 2002.
3. M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *Proceedings of CONCUR 2001*, number 2154 in LNCS, pages 102–120. Springer, 2001.
4. M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *Proceedings of TACS 2001*, number 2215 in LNCS, pages 38–63. Springer, 2001.
5. L. Cardelli and A. D. Gordon. Types for mobile ambients. In *Proceedings of POPL ’99*, pages 79–92. ACM, 1999.
6. L. Cardelli, G. Ghelli, and A. D. Gordon. Types for the ambient calculus. *Information and Computation*, 177:160–194, 2002.
7. G. Castagna, G. Ghelli, and F. Z. Nardelli. Typing mobility in the seal calculus. In *Proceedings of CONCUR 2001*, number 2154 in LNCS, pages 82–101. Springer, 2001.
8. P. Ciancarini, R. Tolksdorf, F. Vitali, D. Rossi, and A. Knoche. Coordinating multi-agent applications on the WWW: A reference architecture. *IEEE TSE*, 24(5):362–366, 1998.
9. R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

10. R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for Access Control. *Theoretical Computer Science*, 240(1):215–254, 2000.
11. D. Deugo. Choosing a Mobile Agent Messaging Model. In *Proceedings of ISADS 2001*, pages 278–286. IEEE, 2001.
12. M. Dezani-Ciancaglini and I. Salvo. Security types for mobile safe ambients. In *Proceedings of ASIAN'00*, volume 1961 of *LNCS*, pages 215–236, Springer, 2000.
13. D. Gelernter. Generative Communication in Linda. *Transactions on Programming Languages and Systems*, 7(1):80–112. ACM, 1985.
14. D. Gorla and R. Pugliese. Resource access and mobility control with dynamic privileges acquisition. Research report, Dipartimento di Sistemi e Informatica, Università di Firenze, 2003. Available at <http://rap.dsi.unifi.it/~pugliese/DOWNLOAD/muklaim-full.pdf>.
15. M. Hennessy and J. Riely. Type-Safe Execution of Mobile Agents in Anonymous Networks. In *Secure Internet Programming*, volume 1603 of *LNCS*, pages 95–115. Springer, 1999.
16. M. Hennessy and J. Riely. Resource Access Control in Systems of Mobile Agents. *Information and Computation*, 173:82–120, 2002.
17. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings of POPL'00*, pages 352–364. ACM, 2000.
18. A. Omicini and F. Zambonelli. Coordination for internet application development. *Autonomous Agents and Multi-agent Systems*, 2(3):251–269, 1999.
19. G. Picco, A. Murphy, and G.-C. Roman. LIME: Linda Meets Mobility. In D. Garlan, editor, *Proc. of the 21st Int. Conference on Software Engineering (ICSE'99)*, pages 368–377. ACM Press, 1999.
20. J. Riely and M. Hennessy. Trust and partial typing in open systems of mobile agents. In *Proc. of POPL'99*, pages 93–104. Full version to appear in *Journal of Automated Reasoning*, 2003.
21. A. Rowstron. WCL: A web co-ordination language. *World Wide Web Journal*, 1(3):167–179, 1998.
22. P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. TSpaces. *IBM Systems Journal*, 37(3):454–474, 1998.
23. N. Yoshida and M. Hennessy. Subtyping and locality in distributed higher order processes. In *Proceedings of CONCUR'99*, volume 1664 of *LNCS*, pages 557–572. Springer, 1999.
24. N. Yoshida and M. Hennessy. Assigning types to processes. In *Proceedings of LICS'00*, pages 334–348. Full version appear in *Information and Computation*, 173:82–120, 2002.

Replication vs. Recursive Definitions in Channel Based 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. We investigate the expressive power of two alternative approaches used to express infinite behaviours in process calculi, namely, replication and recursive definitions. These two approaches are equivalent in the full π -calculus, while there is a common agreement that this is not the case when name mobility is not allowed (as in the case of CCS), even if no formal discriminating results have been proved so far.

We consider a hierarchy of calculi, previously proposed by Sangiorgi, that spans from a fragment of CCS (named “the core of CCS”) to the π -calculus with internal mobility. We prove the following discrimination result between replication and recursive definitions: the termination of processes is an undecidable property in the core of CCS, provided that recursive process definitions are allowed, while termination turns out to be decidable when only replication is permitted. On the other hand, this discrimination result does not hold any longer when we move to the next calculus in the hierarchy, which supports a very limited form of name mobility.

1 Introduction

The π -calculus has been designed more than ten years ago for specifying mobile systems and reasoning formally about their behaviour. Rather than an established, definitive theory it can be considered as a “workshop to express ideas about mobility and interaction” [Mil01], indeed many variants, sub-calculi and extensions appeared since its original definition. Given such a rich variety it is important to formally compare the existing π -calculus “dialects” in order to understand precisely their relative expressive power. Several notions of expressive power are meaningful in this context: The classical notion based on the ability to express recursive functions can be further refined by considering, for example, compositionality properties for the encoding of a language into another or the ability to express some patterns of behaviours (typically connected to mobility).

An important aspect which, in general, may significantly affect the expressive power of a language is the mechanism adopted for extending finite processes in order to express infinite behaviours. In the π -calculus theory both replication and recursive process definitions are used: 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. On the other hand, by using recursively defined process constants one can obtain an “in depth” infinite behaviour, since process copies in this case can be nested at an arbitrary depth by using constant application. It is well known that these two mechanisms are equivalent (for any reasonable notion of expressive power) for full π -calculus [MPW92], as the ability to communicate free names together with replication and restriction allows to simulate process constants (replication can easily be simulated by recursive definitions, provided one admits enough constants). On the other hand, there is a common agreement on the fact that recursion cannot be replaced by replication when name mobility is not allowed (as in the case of CCS), even though this result has not been formally proved so far.

In this paper we compare replication and explicit recursion in the context of the π -calculus with internal mobility. This is a sub-calculus obtained essentially by disallowing the free output construct, thus only the output of private names is allowed. As argued in [San96] and formally proved in [Bor98], internal mobility allows one to retain the expressiveness of the π -calculus without some of the semantic complications arising for the full language. In particular, the λ -calculus and several agent-passing calculi can be encoded in the π -calculus with internal mobility and recursive agent definitions [San96], denoted by πI_D in this paper. On the other hand, [San96] shows that the π -calculus with internal mobility and repetition (fragment we refer to with πI_1) is strictly less expressive than πI_D in the sense that, by using the type system inherited from the π -calculus, recursive types are not needed in πI_1 . This has the following relevant consequence in terms of mobility: in πI_D one can create a dependency chain of unbounded length among names, where a name depends on another if the latter carries the former (e.g. in $x(y)$ name y depends on x). This is not possible in πI_1 , where for each process P there exists a finite limit n to the length of dependency chains that can be created in P . Furthermore, [San96] argues that under certain conditions the λ -calculus cannot be encoded in πI_1 and leaves open the question if λ -calculus can be encoded in πI_1 at all.

We answer positively to this question by providing a deterministic encoding in πI_1 of the Random Access Machines (RAMs), which provide another Turing powerful computing formalism. Actually, we show that the fragment πI_1^2 is sufficient to this purpose, where following the mobility hierarchy defined in [San96] the calculus πI_1^n with $n > 0$ includes only those processes which can be typed with types of order n or less than n . In terms of mobility, processes in πI_1^n are those which have dependency chains among names of length at most n . In particular, πI_1^1 does not allow mobility at all and is the core of CCS.

We also prove that πI_1^1 is strictly less expressive than πI_1^2 , since we prove that termination is decidable for a process in πI_1^1 . This provides a formal proof for the folk theorem mentioned before, as RAMs can be deterministically encoded in πI_1^1 when recursive process definitions are allowed.

The remaining of this paper is organized as follows. Section 2 reports the definitions of the syntax and the operational semantics of the process calculi.

In Section 3 and 4 we present the modeling of RAMs and the proof of the decidability result, respectively. Finally, in Section 5 we discuss related work and we report some conclusive remark. Due to space limits, we do not include the proofs of our Theorems: they can be found in [BGZ03].

2 The Calculi

In this section we recall the (variants of the) π -calculus with internal mobility [San96] that we consider in this paper. The main difference between the full π -calculus and its fragment with internal mobility is that output prefixes can only send fresh names. More formally, we have that only outputs of the form $(\nu\tilde{y})\bar{x}\tilde{y}.P$ can be used, where $(\nu\tilde{y})$ is a binder for the names in the sequence $\tilde{y}$ and $\bar{x}\tilde{y}$ is the output prefix. For the sake of simplicity, in the π -calculus with internal mobility, $\bar{x}(\tilde{y}).P$ is written instead of $(\nu\tilde{y})\bar{x}\tilde{y}.P$. This notation emphasizes the symmetry between the prefixes for output and input (the latter, as usual, is denoted with $x(\tilde{y})$): both the prefixes, under internal mobility, are binders. In [Bor98] it is proved that the restriction to internal mobility does not reduce the expressive power of the full π -calculus.

We start by introducing a finite fragment of the π -calculus with internal mobility, and then we define two infinite extensions; the first one obtained by adding constants with (possibly) recursive definitions, and the second one obtained by introducing replication.

Definition 1. (finite $\pi\mathbf{I}$) *Let Name, ranged over by $x, y, \dots$, be a denumerable set of names. We use $\tilde{x}, \tilde{y}, \dots$, to denote (possibly empty) sequences of names. The class of finite $\pi\mathbf{I}$ processes is described by the following grammar:*

$$P ::= \sum_{i \in I} \alpha_i.P_i \quad | \quad P|P \quad | \quad (\nu x)P \qquad \alpha ::= \tau \quad | \quad x(\tilde{y}) \quad | \quad \bar{x}(\tilde{y})$$

We assume that all names in $\tilde{y}$ are pairwise different. When $\tilde{y}$ is empty, we omit the surrounding parentheses.

The guarded sum construct $\sum_{i \in I} \alpha_i.P_i$ is used to make choice among the summands $\alpha_i.P_i$: we assume that I is a finite indexing set and, if I is empty, we abbreviate the sum as $\mathbf{0}$. We shall write $\alpha_1.P_1 + \dots + \alpha_n.P_n$ for $\sum_{i \in \{1 \dots n\}} \alpha_i.P_i$. 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 .

The possible prefixes α are the silent action τ , the input action $x(\tilde{y})$ and the output action $\bar{x}(\tilde{y})$. For input and output actions, we write $\bar{\alpha}$ for the complementary of α ; that is, if $\alpha = x(\tilde{y})$ then $\bar{\alpha} = \bar{x}(\tilde{y})$, if $\alpha = \bar{x}(\tilde{y})$ then $\bar{\alpha} = x(\tilde{y})$. We write $P \equiv_A Q$ if P and Q are alpha convertible. We write $fn(P)$, $bn(Q)$ (resp. $fn(\alpha)$, $bn(\alpha)$) for the *free names* and the *bounded names* of P (resp. α). The *names* of P , written $n(P)$ and $n(\alpha)$, are the union of their free and bounded names. We use $cn(\alpha)$ to denote the names carried by an action α , i.e., $cn(x(\tilde{y})) = cn(\bar{x}(\tilde{y})) = \tilde{y}$.

Table 1 contains the set of the transition rules for $\pi\mathbf{I}$.

Table 1. The transition system for finite $\pi\mathbf{I}$ (symmetric rules omitted).

ALPHA :	$\frac{P \equiv_A P' \quad P' \xrightarrow{\alpha} P''}{P \xrightarrow{\alpha} P''}$	PRE :	$\alpha.P \xrightarrow{\alpha} P$
RES :	$\frac{P \xrightarrow{\alpha} P'}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'} \quad x \notin n(\alpha)$	SUM :	$\frac{P_i \xrightarrow{\alpha} P'_i, i \in I}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_i}$
PAR :	$\frac{P \xrightarrow{\alpha} P'}{P Q \xrightarrow{\alpha} P' Q} \quad bn(\alpha) \cap fn(Q) = \emptyset$		
COM :	$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P Q \xrightarrow{\tau} (\nu x_1) \dots (\nu x_n)(P' Q')} \quad \alpha \neq \tau, x_1 \dots x_n = cn(\alpha)$		

Definition 2. ($\pi\mathbf{I}_D$) We assume a set of constants, ranged over by D . The class of $\pi\mathbf{I}_D$ 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 $\tilde{x}$. 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 whose length equals the arity of D . As usual, in the 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 $\tilde{x}$.

Definition 3. ($\pi\mathbf{I}_!$) The class of $\pi\mathbf{I}_!$ processes is defined by adding the production $P ::= !P$ to the grammar of Definition 1.

The transition rules for constant and replication are

$$\frac{P \xrightarrow{\alpha} P'}{D\langle\tilde{x}\rangle \xrightarrow{\alpha} P'} \quad \text{if } D \stackrel{def}{=} (\tilde{y})Q \text{ and } (\tilde{y})Q \equiv_A (\tilde{x})P \qquad \frac{P|!P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

In [San96], a hierarchy of fragments of $\pi\mathbf{I}_!$ is introduced denoted with $\{\pi\mathbf{I}_!^n\}_{n>0}$. These calculi differ in the form of mobility they support. In particular, the maximal length of *dependency chains* among names is considered: a name depends on another one if the latter carries the former (e.g., in $x(y)$ name y depends on x). By $\pi\mathbf{I}_!^n$ we denote the fragment of $\pi\mathbf{I}_!$ which permits dependency chains with a maximal length less or equal to n .

In order to define the calculi $\{\pi\mathbf{I}_!^n\}_{n>0}$, we introduce an auxiliary type system.

Definition 4. (typing system for $\pi\mathbf{I}_!$) Consider the types S having the form $S ::= (\tilde{S})$ where by $\tilde{S}$ we denote a (possibly empty) sequence of types. We do not consider recursive types because they are not necessary for typing $\pi\mathbf{I}_!$ (as

Table 2. The typing rules for the operators of $\pi\mathbf{I}$.

$\frac{\Gamma[x] = (\tilde{S}) \quad \Gamma, \tilde{y} : \tilde{S} \vdash P}{\Gamma \vdash x(\tilde{y}).P, \Gamma \vdash \bar{x}(\tilde{y}).P}$	$\frac{\Gamma \vdash P}{\Gamma \vdash \tau.P}$
$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P Q}$	$\frac{\Gamma \vdash P}{\Gamma \vdash !P}$
$\frac{\Gamma \vdash P_i, i \in I}{\Gamma \vdash \sum_{i \in I} P_i}$	$\frac{\Gamma, x : S \vdash P, \text{ for some } S}{\Gamma \vdash (\nu x)P}$

proved in Lemma 6.9 of [San96]). A typing is a finite set of assignments of types to names: $\Gamma ::= \emptyset \mid \Gamma, x : \tilde{S}$. Names in a typing Γ are always taken to be pairwise distinct; this justifies an abuse of notation whereby Γ is regarded as a finite function from names to types: $\Gamma[x]$ is the type assigned to x . A process P in $\pi\mathbf{I}_I$ is well-typed for Γ if $\Gamma \vdash P$ can be inferred from the rules of Table 2. Observe that $\mathbf{0}$, corresponding to $\sum_{i \in \emptyset} P_i$, is well typed for any Γ .

Definition 5. (calculi $\{\pi\mathbf{I}_I^n\}_{n>0}$) The order of a type S is the maximal level of bracket nesting in the definition of S . For example, type $()$ has order 1 and type $((), (()))$ has order 3. A process $P \in \pi\mathbf{I}_I$ is in $\pi\mathbf{I}_I^n$, $n > 0$, if, for some typing Γ , there is a derivation proof of $\Gamma \vdash P$ in which all types used (including those in Γ) have order n or less than n .

Observe that $\pi\mathbf{I}_I^1$ permits only process synchronization and does not support the possibility to communicate names; it corresponds to the fragment of CCS [Mil89] without relabeling, with guarded choice instead of free choice, and with replication instead of recursive definition. For this reason, this calculus is also called “the core of CCS”.

Definition 6. ($\pi\mathbf{I}_D^1$) A process $P \in \pi\mathbf{I}_D$ is in $\pi\mathbf{I}_D^1$ if $\tilde{y}$ is empty for any $x(\tilde{y})$ and $\bar{x}(\tilde{y})$ in P .

Observe that $\pi\mathbf{I}_D^1$ corresponds to $\pi\mathbf{I}_I^1$, where constants are considered instead of replication.

Given a process Q , its internal runs $Q \longrightarrow Q_1 \longrightarrow \dots$ are given by those transitions $\longrightarrow$ that the process can perform in isolation. As usual, internal transitions $\longrightarrow$ correspond to the transitions labeled with τ , i.e. $P \longrightarrow P'$ iff $P \xrightarrow{\tau} P'$. 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* does not corresponds to *process convergence*: a process converges when it has at least one finite (complete) internal run; while it terminates if all its internal runs are finite.

3 Modeling RAMs in πI_D^1 and πI_I^2

In this section we prove that the calculi πI_D^1 and πI_I^2 are expressive enough for modeling Turing powerful formalisms. This is proved by showing how to model Random Access Machines (RAMs), a well known register based Turing powerful formalism.

As a consequence of the fact that the calculi πI_D^1 and πI_I^2 are expressive enough to model RAMs in a deterministic manner, we have that termination is undecidable in these calculi. Formally, P *terminates* is an undecidable property for both πI_D^1 and πI_I^2 . On the other hand, in the following section, we will prove that this is not the case for the calculus πI_I^1 , for which the termination turns out to be decidable.

A RAM (denoted in the following with R) is a computational model composed of a finite set of registers $r_1, \dots, r_n$, that can hold arbitrary large natural numbers, and by a program composed by indexed instructions $(1 : I_1), \dots, (m : I_m)$, 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 $(i, c_1, \dots, c_n)$ where i is the program counter indicating the next instruction to be executed, and $c_1, \dots, c_n$ are the current contents of the registers $r_1, \dots, r_n$, respectively.

Without loss of generality, we assume that the registers contain the value 0 at the beginning of the computation and that the execution of the program begins with the first instruction $(1 : I_1)$. In other words, the initial configuration is $(1, 0, \dots, 0)$. The computation continues by executing the other 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. More formally, we indicate by $(i, c_1, \dots, c_n) \rightarrow_R (i', c'_1, \dots, c'_n)$ the fact that the configuration of the RAM R changes from $(i, c_1, \dots, c_n)$ to $(i', c'_1, \dots, c'_n)$ after the execution of the i -th instruction. If a configuration with program counter i different from any instruction index is reached, then the computation terminates.

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 .

3.1 Modeling RAMs in πI_D^1

In this subsection, we will exploit only a limited form of constant applications. More precisely, every time a term $D\langle\tilde{x}\rangle$ is used, we assume that in the corresponding defining equation $D \stackrel{def}{=} (\tilde{y})P$ we have that $\tilde{x} = \tilde{y}$. In other words, the *actual* names used in constant applications always correspond to the *formal* names used in constant definitions. This permits us to use a simplified notation, by omitting $\langle\tilde{x}\rangle$ (resp. $(\tilde{y})$) in each constant application $D\langle\tilde{x}\rangle$ (resp. constant

definition $D \stackrel{\text{def}}{=} (\tilde{y})P$). This simplified notation does not introduce ambiguity because $\tilde{x}$ and $\tilde{y}$ exactly correspond to the free names appearing in P .

Let R be a RAM with registers $r_1, \dots, r_n$, and instructions $(1 : I_1), \dots, (m : I_m)$; we model R as described in the following. For each $1 \leq i \leq m$, we model the i -th instruction $(i : I_i)$ of R with a program constant $Inst_i$ defined as follows.

$$\begin{aligned} Inst_i &\stackrel{\text{def}}{=} \overline{inc_j}.Inst_{i+1} && \text{if } I_i = Succ(r_j) \\ Inst_i &\stackrel{\text{def}}{=} \overline{dec_j}.ack.Inst_{i+1} + \overline{zero_j}.Inst_s && \text{if } I_i = DecJump(r_j, s) \end{aligned}$$

In the first case, the process $Inst_i$ simply increments the register r_j (by firing the $\overline{inc_j}$ prefix) and activates the subsequent instruction; in the second case the process $Inst_i$ tries either to decrement or to test whether the register r_j is empty. According to the prefix which is fired ($\overline{dec_j}$ or $\overline{zero_j}$, respectively) the corresponding subsequent instruction is activated ($Inst_{i+1}$ or $Inst_s$, respectively). In the case of decrement, the process waits for an acknowledgement ack before activating the next instruction; this is necessary in order to activate the next instruction only after the actual update of the register.

As far as the modeling of the registers is concerned, we represent each register r_j , which is initially empty, with a constant Z_j . The constant Z_j is defined in terms of two other constants O_j and E_j :

$$\begin{aligned} Z_j &\stackrel{\text{def}}{=} zero_j.Z_j + inc_j.(\nu x)(O_j \mid x.\overline{ack}.Z_j) \\ O_j &\stackrel{\text{def}}{=} dec_j.\bar{x} + inc_j.(\nu y)(E_j \mid y.\overline{ack}.O_j) \\ E_j &\stackrel{\text{def}}{=} dec_j.\bar{y} + inc_j.(\nu x)(O_j \mid x.\overline{ack}.E_j) \end{aligned}$$

The idea behind this modeling of the registers is to exploit a chain of nested restrictions with a length corresponding to the content of the register. More precisely, the term Z_j represents the register when empty, while O_j and E_j model the register when it has an odd or an even content, respectively. Each time the register is incremented, the length of the chain of restrictions augments due to the creation of a new name. Observe that in order to avoid name collisions, the two names x and y are alternatively exploited. The use of two different names requires also the exploitation of the two different constants O_j and E_j .

Definition 7. Let R be a RAM with program instructions $(1 : I_1), \dots, (m : I_m)$ and registers $r_1, \dots, r_n$. Given the initial configuration $(1, 0, \dots, 0)$ of R we define

$$\llbracket (1, 0, \dots, 0) \rrbracket_D = Inst_1 \mid Z_1 \mid \dots \mid Z_n$$

with $Inst_i$ and Z_j defined as above.

As we have already discussed, the computation of the encoding of RAMs proceeds deterministically, and corresponds exactly to the computation of the corresponding RAM; thus the encoding terminates if and only if the considered RAM terminates, as stated by the following theorem.

Theorem 1. Let R be a RAM with program instructions $(1 : I_1), \dots, (m : I_m)$ and registers $r_1, \dots, r_n$. Given the initial configuration $(1, 0, \dots, 0)$ of R we have that R terminates if and only if the process $\llbracket (1, 0, \dots, 0) \rrbracket_D$ terminates.

3.2 Modeling RAMs in $\pi\mathbf{I}_1^2$

The modeling of RAMs we have presented in the previous subsection exploits recursive definitions in two ways: (i) in the definition of the instructions (where an instruction $Inst_i$ may be directly or indirectly defined in terms of itself) and (ii) in the definition of the constants Z_j , O_j , and E_j modeling the registers.

Here we show that we can rewrite the modeling of RAMs in terms of replication, at the price of introducing a limited form of mobility of names, namely, the mobility supported by the calculus $\pi\mathbf{I}_1^2$.

Let R be a RAM with program instructions $(1 : I_1), \dots, (m : I_m)$ and registers $r_1, \dots, r_n$. As far as (i) is concerned, we control the flow of execution of the program instructions simply by representing explicitly the program counter. We use $\overline{p_i}$ in order to indicate that $(i : I_i)$ is the next instruction to execute.

According to this approach, each instruction $(i : I_i)$ is represented by a process which is guarded by an input operation p_i , subsequently performs the corresponding operation on the registers, then waits for an acknowledgement indicating that the operation on the registers have been performed, and finally updates the program counter by producing $\overline{p_{i+1}}$ (or $\overline{p_s}$ in the case of jump).

Formally, the instruction $(i : I_i)$ is modeled by $\llbracket (i : I_i) \rrbracket$ which is a shorthand notation for the following processes.

$$\begin{aligned} \llbracket (i : I_i) \rrbracket & : \ !p_i.\overline{inc_j}.ack.\overline{p_{i+1}} & \text{if } I_i = Succ(r_j) \\ \llbracket (l : I_l) \rrbracket & : \ !p_l.(\overline{dec_j}.ack.\overline{p_{l+1}} + \overline{zero_j}.ack.\overline{p_s}) & \text{if } I_i = DecJump(r_j, s) \end{aligned}$$

In this case the acknowledgement is always necessary because, as it will be clear in the following, the update of the register requires several internal steps.

As far as (ii) is concerned, we use the two channel names z_j and s_j . The name z_j is used to trigger a processes which represents the register r_j when empty, while s_j is used to trigger terms that represent the register when it is not empty. We model each register r_j , when it is empty, with the following process, simply denoted with $\llbracket r_j = 0 \rrbracket$ in the following:

$$\begin{aligned} \llbracket r_j = 0 \rrbracket & : \quad (zero_j.\overline{z_j} + inc_j.\overline{s_j}(x).x.\overline{z_j}) \mid \\ & \quad !z_j.\overline{ack}.(zero_j.\overline{z_j} + inc_j.\overline{s_j}(x).x.\overline{z_j}) \mid \\ & \quad !s_j(x).(\nu z)(\overline{z} \mid !z.\overline{ack}.(dec_j.\overline{x} + inc_j.\overline{s_j}(y).y.\overline{z})) \end{aligned}$$

Also in this case, the idea is to exploit a chain of nested restrictions; however, the chain is obtained here with a different technique. In the previous encoding we have used recursively defined processes; in this case, we exploit replicated processes and the possibility to extend the scope of local names using name mobility. More precisely, we use replication to have an unbounded amount of processes, each one representing a unit inside the register; these processes are activated one after the other each time the register is incremented. The chain of restrictions is as follows: each of these processes has a local name y , and when an increment occurs, the last activated process triggers a new instance of these terms, and passes to the new term its local name y .

Definition 8. Let R be a RAM with program instructions $(1 : I_1), \dots, (m : I_m)$ and registers $r_1, \dots, r_n$. Given the initial configuration $(1, 0, \dots, 0)$ of R we define

$$\llbracket (1, 0, \dots, 0) \rrbracket_1 = \overline{p_1} \llbracket (1 : I_1) \rrbracket \dots \llbracket (m : I_m) \rrbracket \llbracket r_1 = 0 \rrbracket \dots \llbracket r_n = 0 \rrbracket$$

with $\llbracket (i : I_i) \rrbracket$ and $\llbracket r_j = 0 \rrbracket$ as above.

Also in this case the encoding faithfully simulates the computation of the corresponding RAM. Indeed, the Theorem 1 holds also for the encoding $\llbracket \cdot \rrbracket_1$.

4 Decidability of Termination in $\pi\mathbf{I}_1^1$

In both of the RAM encodings presented in the previous section natural numbers are represented by chains of nested restrictions, that are constructed by exploiting either constant definitions or name passing in the calculus with replication. In this section we show that in $\pi\mathbf{I}_1^1$ name passing (at least the limited form of name passing of $\pi\mathbf{I}_1^1$) is really needed to obtain Turing completeness. In fact we prove that termination is decidable for $\pi\mathbf{I}_1^1$ processes.

This result is based on the theory of well-structured transition systems [FS01]; first of all, we provide an alternative semantics for $\pi\mathbf{I}_1^1$ that is equivalent w.r.t. termination 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 termination is decidable for $\pi\mathbf{I}_1^1$ 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.

Definition 9. 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 10. 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 $\text{Succ}(s)$ to denote the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of s .

TS is finitely branching if all $\text{Succ}(s)$ 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 11. 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 $s_2 \leq t_2$.

The following theorem (a special case of a result in [FS01]) will be used to obtain our decidability result.

Theorem 2. *Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with strong compatibility, decidable $\leq$ and computable Succ. The existence of an infinite computation starting from a state $s \in S$ is decidable.*

4.1 A Finitely Branching Transition System for πI_1^1

As the results on well-structured transition systems apply to finitely branching transition systems, first of all we need to define an alternative semantics for πI_1^1 , that is based on a finitely branching transition system and that is equivalent w.r.t. termination to the semantics presented in Section 2.

The existence of infinitely branching processes is due to the rules for alpha conversion and for replication. We define a new semantics by removing the rule for alpha conversion and by reformulating the semantics of replication.

The new transition relation $\overset{\alpha}{\mapsto}$ over πI_1^1 processes is the least relation satisfying all the axioms and rules of Table 1 (where $\overset{\alpha}{\mapsto}$ is substituted for $\overset{\alpha}{\rightarrow}$) but ALPHA, plus the following rules REPL1 and REPL2.

$$\text{REPL1 : } \frac{P \overset{\alpha}{\mapsto} P'}{!P \overset{\alpha}{\mapsto} P' \mid !P} \qquad \text{REPL2 : } \frac{P \overset{\alpha}{\mapsto} P' \quad P \overset{\bar{\alpha}}{\mapsto} P''}{!P \overset{\tau}{\mapsto} P' \mid P'' \mid !P}$$

As done for the standard transition system, we assume that the reductions $\mapsto$ of the new semantics corresponds to the τ -labeled transitions $\overset{\tau}{\mapsto}$. Also for the new semantics, we say that a process P terminates if and only if all its computations are finite, i.e. it cannot give rise to an infinite sequence of reductions $\mapsto$.

Proposition 1. *Let $P \in \pi I_1^1$. Then P terminates according to the semantics $\rightarrow$ iff P terminates according to the new semantics $\mapsto$.*

4.2 Termination Is Decidable in $(\pi I_1^1, \mapsto)$

In this section we equip the transition system $(\pi I_1^1, \mapsto)$ with a preorder on processes which turns out to be a well-quasi-ordering compatible with $\mapsto$. Thus, exploiting the Theorem 2 we show that termination is decidable.

Definition 12. *Let $P \in \pi I_1^1$. With $\text{Deriv}(P)$ we denote the set of processes reachable from P with a sequence of reduction steps:*

$$\text{Deriv}(P) = \{Q \mid P \mapsto^* Q\}$$

To define the wqo on processes we need the following structural congruence, that turns out to be compatible with $\mapsto$.

Definition 13. We define $\equiv$ as the least congruence relation satisfying the following axioms: $P|Q \equiv Q|P$ $P|(Q|R) \equiv (P|Q)|R$ $P|0 \equiv P$

Proposition 2. Let $P, Q \in \pi I_1^1$. If $P \equiv Q$ and $Q \xrightarrow{\alpha} Q'$ then there exists P' such that $P \xrightarrow{\alpha} P'$ and $P' \equiv Q'$.

Now we are ready to define the preorder on processes:

Definition 14. Let $P, Q \in \pi I_1^1$. We write $P \preceq Q$ iff there exist $n, x_1, \dots, x_n, P', R, P_1, \dots, P_n, Q_1, \dots, Q_n$ 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 $i = 1, \dots, n$.

Theorem 3. Let $P \in \pi I_1^1$. Then the transition system $(\text{Deriv}(P), \mapsto, \preceq)$ is a well-structured transition system with strong compatibility, decidable $\preceq$ and computable Succ.

Corollary 1. Let $P \in \pi I_1^1$. The termination of process P is decidable.

5 Related Work and Conclusion

We have studied the expressive power of repetition and recursive process definition in the context of π -calculus with internal mobility. We have considered a mobility hierarchy $\pi I_1^1, \pi I_1^2, \dots, \pi I_1^n, \dots, \pi I_1$ [San96] which leads from the core of CCS to the π -calculus with internal mobility via a sequence of calculi with strictly increasing expressive power: each calculus πI_1^n allows mobility of order n , that is, it allows dependency chains among names of length at most n . We have proved that repetition and recursive process definition have the same expressive power, provided that the minimal form of mobility in the hierarchy is allowed. More precisely, we have shown that Random Access Machines (and therefore Turing machines) can be deterministically encoded in the language πI_1^2 which uses repetition instead of recursive definitions.

On the other hand, such an equivalence does not hold when mobility is not allowed, as we have proved that termination is decidable in πI_1^1 . Since Turing machines can be encoded deterministically in πI_D^1 (core of CCS with recursive definitions) this implies that πI_D^1 is strictly more expressive than πI_1^1 and therefore provides a formal account for the common agreement which considers recursive process definition more powerful than repetition when no name mobility is allowed. As previously mentioned πI_1^1 can be seen as a core CCS (with replication), as πI_1^1 does not allow relabeling and uses guarded choice rather than general choice. Nevertheless, we claim that our discriminating result, which shows that recursive process definition is more powerful than repetition, holds also for full CCS, since the arguments that we have used in the proofs can be extended to deal with relabeling and general choice.

Our results have the following intuitive explanation. In order to obtain a Turing powerful formalism one needs to express and manipulate the natural numbers, modeled in terms of some suitable representation. In the case of πD^1 such a representation is provided by the nesting of processes, as we have shown with our encoding of the RAMs: the successor of n can be obtained by constant application in the scope of the term representing n . Such a representation is not possible when replication is used, since the copies of a process are all at the same level and the different copies can communicate only via pure synchronization messages. Therefore, in presence of replication, we express natural numbers by exploiting name mobility, since the exchange of new names allows to hierarchically structure the various copies of a process. We have shown that a minimal form of mobility is enough, as it is sufficient to link two processes by making a name dependent on another one.

Related to the present work is also the paper [NPV02] where the authors investigate the expressive power of several timed concurrent constraint languages obtained by using different extensions of finite processes. In particular, one of the results in that paper shows that the language with replication is strictly less expressive than the language with recursive definitions of processes (in case process constants have parameters). Because of the very different underlying computational model, these results cannot be applied directly to the π -calculus.

Acknowledgments. We thank the referees for their comments and Catuscia Palamidessi and Frank Valencia for fruitful discussions on a preliminary version of the paper.

References

- [Bor98] M. Boreale. On the Expressiveness of Internal Mobility in Name-Passing Calculi. *Theoretical Computer Science*, 195(2): 205–226, 1998.
- [BGZ03] N. Busi, M. Gabbrielli, and G. Zavattaro. Replication vs. Recursive Definitions in Channel Based Calculi (extended version). Available at <http://cs.unibo.it/~zavattar/papers.html>.
- [CG00] L. Cardelli and A.D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
- [FS01] A. Finkel and Ph. Schnoebelen. Well-Structured Transition Systems Everywhere ! *Theoretical Computer Science*, 256:63–92, 2001.
- [Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
- [Mil01] R. Milner. Foreword of *The pi-calculus: a Theory of Mobile Processes*, by D. Sangiorgi and D. Walker. Cambridge University Press, 2001.
- [MPW92] R. Milner, J. Parrow, D. Walker. A calculus of mobile processes. *Journal of Information and Computation*, 100:1–77. Academic Press, 1992.
- [NPV02] M. Nielsen, C. Palamidessi, and F. D. Valencia. On the Expressive Power of Temporal Concurrent Constraint Programming Languages. In *Proc. of 4th International Conference on Principles and Practice of Declarative Programming (PPDP 2002)*. ACM Press, 2002.
- [San96] D. Sangiorgi. π -calculus, internal mobility, and agent-passing calculi. *Theoretical Computer Science*, 167(2):235–274, 1996.

Improved Combinatorial Approximation Algorithms for the k -Level Facility Location Problem

Alexander Ageev^{*1}, Yinyu Ye^{**2}, and Jiawei Zhang^{***2}

¹ Sobolev Institute of Mathematics, pr. Koptyuga 4, Novosibirsk, 630090, Russia
ageev@math.nsc.ru

² Department of Management Science and Engineering, Stanford University,
Stanford, CA 94305, USA {yinyu-ye, jiazhang}@stanford.edu

Abstract. In this paper we present improved combinatorial approximation algorithms for the k -level facility location problem. First, by modifying the path reduction developed in [2], we obtain a combinatorial algorithm with a performance factor of 3.27 for any $k \geq 2$, thus improving the previous bound of 4.56. Then we develop another combinatorial algorithm that has a better performance guarantee and uses the first algorithm as a subroutine. The latter algorithm can be recursively implemented and achieves a guarantee factor $h(k)$, where $h(k)$ is strictly less than 3.27 for any k and tends to 3.27 as k goes to ∞ . The values of $h(k)$ can be easily computed with an arbitrary accuracy: $h(2) \approx 2.4211$, $h(3) \approx 2.8446$, $h(4) \approx 3.0565$, $h(5) \approx 3.1678$ and so on. Thus, for the cases of $k = 2$ and $k = 3$ the second combinatorial algorithm ensures an approximation factor significantly better than 3, which is currently the best approximation ratio for the k -level problem provided by the non-combinatorial algorithm due to Aardal, Chudak, and Shmoys [1].

1 Introduction

In the k -level facility location problem (for brevity, k -LFLP) we are given a complete $(k + 1)$ -partite graph $G = (D \cup \mathcal{F}_1 \cup \dots \cup \mathcal{F}_k; E)$ whose node set is the union of $k + 1$ disjoint sets $D, \mathcal{F}_1, \dots, \mathcal{F}_k$ and the edge set E consists of all edges between these sets. The nodes in D are called demand points and the nodes in $\mathcal{F} = \mathcal{F}_1 \cup \dots \cup \mathcal{F}_k$ are facilities (of level 1, $\dots$, k respectively). We are given edge costs $c \in \mathbb{R}_+^E$ and opening costs $f \in \mathbb{R}_+^{\mathcal{F}}$ (i. e., opening a facility $i \in \mathcal{F}$ incurs a cost $f_i \geq 0$). The objective is to open some facilities $X_t \subseteq F_t$ on each level $t = 1, \dots, k$ and to connect each demand site $j \in D$ to a path

* Research was partially supported by the Russian Foundation for Basic Research, project codes 01-01-00786, 02-01-01153, by INTAS, project code 00-217, and by the Programme “Universities of Russia”, project code UR.04.01.012.

** Research supported in part by NSF grant DMI-0231600.

*** Research supported in part by NSF grant DMI-0231600.

(or chain) $\varphi(j) = (i_1(j), i_2(j), \dots, i_k(j))$ along open facilities $i_1(j) \in X_1, i_2(j) \in X_2, \dots, i_k(j) \in X_k$ so that the total cost of opening and connecting

$$\sum_{i \in X_1 \cup \dots \cup X_k} f_i + \sum_{j \in D} \left(c(j, i_1(j)) + c(i_1(j), i_2(j)) + \dots + c(i_{k-1}(j), i_k(j)) \right)$$

is minimized.

In this paper we consider the metric case of the problem where c is induced by a metric on the whole set of nodes $V = D \cup \mathcal{F}_1 \cup \dots \cup \mathcal{F}_k$. Recent applications of metric facility location problems include finding product clustering, cost-effective placement of servers on the internet, and optimized supply-chains.

Since the metric k -LFLP is NP-hard, the major part of research work is concentrated on designing approximation algorithms. We say that an algorithm for a minimization problem with non-negative objective function is a ρ -approximation algorithm if it runs in polynomial time and for any instance, outputs a solution of cost at most ρ times the optimum.

The special case of k -LFLP where $k = 1$ (1-LFLP) is nothing but the well-known (metric) uncapacitated facility location problem (for brevity, UFLP). It is known that the existence of a 1.463-approximation algorithm for solving UFLP would imply $P = NP$ [5]. In recent years quite a number of approximation algorithms have been developed for solving UFLP. The currently best approximation algorithm due to Mahdian, Ye, and Zhang [9] achieves a factor of 1.517. It is based on the technique of factor-revealing LP developed in [8] and [7]. See Shmoys [11] and [9] for a detailed survey on approximation algorithms for UFLP.

Obviously, the lower approximability bound 1.463 also applies to k -LFLP. On the positive side, it is known that k -LFLP can be solved within a factor of 3 by an LP rounding algorithm due to Aardal, Chudak, and Shmoys [1]. A drawback of this algorithm is that it includes a phase of solving a linear relaxation with exponential number of variables. Despite the fact that this relaxation can be solved by the ellipsoid method in polynomial time, the algorithm would be inefficient in practice. For this reason, very recently several combinatorial approximation algorithms have been developed to solve this problem. These algorithms run in strongly polynomial time but with a sacrifice in the performance guarantee. The first such algorithm by Meyerson, Munagala, and Plotkin [10] had an approximation factor of $O(\ln |\mathcal{D}|)$. A constant factor of 9.2 was later obtained by Guha, Meyerson, and Munagala [6]. Bumb and Kern [3] developed a dual ascent algorithm which had a performance guarantee of 6. Ageev [2] established that any ρ -approximation algorithm for UFLP could be translated to a 3ρ -approximation algorithm for k -LFLP. Thus, the algorithm in [9] yields a combinatorial 4.56-approximation algorithm for k -LFLP. We will refer to this approach as the *path reduction* technique.

None of the above algorithms has a performance guarantee better than 3. Whether or not k -LFLP can be approximated in polynomial time by a factor less than 3 has become a challenging open question in this field.

In this paper we present improved combinatorial approximation algorithms for the k -level facility location problem.

First, by modifying the path reduction of the k -level problem to the 1-level case developed in [2], we obtain a combinatorial algorithm with a performance guarantee of 3.27 for any k , thus improving the previous bound of 4.56. The algorithm runs in time $O(m_1^3 n^3 + m^2 n)$ where $m = |\mathcal{F}|$, $m_1 = |\mathcal{F}_1|$, and $n = |D|$. Note that the approximation ratio of this path reduction algorithm is fairly close to a factor of 3 provided by the LP rounding algorithm [1].

Though the intuition suggests that k -LFLP for small values of $k \geq 2$ may be better approximable than the general problem, our path reduction algorithm, as all the previous algorithms, has the same approximation factor for each k . This drawback motivated our work on a better algorithm whose performance factor would be an increasing function of k with values strictly less than 3.27. Our efforts resulted in a recursive combinatorial algorithm for k -LFLP, which is presented in the second part of this paper. It is based on a combination of our path reduction algorithm and a recursive reduction of k -LFLP to $(k-1)$ -LFLP and UFLP. The algorithm runs in time $O(k(m_1^3 n^3 + m^2 n))$ and achieves an approximation factor $h(k)$, where $h(k)$ is strictly less than 3.27 for any $k \geq 1$ and tends to 3.27 as k tends to ∞ . The values of $h(k)$ can be easily computed with an arbitrary accuracy. In particular, $h(2) \leq 2.4211$, $h(3) \leq 2.8446$, $h(4) \leq 3.0565$, $h(5) \leq 3.1678$. Thus, for 2-LFLP and 3-LFLP, the second algorithm achieves an approximation factor significantly better than 3.

2 The Path Reduction Algorithm

In this section we present a parameterized version of the path reduction, which in combination with the greedy algorithm developed in [9] yields a 3.27-approximation algorithm for solving k -LFLP.

2.1 Definitions and Notation

Denote by $\mathcal{P}$ the set of all paths of length $k-1$ connecting a node in $\mathcal{F}_1$ to a node in $\mathcal{F}_k$. For a path $p = (i_1, i_2, \dots, i_k) \in \mathcal{P}$, let $c(p) = \sum_{t=2}^k c(i_{t-1}, i_t)$. For any subset $X \subseteq \mathcal{F}$, let $f(X) = \sum_{i \in X} f_i$ and let $\mathcal{P}(X)$ denote the subset of paths in $\mathcal{P}$ passing through facilities in X .

Let $\mathcal{M}$ be an instance of k -LFLP and SOL be a solution of it. Recall that SOL is a pair (X, φ) where X is a set of open facilities and φ is an assignment mapping D to $\mathcal{P}(X)$. We call a path in $\mathcal{P}(D)$ a *service path*.

For our analysis it would be convenient to represent the total cost of any solution SOL for k -LFLP in the split form $F^{SOL} + C^{SOL}$, where F^{SOL} and C^{SOL} stand for the facility and connection costs, respectively. To break down C^{SOL} further, for any $t = 2, \dots, k$, let C_t^{SOL} denote the total connection cost between open facilities on level $t-1$ and open facilities on level t . Hence $C^{SOL} = \sum_{t=1}^k C_t^{SOL}$ where C_1^{SOL} stands for the total connection cost between demand sites and facilities on level 1. Similarly, let F_t^{SOL} denote the total cost to open facilities on level t , and thus $F^{SOL} = \sum_{t=1}^k F_t^{SOL}$.

To exploit the cost-split character of the objective function in k -LFLP we modify the standard definition of performance guarantee in the split way:

Definition 1. A feasible solution SOL of a k -LFLP is called (a, b) -approximate if for any other feasible solution SOL^* of the problem, the cost of SOL is at most $aF^{SOL^*} + bC^{SOL^*}$. An algorithm for a k -LFLP is a (a, b) -approximation algorithm if the solution found by the algorithm is (a, b) -approximate.

Our path reduction algorithm was inspired by the observation that the path reduction developed in [2] admits a slight modification implying that any (a, b) -approximation algorithm for UFLP can be translated into a $(a, 3b)$ -approximation algorithm for k -LFLP. Therefore, to obtain a good approximation factor for k -LFLP, we have to solve the reduced UFLP in such a way that the performance guarantee pair (a, b) approximately satisfies $a = 3b$. To this point we apply the algorithm of Mahdian et al. [9] to obtain a guarantee pair $(3.27, 1.09)$ for UFLP, which then implies a 3.27-approximation for k -LFLP.

2.2 Parameterized Path Reduction

We now describe a path reduction with positive parameters a, b that generalizes the reduction in [2] (corresponding to the case $a = b = 1$).

Path reduction with parameters (a, b) . Let $\mathcal{M}$ be an instance of k -LFLP. For each $i_1 \in \mathcal{F}_1$ and $t \in \{1, \dots, |D|\}$, compute a path $p(i_1, t)$ that has the minimum value of $t \cdot bc(p) + af(p)$ over all paths $p \in \mathcal{P}$ starting from i_1 . (Note that the problem of finding such paths can be easily reduced to the shortest path problem and there are total $|\mathcal{F}_1| \cdot |D|$ of such paths.) Then, associate with $\mathcal{M}$ an instance $\mathcal{S}$ of UFLP in which the set of demand nodes is $\mathcal{D}$, and the set of “facilities” is the set of all pairs (i_1, t) where $i_1 \in \mathcal{F}_1$ and $t \in \{1, \dots, |D|\}$. In $\mathcal{S}$, for any demand node $j \in D$ and “facility” (i_1, t) , the cost of connecting j to (i_1, t) is defined to be $c(j, i_1) + c(p(i_1, t))$; and the cost of opening (i_1, t) is defined to be $f(p(i_1, t))$ (i.e., equal to the cost of opening all facilities on path $p(i_1, t)$). Given a solution $SOLS$ of $\mathcal{S}$, we construct back a solution $SOLM$ of $\mathcal{M}$ as follows: for any $j \in D$, connect j to the service path $p(i_1(j), t)$ such that $(i_1(j), t)$ is the “facility” serving j in $\mathcal{S}$, and open the facilities on all such service paths.

The main result of this subsection is the following theorem.

Theorem 1. If $SOLS$ is an (a, b) -approximate solution of $\mathcal{I}$, then $SOLM$ is an $(a, 3b)$ -approximate solution of $\mathcal{M}$. Furthermore, for any solution SOL of $\mathcal{M}$,

$$F^{SOLM} + C^{SOLM} \leq aF^{SOL} + bC_1^{SOL} + 3b \sum_{i=2}^k C_i^{SOL}. \quad (1)$$

Therefore, we have

Corollary 1. Any (a, b) -approximation algorithm for solving UFLP yields an $(a, 3b)$ -approximation algorithm for solving k -LFLP. $\square$

Our proof of the theorem is based on Lemmas 1 and 2 below. The first lemma is an easy counterpart of Lemma 2 in [2].

Lemma 1.

$$F^{SOLM} \leq F^{SOLS} \quad \text{and} \quad C^{SOLM} = C^{SOLS}.$$

□

The second lemma is a counterpart of Lemma 4 in [2].

Lemma 2. *For any solution SOL of $\mathcal{M}$, there exists a corresponding solution SOL^* of the reduced $\mathcal{S}$ such that*

$$aF^{SOL^*} + bC^{SOL^*} \leq aF^{SOL} + bC_1^{SOL} + 3b \sum_{t=2}^k C_t^{SOL}. \quad (2)$$

We first deduce Theorem 1 from the above lemmas.

Proof (of Theorem 1.). Let SOL^* be any solution of $\mathcal{M}$. By Lemma 2, there exists a corresponding solution SOL of $\mathcal{S}$ such that

$$aF^{SOL} + bC^{SOL} \leq aF^{SOL^*} + bC_1^{SOL^*} + 3b \sum_{t=2}^k C_t^{SOL^*} \leq aF^{SOL^*} + 3bC^{SOL^*}.$$

On the other hand, by using Lemma 1 and the fact that $SOLS$ is an (a, b) -approximate solution of $\mathcal{S}$, we have

$$F^{SOLM} + C^{SOLM} \leq F^{SOLS} + C^{SOLS} \leq aF^{SOL} + bC^{SOL},$$

which proves (1). □

To prove Lemma 2 we need the following easy statement, which being a bit stronger than Lemma 3 in [2], has an almost identical proof.

Lemma 3. *Let $\mathcal{I}$ be an instance of k -level FLP and $\overline{SOL}$ be a solution of $\mathcal{I}$. Then $\mathcal{I}$ has a solution $SOL = (X, \varphi)$ such that*

- (i) *if in paths $\varphi(j') = (i'_1, \dots, i'_k)$ and $\varphi(j'') = (i''_1, \dots, i''_k)$ $i'_l = i''_l$ for some l , then $i'_r = i''_r$ for all $r \geq l$;*
- (ii) $C_1^{SOL} = C_1^{\overline{SOL}}$, $\sum_{l=2}^k C_l^{SOL} \leq \sum_{l=2}^k C_l^{\overline{SOL}}$, $F^{SOL} \leq F^{\overline{SOL}}$. □

The above lemma implies that any solution $\overline{SOL}$ of k -LFLP can be replaced by a solution SOL satisfying (ii) and whose service paths constitute a forest consisting of trees rooted at level k .

Proof (of Lemma 2). Let $SOL = (X, \varphi)$ be a solution of $\mathcal{M}$. For any $j \in D$, let $\varphi(j) = (i_1(j), \dots, i_k(j))$. By Lemma 3 we may assume that SOL satisfies property (i) and thus the service paths of SOL constitute a forest consisting of trees rooted at open facilities in $\mathcal{F}_k$.

For every open facility $u \in X_k = X \cap \mathcal{F}_k$ lying on level k , let D_u be the set of demand sites assigned, by φ , to a path finishing in u , and $p(u)$ be a path having minimum value of $c(p)$ among all service paths p ending in u . Also, let $\mu(u)$ be the starting facility of $p(u)$ lying on level 1.

Define a new solution $SOLP = (X, \varphi')$ by reassigning each $j \in D_u$ to the path $p(u)$, i. e., by setting $\varphi'(j) = p(u)$ for all $u \in X_k$. Thus, by definition, $SOLP$ satisfies

$$F^{SOLP} \leq F^{SOL} \quad (3)$$

and

$$C^{SOLP} = \sum_{u \in X_k} \sum_{j \in D_u} (c(j, \mu(u)) + c(p(u)))$$

By the triangle inequality and the definitions of $p(u)$ and $\mu(u)$,

$$\begin{aligned} c(j, \mu(u)) + c(p(u)) &\leq (c(j, i_1(j)) + c(\varphi(j)) + c(p(u))) + c(p(u)) \\ &\leq c(j, i_1(j)) + 3c(\varphi(j)). \end{aligned}$$

Thus we have

$$\begin{aligned} C^{SOLP} &\leq \sum_{u \in X_k} \sum_{j \in D_u} (c(j, i_1(j)) + 3c(\varphi(j))) \\ &= C_1^{SOL} + 3 \sum_{t=2}^k C_t^{SOL}. \end{aligned} \quad (4)$$

Now, by (3) and (4), it suffices to show that there exists a solution SOL^* of $\mathcal{S}$ such that

$$aF^{SOL^*} + bC^{SOL^*} \leq aF^{SOLP} + bC^{SOLP}. \quad (5)$$

Since the service paths of $SOLP$ are disjoint, we have

$$\begin{aligned} aF^{SOLP} + bC^{SOLP} &= \sum_{u \in X_k} \left(af(p(u)) + b \sum_{j \in D_u} (c(j, \mu(u)) + c(p(u))) \right) \\ &= \sum_{u \in X_k} \left(af(p(u)) + b|D_u| \cdot c(p(u)) + b \sum_{j \in D_u} c(j, \mu(u)) \right) \\ &= \sum_{u \in X_k} \left(af(p(u)) + b|D_u| \cdot c(p(u)) \right) + bC_1^{SOLP}. \end{aligned}$$

Now we define a solution SOL^* of $\mathcal{S}$ by declaring all facilities lying on the paths $p(\mu(u), |D_u|)$, $u \in X_k$, open and by connecting j to the path $p(\mu(u), |D_u|)$ whenever $j \in D_u$. Then we have

$$\begin{aligned} aF^{SOL^*} + bC^{SOL^*} &= \sum_{u \in X_k} \left(af(p(\mu(u), |D_u|)) + b|D_u| \cdot c(p(\mu(u), |D_u|)) \right) \\ &\quad + bC_1^{SOLP} \leq aF^{SOLP} + bC^{SOLP}. \end{aligned}$$

The last inequality holds because for each $u \in X_k$, by the construction of paths $p(i_1, t)$ in the parameterized path reduction,

$$af(p(\mu(u), |D_u|)) + b|D_u| \cdot c(p(\mu(u), |D_u|)) \leq af(p(u)) + b|D_u| \cdot c(p(u)). \quad \square$$

The next subsection analyzes particular values of parameters (a, b) to establish our final result.

2.3 Algorithm PATH REDUCTION&GREEDY

To solve the instance $\mathcal{S}$ of UFLP we use the greedy algorithm developed in [9] (in the sequel referred to as GREEDY). We refer the reader to [9] for the details of GREEDY. Here we only need two results from [9].

Lemma 4 ([9]). *Let $\gamma_f^* \geq 1$ and $\gamma_c^* = \sup_k \{z_k\}$, where z_k is the solution of the following optimization program (which we call the factor-revealing LP).*

$$\begin{aligned}
 & \text{Maximize} \quad \frac{\sum_{i=1}^k \alpha_i - \gamma_f^* f}{\sum_{i=1}^k d_i} \\
 & \text{subject to:} \\
 & \quad \alpha_i \leq \alpha_{i+1} \quad \forall 1 \leq i < k, \\
 & \quad r_{j,i} \geq r_{j,i+1} \quad \forall 1 \leq j < i < k, \\
 & \quad \alpha_i \leq r_{j,i} + d_i + d_j \quad \forall 1 \leq j < i \leq k, \\
 & \quad \sum_{j=1}^{i-1} \max(r_{j,i} - d_j, 0) + \sum_{j=i}^k \max(\alpha_i - d_j, 0) \leq f \quad \forall 1 \leq i \leq k, \\
 & \quad \alpha_j, d_j, f, r_{j,i} \geq 0 \quad \forall 1 \leq j \leq i \leq k.
 \end{aligned}$$

Then for any $\delta \geq 1$, Algorithm GREEDY is a $(\gamma_f^* + \ln \delta, 1 + \frac{\gamma_c^* - 1}{\delta})$ -approximation algorithm for UFLP.

For any given γ_f^* , one can solve the above linear program to compute γ_c^* . However, since the number of variables here is unbounded, it is unlikely to be computable exactly. In [9], the problem is solved by constructing a feasible solution to the dual of this linear program, which provides an upper bound on γ_c^* .

The crucial result of [9] is the following

Lemma 5 ([9]). *If $\gamma_f^* = 1.11$, then $\gamma_c = 1.78$ is an upper bound on γ_c^* .*

Let $\gamma_f(\delta) = \gamma_f + \ln \delta$ and $\gamma_c(\delta) = 1 + \frac{\gamma_c - 1}{\delta}$ where $\gamma_f = 1.11$, $\gamma_c = 1.78$. By Lemmas 4 and 5, we have the following

Lemma 6. *Algorithm GREEDY is an $(\gamma_f(\delta), \gamma_c(\delta))$ -approximation algorithm for any $\delta \geq 1$. $\square$*

By this lemma, the path reduction algorithm produces a $(\gamma_f(\delta), 3\gamma_c(\delta))$ -approximation algorithm for k -LFLP where δ is an arbitrary number ≥ 1 . By taking $\delta = 8.67$, one can see that our algorithm, which we will further refer to as PATH REDUCTION&GREEDY, finds a solution within a factor of 3.27 of the minimal cost.

Note that the paths $p(i_1, t)$ in the parameterized path reduction can be computed in $O(m^2n)$ time. On the other hand, the total number of demand sites and facilities in the reduced $\mathcal{S}$ is $n + m_1n$ and thus GREEDY requires $O(m_1^3n^3)$ time to solve it. Therefore, the overall running time of PATH REDUCTION&GREEDY is $O(m_1^3n^3 + m^2n)$.

We remark that the bound 3.27 cannot be improved much by just using Corollary 1 as a tool box. It is known [5] that for any $x \geq 1$, the existence of $(x, 1 + 2e^{-x})$ -approximation algorithm for UFLP would imply $P = NP$. Therefore, the best we could get by using Corollary 1 is 3.236 since $x + 3(1 + 2e^{-x}) \geq 6.472$ for any $x \geq 1$.

3 The Recursive Path Reduction Algorithm

A drawback of algorithm PATH REDUCTION&GREEDY is that the approximation factor of 3.27 it provides does not depend on the number of levels k , whereas 1-LFLP admits a 1.52-approximation and the intuition suggests that k -LFLP for small values of k must be much better approximable than the general problem.

In this section, we present an improved combinatorial algorithm for k -LFLP, which we refer to as SPLIT&RECURSION. It is based on a combination of PATH REDUCTION&GREEDY and a recursive reduction of k -LFLP to $(k - 1)$ -LFLP and UFLP. Algorithm SPLIT&RECURSION runs in time $O(km_1^3n^3 + km^2n)$ and achieves an approximation factor $h(k)$, where $h(k) < 3.27$ for any $k \geq 1$ and tends to 3.27 as k tends to ∞ . The values of $h(k)$ can be easily computed with an arbitrary accuracy. In particular, $h(2) \approx 2.4211$, $h(3) \approx 2.8446$, $h(4) \approx 3.0565$, $h(5) \approx 3.1678$.

3.1 Definitions and High Level Description

We first give a few definitions.

For any instance $\mathcal{M}$ of k -LFLP, we define an instance $\mathcal{M}_{k-1}$ of $(k - 1)$ -LFLP and an instance $\mathcal{S}$ of UFLP (1-LFLP) in the following way:

1. $\mathcal{M}_{k-1}$ is obtained from $\mathcal{M}$ by deleting all the facilities at level 1 (or, by opening for free all facilities on level 1). Thus, in $\mathcal{M}_{k-1}$ the set of facilities lying on level r is $\mathcal{F}_{r+1}$, and the connection cost between $j \in D$ and $i_2 \in \mathcal{F}_2$ is

$$\min_{v \in \mathcal{F}_1} \{c(j, v) + c(v, i_2)\}.$$

2. $\mathcal{S}$ is obtained from $\mathcal{M}$ by deleting all facilities at levels greater than 1 (and all edges incident with these facilities), and by doubling all the edge costs between D and $\mathcal{F}_1$.

We now ready to proceed to a high level description of the algorithm.

In the case $k = 2$, $\mathcal{M}_1$ and $\mathcal{S}$ are both instances of UFLP and we solve them by GREEDY. Now we have that each $j \in D$ is assigned to a facility $i_2(j) \in \mathcal{F}_2$ by the solution for $\mathcal{M}_1$ and to a facility $i_1(j) \in \mathcal{F}_1$ by the solution for $\mathcal{S}$. On the basis of these solutions we construct a solution for $\mathcal{M}$, denoted by *SOLMS*, by connecting each j to the path $(i_1(j), i_2(j))$.

Note that the straightforward variant of the above construction where the connection costs coincide with the original ones in both instances of UFLP yields

a simple factor 3 reduction of 2-LFLP to UFLP. This reduction was first observed by Gimadi [4].

When $k \geq 3$ our algorithm solves $\mathcal{S}$ by applying GREEDY and calls itself to solve $\mathcal{M}_{k-1}$. Now we have that the solution of $\mathcal{S}$ assigns each $j \in D$ to a facility $i_1(j) \in cF_1$ while the solution to $\mathcal{M}_{k-1}$ assigns each $j \in D$ to a path $(i_2(j) \in \mathcal{F}_2, \dots, i_k(j) \in \mathcal{F}_k)$. In this case the solution *SOLMS* for $\mathcal{M}$ is constructed by connecting each j to the composite path $(i_1(j), i_2(j), \dots, i_k(j))$.

However, the constructed solution *SOLMS* is not yet the output of the algorithm. In addition, we find another solution *SOLPG* for $\mathcal{M}$ by applying PATH REDUCTION&GREEDY and finally output a solution having lower cost among the two.

By unfolding this recursive description one can easily obtain a conventional implementation as follows. The algorithm applies GREEDY to solve k instances of UFLP obtained from the original instance $\mathcal{M}$ by deleting the facilities on all levels except a fixed one. It then applies PATH REDUCTION&GREEDY to solve $k-1$ instances of k -LFLP obtained from $\mathcal{M}$ by deleting the facilities on all levels smaller than a fixed one. Finally, in $k-1$ steps, on the basis of the retrieved solutions, it constructs an output solution.

From the above implementation it is clear that SPLIT&RECURSION can be implemented in $O(k(m_1^3 n^3 + m^2 n))$ time.

3.2 Algorithm SPLIT&RECURSION

Now we proceed to a formal description and analysis of the algorithm.

Algorithm SPLIT&RECURSION:

Input: An instance $\mathcal{M}$ of k -LFLP.

Output: A solution *SOL* for $\mathcal{M}$.

if $k = 1$ then

SOL := the solution obtained by applying GREEDY to $\mathcal{M}$;

endif

if $k \geq 2$ then

Apply SPLIT&RECURSION to find a solution *SOLM* for $\mathcal{M}_{k-1}$ and GREEDY to find a solution *SOLS* for $\mathcal{S}$;

Construct a solution *SOLMS* for $\mathcal{M}_{k-1}$ by connecting each $j \in D$ to the path $(i_1(j), i_2(j), \dots, i_k(j))$ whenever j connects to $i_1(j)$ in *SOLS* and to the path $(i_2(j), \dots, i_k(j))$ in *SOLM*;

Apply PATH REDUCTION&GREEDY to find a solution *SOLPG* of $\mathcal{M}$;

SOL := a solution having lower cost among *SOLMS* and *SOLPG*.

endif

The following theorem is the main result of this section.

Theorem 2. *Let $k \geq 2$. For any solution SOL^* of $\mathcal{M}$ and any $\delta \geq 1$, the solution SOL retrieved by SPLIT&RECURSION satisfies*

$$F^{SOL} + C^{SOL} \leq \gamma_f(\delta)F^{SOL^*} + \theta(k)\gamma_c(\delta)C^{SOL^*} \quad (6)$$

where

$$\theta(k) = 3 \left(1 - \frac{1}{2^{k-2}} \right) + \frac{1}{2^{k-3}}.$$

Since $\gamma_f(\delta)$ is a strictly increasing function of δ on the interval $[1, \infty)$ whereas $\theta(k)\gamma_c(\delta)$ is strictly decreasing, the minimum value of

$$\rho_k(\delta) = \max(\gamma_f(\delta), \theta(k)\gamma_c(\delta))$$

is attained at a unique root δ_k of the transcendent equation

$$\gamma_f(\delta) = \theta(k)\gamma_c(\delta).$$

Thus we derive

Corollary 2. *SPLIT&RECURSION is a $\rho_k(\delta_k)$ -approximation algorithm for k -LFLP.*

By using a binary search, it is easy to compute δ_k approximately for every k . This gives

$$\begin{aligned} \rho_2(\delta_2) &\leq \rho_2(3.71) < 2.4211, \\ \rho_3(\delta_3) &\leq \rho_3(5.66) < 2.8446, \\ \rho_4(\delta_4) &\leq \rho_4(7.0) < 3.0565, \\ \rho_5(\delta_5) &\leq \rho_5(7.66) < 3.1678. \end{aligned}$$

One can also see that as $k \rightarrow \infty$, $\theta(k)$ tends to 3 and the performance factor tends to 3.27 as in algorithm PATH REDUCTION&GREEDY.

Proof (Proof of Theorem 2.). We proceed by induction on k . Let SOL^* be any solution of $\mathcal{M}$. Then, by Theorem 1,

$$F^{SOLPG} + C^{SOLPG} \leq \gamma_f(\delta)F^{SOL^*} + \gamma_c(\delta)C_1^{SOL^*} + 3\gamma_c(\delta) \sum_{t=2}^k C_t^{SOL^*}. \quad (7)$$

Observe that SOL^* induces a solution, $SOLS^*$, to $\mathcal{S}$ and a solution, $SOLM^*$, to $\mathcal{M}_{k-1}$; as SOL^* assigns every demand node j to a facility, say $i_t^*(j) \in \mathcal{F}_t$ for each $t = 1, \dots, k$. That is, j in $SOLS^*$ is assigned to $i_1^*(j)$ of $\mathcal{S}$ with connection cost $c(j, i_1^*(j))$, and j in $SOLM^*$ is assigned to $(i_2^*(j), \dots, i_k^*(j))$ in $\mathcal{M}_{k-1}$ with connection cost at most

$$c(j, i_1^*(j)) + c(i_1^*(j), i_2^*(j)) + c((i_2^*(j), \dots, i_k^*(j)))$$

from the construction of the connection costs, see (1).

Recall that the connections costs in $\mathcal{S}$ are doubled from the edge costs between D and $\mathcal{F}_1$ in $\mathcal{M}$. Hence, by Lemma 6, we have

$$\begin{aligned} F^{SOLS} + C^{SOLS} &\leq \gamma_f(\delta)F^{SOLS*} + 2\gamma_c(\delta)C^{SOLS*} \\ &= \gamma_f(\delta)F_1^{SOL*} + 2\gamma_c(\delta)C_1^{SOL*} \end{aligned} \quad (8)$$

Assume now that $k = 2$. In this case $\mathcal{M}_{k-1}$ is an instance of UFLP and thus by Lemma 6 and the definition of $\mathcal{M}_{k-1}$,

$$\begin{aligned} F^{SOLM} + C^{SOLM} &\leq \gamma_f(\delta)F^{SOLM*} + \gamma_c(\delta)C^{SOLM*} \\ &\leq \gamma_f(\delta)F_2^{SOL*} + \gamma_c(\delta)(C_1^{SOL*} + C_2^{SOL*}). \end{aligned} \quad (9)$$

By the construction of *SOLMS* and the triangle inequality,

$$\begin{aligned} F^{SOLMS} + C^{SOLMS} &= F^{SOLS} + F^{SOLM} + \frac{1}{2}C^{SOLS} + \sum_{j \in D} c(i_1(j), i_2(j)) \\ &\leq F^{SOLS} + F^{SOLM} + \frac{1}{2}C^{SOLS} + \left(\frac{1}{2}C^{SOLS} + C^{SOLM} \right) \\ &= F^{SOLS} + C^{SOLS} + F^{SOLM} + C^{SOLM}, \end{aligned}$$

and thus, by (8) and (9), we have

$$F^{SOLMS} + C^{SOLMS} \leq \gamma_f(\delta)F^{SOL*} + 3\gamma_c(\delta)C_1^{SOL*} + \gamma_c(\delta)C_2^{SOL*}.$$

Since the cost of *SOL* is at most half as great as the sum of costs of *SOLMS* and *SOLPG* (7),

$$F^{SOL} + C^{SOL} \leq \gamma_f(\delta)F^{SOL*} + 2\gamma_c(\delta)C_1^{SOL*} + 2\gamma_c(\delta)C_2^{SOL*},$$

which is nothing but (6) for $k = 2$.

Now, assume that (6) is true for each number of levels smaller than k . Then, since $\mathcal{M}_{k-1}$ is an instance of $(k-1)$ -LFLP, and by the definition of $\mathcal{M}_{k-1}$,

$$\begin{aligned} F^{SOLM} + C^{SOLM} &\leq \gamma_f(\delta) \sum_{t=2}^k F_t^{SOL*} + \theta(k-1)\gamma_c(\delta)(C_1^{SOL*} + C_2^{SOL*}) + \\ &\quad \theta(k-1)\gamma_c(\delta) \sum_{t=3}^k C_t^{SOL*}. \end{aligned} \quad (10)$$

Again, by the construction of *SOLMS* and the triangle inequality,

$$F^{SOLMS} + C^{SOLMS} \leq F^{SOLS} + C^{SOLS} + F^{SOLM} + C^{SOLM},$$

and thus, by (8) and (10),

$$\begin{aligned} F^{SOLMS} + C^{SOLMS} &\leq \gamma_f(\delta)F^{SOL*} + (\theta(k-1) + 2)\gamma_c(\delta)C_1^{SOL*} + \\ &\quad \theta(k-1)\gamma_c(\delta) \sum_{t=2}^k C_t^{SOL*}. \end{aligned}$$

Together with (7), this yields

$$F^{SOL} + C^{SOL} \leq \gamma_f(\delta) F^{SOL*} + \frac{\theta(k-1) + 3}{2} \gamma_c(\delta) C^{SOL*}.$$

Since

$$\theta(k) = \frac{\theta(k-1) + 3}{2},$$

(6) follows. $\square$

References

1. K. Aardal, F.A. Chudak, and D.B. Shmoys, “A 3-approximation algorithm for the k -level uncapacitated facility location problem,” *Information Processing Letters* 72 (1999), 161–167.
2. A. A. Ageev, “Improved approximation algorithms for multilevel facility location problems,” *Oper. Res. Letters* 30 (2002), 327–332. The conference version appeared in *Proceedings of 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2002)*, LNCS 2462 (2002), 5–13.
3. A.F. Bumb and W. Kern, “A simple dual ascent algorithm for the multilevel facility location problem,” *4th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2001)*, LNCS 2129 (2001), 55–62.
4. E. Kh. Gimadi, personal communication.
5. S. Guha and S. Kuller, “Greedy strikes back: improved facility location algorithms,” *Journal of Algorithms* 31 (1999), 228–248.
6. S. Guha, A. Meyerson, and K. Munagala, “Hierarchical placement and network design problems,” *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS 2000)*, 2000, 603–612.
7. K. Jain, M. Mahdian, and A. Saberi, “A new greedy approach for facility location problems”, in: *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC’02)*, Montreal, Quebec, Canada, May 19–21, 2002.
8. M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani, “A greedy facility location algorithm analyzed using dual fitting”, in: *Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX’2001)*, Berkeley, CA, USA, August 18–20, Lecture Notes in Computer Science, Vol. 2129, 127–137.
9. M. Mahdian, Y. Ye, and J. Zhang, “Improved approximation algorithms for metric facility location problems,” *5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2002)*, LNCS 2462 (2002) 229–242.
10. A. Meyerson, K. Munagala, and S. Plotkin, “Cost-distance: two-metric network design,” *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS 2000)*, 2000, 624–630.
11. D. B. Shmoys, “Approximation algorithms for facility location problems,” *3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, LNCS 1913 (2000) 27–33.

An Improved Approximation Algorithm for the Asymmetric TSP with Strengthened Triangle Inequality

Markus Bläser

Institut für Theoretische Informatik, Universität zu Lübeck
Wallstraße 40, 23560 Lübeck, Germany
blaeser@tcs.uni-luebeck.de

Abstract. We consider the asymmetric traveling salesperson problem with γ -parameterized triangle inequality for $\gamma \in [\frac{1}{2}, 1)$. That means, the edge weights fulfill $w(u, v) \leq \gamma \cdot (w(u, x) + w(x, v))$ for all nodes u, v, x . Chandran and Ram [6] recently gave the first constant factor approximation algorithm with polynomial running time for this problem. They achieve performance ratio $\frac{\gamma}{1-\gamma}$. We devise an approximation algorithm with performance ratio $\frac{1}{1-\frac{1}{2}(\gamma+\gamma^3)}$, which is better than the one by Chandran and Ram for $\gamma \in [0.6507, 1)$, that is, for the particularly interesting large values of γ .

1 Introduction

The traveling salesperson problem is a well-known NP optimization problem. Given a complete loopless graph G and a weight function w that assigns to each edge a nonnegative weight, our goal is to find a tour of minimum weight that visits each node exactly once. In general, the graph G may be directed. In this case, one also speaks of the asymmetric traveling salesperson problem (ATSP). An important and well-studied special case is the case where w is symmetric (TSP), that is, $w(u, v) = w(v, u)$ for all $u, v \in V$. In other words, the underlying graph can be considered undirected.

TSP and henceforth ATSP are both NPO-complete. Thus there is no good approximation algorithm for these two problems, unless $\text{NP} = \text{P}$. A natural restriction is that the weight function w should fulfill the triangle inequality

$$w(u, v) \leq w(u, x) + w(x, v) \quad \text{for all } u, x, v \in V. \quad (1)$$

We call the corresponding problems Δ -ATSP and Δ -TSP in the asymmetric and symmetric case, respectively. For Δ -TSP, Christofides [7] devised a $\frac{3}{2}$ approximation algorithm with polynomial running time, whereas the best approximation algorithm for Δ -ATSP has only performance ratio $\log n$. This was shown by Frieze, Galbiati, and Maffioli [8]. See also [4] for some slight improvement. Many researchers conjecture that there is also a constant factor approximation algorithm for Δ -ATSP, but this question is still open after more than two decades.

We here consider a strengthening of the triangle inequality (1), which allows a constant factor approximation: Let γ be some constant with $\frac{1}{2} \leq \gamma < 1$. An instance of the problem $\Delta(\gamma)$ -ATSP is a complete loopless directed graph G with node set V and a weight function w assigning to each edge of G a nonnegative weight. The weight function fulfills the γ -parameterized triangle inequality, i.e.,

$$w(u, v) \leq \gamma \cdot (w(u, x) + w(x, v)) \quad \text{for all } u, x, v \in V. \quad (2)$$

The goal is to compute a TSP tour of minimum weight.

One can also view the γ -parameterized triangle inequality as a data dependent bound. Given an instance of Δ -ATSP, we compute $\tilde{\gamma} = \max\{\frac{w(u, v)}{w(u, x) + w(x, v)}\}$ and use our algorithm to obtain better performance guarantees on instances where $\tilde{\gamma}$ is small enough. If $\tilde{\gamma} + \tilde{\gamma}^3 \leq 2 - 2/\log n$, then this is better than the $\log n$ upper bound.

1.1 Previous and New Results

As mentioned above, for Δ -ATSP and Δ -TSP, there are approximation algorithms with polynomial running time achieving performance ratios $\log n$ and $\frac{3}{2}$, respectively.

Böckenhauer et al. [5] studied the symmetric traveling salesperson problem with γ -parameterized triangle inequality for $\gamma \in [\frac{1}{2}, 1)$. They achieve approximation performance $\min\{1 + \frac{2\gamma-1}{3\gamma^2-2\gamma+1}, \frac{2}{3} + \frac{\gamma}{3(1-\gamma)}\}$. Andreae and Bandelt [2] as well as Bender and Chekuri [3] considered the symmetric case with γ -parameterized triangle inequality for $\gamma \geq 1$. Combining their algorithms, we get an approximation algorithm with performance guarantee $\min\{\frac{3\gamma^2+\gamma}{2}, 4\gamma\}$.

Recently, Chandran and Ram [6] studied the asymmetric traveling salesperson problem with γ -parameterized triangle inequality for $\gamma \in [\frac{1}{2}, 1)$. They designed a constant factor approximation algorithm with performance ratio (asymptotically) $\frac{\gamma}{1-\gamma}$, in contrast to the $\log n$ upper bound for Δ -ATSP. Our Lemma 2 shows that the algorithm of Frieze, Galbiati, and Maffioli without any modifications already yields a $\frac{1}{1-\gamma}$ approximation for $\Delta(\gamma)$ -ATSP. Since we even do not know whether for $\gamma = 1$ an approximation algorithm with constant performance ratio exists, studying the case $\gamma \geq 1$ does not look very promising at the moment.

As our main result, we present an approximation algorithm with performance ratio $\frac{1}{1-\frac{1}{2}(\gamma+\gamma^3)}$. This improves the result by Chandran and Ram for $\gamma \in [0.6507, 1)$, that is, for the particularly interesting large values of γ . The running time of our algorithm is $O(n^3)$, which matches the running time of the algorithm by Chandran and Ram.

1.2 Notations and Conventions

For a set of nodes V , let $K(V)$ denote the set of edges $V \times V \setminus \{(v, v) \mid v \in V\}$. Throughout this work, we are considering directed graphs $G = (V, K(V))$

together with a weight function $w : K(V) \rightarrow \mathbb{Q}_{\geq 0}$ and a parameter $\gamma \in [\frac{1}{2}, 1)$. We always require that w fulfills the γ -parameterized triangle inequality (2). (Note that if w fulfills the γ -parameterized triangle inequality for some γ , then necessarily $\gamma \geq \frac{1}{2}$. Thus the lower bound is no restriction. We require $\gamma < 1$, since we already do not know how to achieve constant performance ratio for $\gamma = 1$.)

For a directed edge $e = (u, v)$, u is called the *tail* of e and v is called the *head* of e .

A cycle cover of a directed graph G is a spanning subgraph that consists solely of node disjoint directed cycles. A cycle is called a k -cycle if it has length *exactly* k . For any subgraph $S = (V, E)$ of G , the weight $w(S)$ of S is defined as the sum of the weights of the edges in E , that is, $w(S) = \sum_{e \in E} w(e)$. In particular, this defines the weight of cycle covers and TSP tours.

For a given directed graph G with weight function w , let $\text{AB}(G)$ denote the weight of a minimum weight cycle cover. (This is also called the *assignment bound*.) Furthermore, let $\text{TSP}(G)$ denote the weight of a minimum weight TSP tour of G . Obviously, we have $\text{AB}(G) \leq \text{TSP}(G)$. $\text{AB}(G)$ and a corresponding minimum weight cycle cover can be computed in polynomial time.

2 Approximation Algorithm

Figure 1 shows our new approximation algorithm. It generalizes a repeated cycle cover approach by Frieze, Galbiati, and Maffioli [8]. We first compute a minimum weight cycle cover C . This can be done in time $O(n^3)$. (There are various algorithms with this time bound, see e.g. [1] for an overview.) Then we choose from each cycle two nodes as representatives. One of them is placed in the set V_1 , the other is put into the set V_2 . Then we recursively compute two TSP tours T_1 and T_2 , one in the graph G_1 induced by V_1 (i.e., $G_1 = (V_1, K(V_1))$) with weight function w_1 where w_1 is the restriction of w to $K(V_1)$ and the other in the graph G_2 induced by V_2 . Then we combine C and the lighter of the two tours T_1 and T_2 and obtain a tour T by taking shortcuts.

The next lemma bounds the weight of a minimum weight TSP tour of G_1 and G_2 in terms of the weight of a minimum weight TSP tour of G .

Lemma 1. *Let C be a cycle cover of G and let $V_1, V_2 \subseteq V$ be disjoint sets such that V_1 and V_2 contain exactly one node from each cycle of C . Let G_1 and G_2 be the graphs induced by V_1 and V_2 , respectively. Then*

$$\text{TSP}(G_1) + \text{TSP}(G_2) \leq (1 + \gamma^2) \cdot \text{TSP}(G).$$

Proof. First we assume that each cycle in C is a 2-cycle. Thereafter, we reduce the general case to this special case.

Let T be a minimum weight TSP tour of G . Thus $w(T) = \text{TSP}(G)$. We construct two TSP tours T_1 and T_2 of G_1 and G_2 , respectively, such that $w(T_1) + w(T_2) \leq (1 + \gamma^2) \cdot w(T)$. This proves the claim of the lemma for the special case where each cycle of C is a 2-cycle.

Input: directed graph $G = (V, K(V))$ with weight function w
 where w fulfills the γ -parameterized triangle inequality (2)
 for some $\frac{1}{2} \leq \gamma < 1$.
Output: TSP tour T .

1. Compute a minimum weight cycle cover C of G .
2. Choose two disjoint sets $V_1, V_2 \subseteq V$ such that both V_1 and V_2 contain exactly one node from each cycle of C .
3. If $|V_1| > 1$ then recursively compute two TSP tours T_1 and T_2 of the graphs G_1 and G_2 that are induced by V_1 and V_2 .
4. W.l.o.g. assume that T_1 is lighter than T_2 . Construct T from C and T_1 as follows:
 - a) Construct an Eulerian tour E of $(V, C \cup T_1)$. This tour visits the nodes of V_1 in the order given by T_1 . For each such node $v \in V_1$, the tour runs through the (unique) cycle in C that v belongs to. After that, it goes on with the next node of T_1 .
 - b) From E , we obtain T by taking *shortcuts*: Whenever E would visit a node it has visited before, T goes directly to the next node not visited before.

Fig. 1. Approximation algorithm for $\Delta(\gamma)$ -ATSP

Given T , we construct T_1 and T_2 by taking *shortcuts*, that is, we move along the tour T starting with an arbitrary node in V_1 or V_2 , respectively. Whenever we would visit a node not in V_1 or V_2 , respectively, we directly go to the next node of T that is in V_1 or V_2 .

Let $e = (u, v)$ be an edge of T . Since C consists solely of 2-cycles by assumption, $u, v \in V_1 \cup V_2$. If both u and v belong to V_1 , then the edge e appears in T_1 but is contracted twice when constructing T_2 . Since w satisfies the γ -parameterized triangle inequality, e contributes weight $w(e)$ to T_1 and $\gamma^2 \cdot w(e)$ to T_2 yielding a total contribution of $(1 + \gamma^2) \cdot w(e)$. If both u and v belong to V_2 , the same analysis works. If u belongs to V_1 and v belongs to V_2 or vice versa, then e is contracted once to obtain T_1 and once to obtain T_2 . Thus the total contribution is $2\gamma \cdot w(e) \leq (1 + \gamma^2) \cdot w(e)$. Summing over all edges e of T yields the result.

The special case proven above implies the general case as follows: We construct a TSP tour T' from T by taking shortcuts such that T' only visits nodes from V_1 and V_2 . Since w particularly obeys the triangle inequality, $w(T') \leq w(T)$. Now we can apply the above special case to T' . $\square$

The following lemma bounds the weight of the final tour T in terms of the weight of C and T_1 .

Lemma 2. *For the TSP tour T constructed in the algorithm in Figure 1, we have $w(T) \leq w(C) + \gamma \cdot w(T_1)$.*

Proof. The only nodes that are visited more than once (namely twice) in the Eulerian tour E are the nodes of T_1 . Hence each edge e of T_1 is contracted once

and yields weight only $\gamma \cdot w(e)$ in T . Thus the total weight of the constructed TSP tour T is at most $w(C) + \gamma \cdot w(T_1)$. (Also some edges of C are contracted. It is not clear how to get some improvement out of this observation, since these edges could have negligible weight.) $\square$

Now we can estimate the approximation performance of our algorithm.

Theorem 1. *The approximation performance of the algorithm in Figure 1 is bounded by $\frac{1}{1 - \frac{1}{2}(\gamma + \gamma^3)}$. The running time of the algorithm is polynomial.*

Proof. The bound on the approximation performance is shown by induction in the number of nodes. For graphs with one node, the problem is trivial.

Suppose that G has more nodes. By Lemma 1, $\text{TSP}(G_1) + \text{TSP}(G_2) \leq (1 + \gamma^2) \cdot \text{TSP}(G)$. Particularly, $\text{TSP}(G_i) \leq \frac{1}{2}(1 + \gamma^2) \cdot \text{TSP}(G)$ for some $i \in \{1, 2\}$. W.l.o.g. assume that $i = 1$.

By the induction hypothesis,

$$w(T_1) \leq \frac{1}{1 - \frac{1}{2}(\gamma + \gamma^3)} \cdot \text{TSP}(G_1) \leq \frac{1 + \gamma^2}{2 - (\gamma + \gamma^3)} \cdot \text{TSP}(G). \quad (3)$$

Furthermore we can assume that T_1 is indeed the lighter of the two TSP tours, because otherwise (3) also holds for T_2 , as $w(T_2) \leq w(T_1)$.

The TSP tour T computed in step 4 from C and T_1 has weight at most

$$\begin{aligned} w(C) + \gamma \cdot w(T_1) &\leq \left(1 + \gamma \cdot \frac{1 + \gamma^2}{2 - (\gamma + \gamma^3)}\right) \cdot \text{TSP}(G) \\ &\leq \frac{2}{2 - (\gamma + \gamma^3)} \cdot \text{TSP}(G). \end{aligned}$$

by Lemma 2. This proves the claim about the approximation performance.

Let $S(n)$ denote the worst case running time of the algorithm on instances with n nodes. We have $S(1) = 1$ and

$$S(n) \leq 2 \cdot S(n/2) + O(n^3) \quad \text{for all } n > 1,$$

because each instance is divided into two subproblems of size at most $n/2$. The time for computing the two subinstances is dominated by the time needed to construct the cycle cover C . Thus it is $O(n^3)$, see e.g. [1]. Solving the recurrence, we obtain $S(n) = O(n^3)$. $\square$

The approximation performance shown in Theorem 1 is better than the one obtained by Chandran and Ram [6], if

$$\frac{\gamma}{1 - \gamma} \geq \frac{1}{1 - \frac{1}{2}(\gamma + \gamma^3)} \iff -\frac{\gamma^4}{2} - \frac{\gamma^2}{2} + 2\gamma - 1 \geq 0.$$

The real valued roots of the polynomial $p(\gamma) := -\frac{\gamma^4}{2} - \frac{\gamma^2}{2} + 2\gamma - 1$ can be computed exactly. They are $-\frac{1}{3} - \frac{5^{2/3}}{3(7+3\sqrt{6})^{1/3}} + \frac{1}{3}(5(7+3\sqrt{6}))^{1/3}$ and 1. In particular, $p(\gamma) > 0$ holds for $\gamma \in [0.6507, 1)$. Figure 2 compares the two performances in dependence of γ .

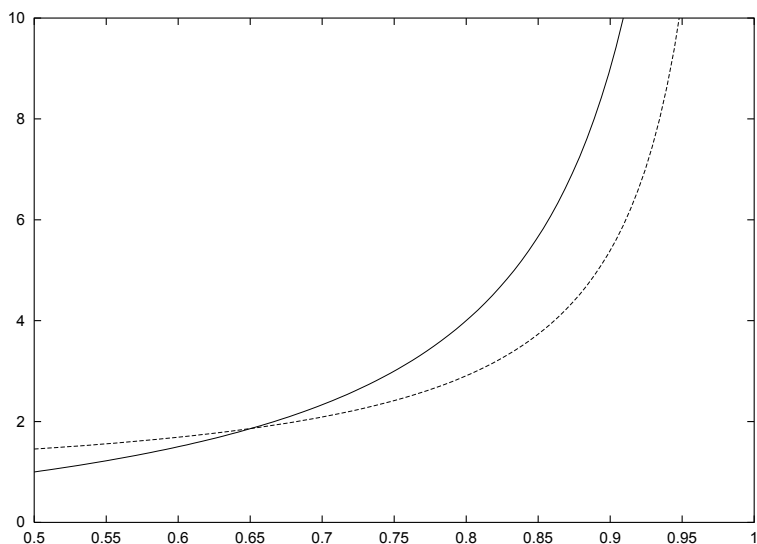


Fig. 2. The approximation performance of the algorithm in Figure 1 (drawn dashed) compared to the one by Chandran and Ram (drawn solid).

Acknowledgment. I would like to thank the anonymous referees for some valuable suggestions that simplified some of the arguments.

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.
2. Thomas Andreae and Hans-Jürgen Bandelt. Performance guarantees for approximation algorithms depending on parameterized triangle inequalities. *SIAM J. Disc. Math.*, 8(1):1–16, 1995.
3. Michael A. Bender and Chandra Chekuri. Performance guarantees for the TSP with a parameterized triangle inequality. In *Proc. 6th Int. Workshop on Algorithms and Data Structures (WADS)*, volume 1663 of *Lecture Notes in Comput. Sci.*, pages 80–85, 1999.
4. Markus Bläser. A new approximation algorithm for asymmetric TSP with triangle inequality. In *Proc. 14th Ann. ACM–SIAM Symp. on Discrete Algorithms (SODA)*, pages 639–647, 2003.
5. J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, and W. Unger. An improved lower bound on the approximability of metric TSP and approximation algorithms for the TSP with sharpened triangle inequality. In *Proc. 17th Int. Symp. on Theoret. Aspects of Comput. Sci. (STACS)*, volume 1770 of *Lecture Notes in Comput. Sci.*, pages 382–394. Springer, 2000.

6. L. Sunil Chandran and L. Shankar Ram. Approximations for ATSP with parametrized triangle inequality. In *Proc. 19th Int. Symp. on Theoret. Aspects of Comput. Sci. (STACS)*, volume 2285 of *Lecture Notes in Comput. Sci.*, pages 227–237, 2002.
7. Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, page 441. Academic Press, 1976.
8. A. M. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12(1):23–39, 1982.

An Improved Approximation Algorithm for Vertex Cover with Hard Capacities (Extended Abstract)

Rajiv Gandhi^{1*}, Eran Halperin^{2**}, Samir Khuller^{3***}, Guy Kortsarz⁴, and
Aravind Srinivasan^{5†}

¹ Department of Computer Science, University of Maryland, College Park,
MD 20742.

gandhi@cs.umd.edu.

² International Computer Science Institute, Berkeley, CA 94704 and Computer
Science Division, University of California, Berkeley, CA 94720.

eran@cs.berkeley.edu.

³ Department of Computer Science and Institute for Advanced Computer Studies,
University of Maryland, College Park, MD 20742.

samir@cs.umd.edu.

⁴ Department of Computer Science, Rutgers University, Camden, NJ 08102.

guyk@camden.rutgers.edu.

⁵ Department of Computer Science and Institute for Advanced Computer Studies,
University of Maryland, College Park, MD 20742. srin@cs.umd.edu.

Abstract. In this paper we study the capacitated vertex cover problem, a generalization of the well-known vertex cover problem. Given a graph $G = (V, E)$, the goal is to cover all the edges by picking a minimum cover using the vertices. When we pick a vertex, we can cover up to a pre-specified number of edges incident on this vertex (its capacity). The problem is clearly NP-hard as it generalizes the well-known vertex cover problem. Previously, 2-approximation algorithms were developed with the assumption that multiple copies of a vertex may be chosen in the cover. If we are allowed to pick at most a given number of copies of each vertex, then the problem is significantly harder to solve. Chuzhoy and Naor (*Proc. IEEE Symposium on Foundations of Computer Science*, 481–489, 2002) have recently shown that the weighted version of this problem is at least as hard as set cover; they have also developed a 3-approximation algorithm for the unweighted version. We give a 2-approximation algorithm for the unweighted version, improving the Chuzhoy-Naor bound of 3 and matching (up to lower-order terms) the best approximation ratio known for the vertex cover problem.

Keywords and Phrases: Approximation algorithms, capacitated covering, set cover, vertex cover, linear programming, randomized rounding.

* Research supported by NSF Award CCR-9820965.

** Supported in part by NSF grants CCR-9820951 and CCR-0121555 and DARPA cooperative agreement F30602-00-2-0601.

*** Research supported by NSF Award CCR-9820965 and an NSF CAREER Award CCR-9501355.

† Supported in part by NSF Award CCR-0208005.

1 Introduction

The capacitated vertex cover problem can be described as follows. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E . Suppose that w_v denotes the weight of vertex v and k_v denotes the capacity of vertex v (we assume that k_v is an integer). A *capacitated vertex cover* is a function that determines a value $x_v \in \{0, 1, \dots, b_v\}$, $\forall v \in V$ such that there exists an orientation of the edges of G in which the number of edges directed into vertex $v \in V$ is at most $k_v x_v$. (These edges are said to be *covered* by or *assigned* to v .) The *weight* of the cover is $\sum_{v \in V} x_v w_v$. The MINIMUM CAPACITATED VERTEX COVER problem is that of computing a minimum weight capacitated cover. The problem generalizes the MINIMUM WEIGHT VERTEX COVER problem which can be obtained by setting $k_v = |V| - 1$ for every $v \in V$. The main difference is that in vertex cover, by picking a node v in the cover we can cover all edges incident to v , and in this problem we can only cover a subset of at most k_v edges incident to node v .

Guha *et al.* [8] studied the version of the problem in which b_v is unbounded. They obtain a 2-approximation algorithm using the primal-dual method. They also gave a 4-approximate solution using LP-rounding. Gandhi *et al.* [7] gave a 2-approximate solution using LP-rounding for the same problem.

The problem becomes significantly harder when b_v is specified for each vertex. For arbitrary weights on the vertices the problem is at least as hard to approximate as the set cover problem for which it is known that an approximation guarantee of $(1 - \epsilon) \ln n$ will imply that $NP \subseteq DTIME[n^{\log \log n}]$. For the case when $w_v = 1$, for all $v \in V$, Chuzhoy and Naor [5] gave a nice 3-approximation algorithm for this problem in polynomial time. Their algorithm uses randomized LP-rounding with alterations. In this paper, we modify the algorithm of Chuzhoy and Naor in two crucial ways to obtain a 2-approximate solution. This is in a sense the “best ratio” possible at the moment, as 2 is also the best ratio known for the simpler vertex-cover problem. We add a pre-processing step in which we make certain capacity-1 vertices ineffective by making their capacities 0. We also modify their alteration step in an important way that helps us to bound the cost of the alteration step in a better way and changes the algorithm.

Related work: The best-known approximation algorithms for the vertex cover problem achieve an approximation ratio of $(2 - o(1))$ for arbitrary graphs [1, 10, 11]. A nice overview of the work on this problem is presented in [13].

The vertex cover is a special case of the set-cover problem that requires to select a minimum number (or minimum cost) collection of subsets that cover the entire universe. The set-cover problem with hard capacities generalizes the set-cover problem in that a set has a capacity bound on the number of elements it can cover. In a seminal paper Johnson [9], gave the first (greedy) logarithmic ratio approximation for the unweighted uncapacitated set cover problem. This was generalized by Chvátal [3] to the weighted uncapacitated case, and further generalized by Dobson [6] to approximating with logarithmic ratio the integer linear program $\min c \cdot x$ subject to $Ax \geq b$ with all the entries in A nonnegative. A much more general result is given by Wolsey [18], giving a logarithmic

ratio approximation algorithm for submodular cover problems. Both the vertex cover problem with hard capacities, and set cover problem with hard capacities are an example of a submodular cover problem. Hence [18] gave the first nontrivial approximation for both problems. See also the work by Bar-Ilan et. al. [2], for a generalization of the method including, e.g., generalization of the set-cover problem with hard capacities problem, facility location problems under flow constraints and the 2-layered facility location problem (without triangle inequality) under hard capacity constraints.

Indeed, a closely related problem to set cover with hard capacities, is facility location with hard capacities. In this problem, we are given a set of facilities F and a set of clients C . There is a cost function $d: L \rightarrow F$ which defines the cost of assigning a client to a facility. Each facility $f \in F$ has a cost w_f , a bound b_f denoting the number of available copies of f and capacity k_f denoting the maximum number of clients that can be assigned to an open facility. Each client i has demand g_i . The goal is to open facilities so that each client can be assigned to some open facility. The objective is to minimize the sum of cost of open facilities and the cost of assigning the clients to them. A logarithmic greedy approximation problem for the uncapacitated case appears in [12] and for the capacitated case and some generalizations in [2]. Slightly improved (still logarithmic) bounds for the uncapacitated case are given in [19] using randomized methods.

There has been a lot of work on metric facility location (see [17] for details). For the metric facility location problem with hard capacities, Pál, Tardos and Wexler [16] gave a $(9 + \epsilon)$ -approximation algorithm using local search.

Research has also been conducted on the multi-set multi-cover problem. In this problem, the input sets are actually multi-sets, i.e., an element can appear in a set more than once. The problem with unbounded set capacities can be defined as the following IP: $\min\{w^T x \mid Ax \geq d, 0 \leq x \leq b, x \in Z\}$. The LP has an unbounded Integrality gap. Dobson [6] gave a greedy algorithm achieving a guarantee of $H(\max_{1 \leq j \leq n} A_{ij})$. Recently, Carr *et al.* [4] gave a p -approximation algorithm, where p denotes the maximum number of variables in any constraint. Their algorithm is based on a stronger LP-relaxation. Kolliopoulos and Young [14] obtained an $O(\log n)$ approximation algorithm.

Remark: We will solve the special case of the vertex cover problem in which at most one copy of each vertex can be used. As in [5], our algorithm can be easily extended to the general case where multiple copies of each vertex can be used.

2 IP Formulation and Relaxation

A linear integer program (IP) for the problem can be written as follows (as in [8]).

In this formulation, $y_{ev} = 1$ if and only if the edge $e \in E$ is covered by vertex v . Clearly, the values of x in a feasible solution correspond to a capacitated cover. While we do not really need the constraint $x_v \geq y_{ev}$ $v \in e \in E$ for the IP formulation, this constraint will play an important role in the relaxation. (In fact, without this constraint there is a large Integrality gap between the best

fractional and integral solutions). For any vertex v , let $E(v)$ denote the set of edges incident on v .

$$\begin{aligned}
 & \text{Minimize } \sum_v x_v \\
 & y_{eu} + y_{ev} = 1 \quad e = \{u, v\} \in E, \\
 & k_v x_v - \sum_{e \in E(v)} y_{ev} \geq 0 \quad v \in V, \\
 & x_v \geq y_{ev} \quad v \in e \in E, \\
 & y_{ev} \in \{0, 1\} \quad v \in e \in E, \\
 & x_v \in \{0, 1\} \quad v \in V.
 \end{aligned} \tag{1}$$

In the relaxation to a linear program, we restrict $y_{ev} \geq 0$ and $0 \leq x_v \leq 1$.

3 Algorithm

Our algorithm differs from the Chuzhoy-Naor algorithm in the following two ways. We perform a pre-processing step (Step 1) in which we make some of the capacity-1 vertices ineffective by making their capacities 0. Our alteration step (Step 5) is also different than the alteration step used in the Chuzhoy-Naor algorithm. Both these changes are crucial to our analysis. Let (x', y') be a solution in which x' is a binary vector and y' is fractional. Once we have such a solution, we can convert it to a solution (x', y'') in which y'' is integral (Step 6).

1. **Pre-Processing.** Keep “removing” capacity 1 vertices (make their capacity 0) from the graph until we have a graph in which removing any capacity 1 vertex will result in an infeasible solution. Include the remaining capacity 1 vertices in the cover (add the “ $x_v = 1$ ” constraint in the LP for each such vertex v). Checking whether a graph, $G = (V, E)$, has a feasible solution or not can be done as follows. Let $B = (A_1, A_2, F)$ be a bipartite graph in which each node in A_1 represents an edge in E and each vertex in A_2 represents a vertex in V . An edge $(e, v) \in F$ iff in G , edge e is incident to vertex v . Construct a flow network in which the source is connected to all vertices in A_1 and each vertex in A_2 is connected to the sink. The capacities of the edges in F is 1. The capacities of the edges emanating from the source are all 1. The capacity of an edge from any node $v \in A_2$ to the sink is k_v . G has a feasible solution iff the maximum flow value from the source to the sink is $|E|$.
2. **LP Solution.** Solve the LP relaxation (that has the additional constraint $x_v = 1$ for each capacity-1 vertex v that survived the pre-processing step) optimally. To facilitate the discussion of the remainder of the algorithm let us introduce some notation.

$$U = \{u | x_u \geq 1/2\}.$$

$$\bar{U} = V \setminus U.$$

$$E' = \{(u, v) | u \in U, v \in \bar{U}\}.$$

$$\forall u \in V, E'(u) = E' \cap E(u) \text{ and } d_u = |E'(u)|.$$

$$\forall u \in U, \epsilon_u = 1 - x_u, 0 \leq \epsilon_u \leq 1/2.$$

3. **Partial Cover.** Include all vertices of U in the cover, i.e., $\forall u \in U, x'_u = 1$. Note that all capacity-1 vertices belong to U . For any edge $e = (u, v) \in E \setminus E'$, set $y'_{eu} = y_{eu}$ and $y'_{ev} = y_{ev}$. The contribution of $u \in U$ towards covering edge $e = (u, v) \in E'(u)$ is at least $y'_{eu} = y_{eu}/x_u \geq (1 - y_{ev})/(1 - \epsilon_u)$. For each $e = (u, v) \in E'(u)$, let $h_{ev} = 1 - (1 - y_{ev})/(1 - \epsilon_u) = (y_{ev} - \epsilon_u)/(1 - \epsilon_u)$. To cover all the edges in $E'(u)$ fractionally, we are going to need an additional coverage of $h_u = \sum_{e=(u,v) \in E'(u)} h_{ev}$. In the following steps we will get the necessary additional coverage from vertices in $\bar{U}$. Note that there are no edges within $\bar{U}$.
4. **Randomized Rounding.** Round each vertex $v \in \bar{U}$ to 1 *independently* with probability $2x_v$. Let I be the set of vertices that are rounded to 1 in this step. For each edge $e = (u, v) \in E'$ such that $v \in I$, let $y'_{ev} = y_{ev}/x_v$ be the contribution of v towards covering e . By constraint (1), $\sum_{e \in E(v)} y_{ev}/x_v = \sum_{e \in E(v)} y'_{ev} \leq k_v$.
5. **Alteration.** Let P be the set of vertices in U that still need some help from vertices in $\bar{U}$, i.e., $P = \{u \in U \mid \sum_{e=(u,v), v \in I} y'_{ev} < h_u\}$. In this step, we will choose a set of vertices $I' \subseteq \bar{U} \setminus I$, such that $\forall u \in P, \sum_{e=(u,v), v \in I \cup I'} y'_{ev} \geq h_u$, where for each vertex $v \in I'$, y'_{ev} is set according to step (c) below. For each vertex $u \in P$, we define a set of vertices $helper(u)$. Each vertex in $helper(u)$ contributes towards h_u . Each vertex in I' belongs to exactly one such set. Initially, $I' \leftarrow \emptyset$ and $helper(u) \leftarrow \emptyset, \forall u \in P$. We perform the following steps until P is empty.
- Pick a vertex $u \in P$.
 - Consider any edge (u, v) such that $v \in \bar{U} \setminus (I \cup I')$. $helper(u) \leftarrow helper(u) \cup \{v\}$. $I' \leftarrow I' \cup \{v\}$. Let $P'_v = \{w \in P \mid w \neq u, e' = (w, v) \in E'\}$.
 - For each $w \in P'_v$ and $e' = (w, v)$, set $y'_{e'v} = y_{e'v}$ and set $y'_{e'w} = 1 - y'_{e'v}$. Set y'_{ev} , where $e = (u, v)$, to be the minimum of 1 and the remaining capacity of v . Set $y'_{eu} = 1 - y'_{ev}$.
 - For each vertex $w \in P'_v$, if $\sum_{e=(w,a), a \in I \cup I'} y'_{ea} \geq h_w$ remove w from P . For each edge $f = (w, b) \in E'$ such that $b \notin I \cup I'$, set $y'_{fb} = 0$ and $y'_{fw} = 1$.
 - Remove u from P iff $\sum_{e=(u,a), a \in I \cup I'} y'_{ea} \geq h_u$.

Once P is empty, we have a feasible solution (x', y') in which x' is integral and y' may be fractional.

6. **Integral Solution.** At this point x' is a binary vector but y' is fractional. This can be converted to an integral solution using the integrality of flows property on a flow network. The flow network is exactly the same as the one constructed in Step 1 with the difference that the capacity of an edge going from a node representing $v \in V$, to the sink is $k_v x'_v$.

4 Analysis

In Step 5 of the algorithm we choose the set of vertices I' and include them as part of our cover. We have to account for the cost of these vertices. Note that for

each vertex $v \in I'$ there is exactly one vertex $u \in P$, such that $v \in \text{helper}(u)$. We will charge u the cost of adding v to our solution. Note that in the LP solution the cost of vertex u is $x_u = 1 - \epsilon_u$. In our solution, vertex $u \in U$ pays for itself and for the vertices in $\text{helper}(u)$. We will show that the total expected charge on u due to vertices in $\text{helper}(u)$ is at most $1 - 2\epsilon_u$. Thus, the total expected cost of vertex u is $2 - 2\epsilon_u = 2x_u$. Also, the total expected size of I is $\sum_{v \in \bar{U}} 2x_v$. Thus we obtain a 2-approximation in expectation, by using the linearity of expectation.

Theorem 1. *Let Cost be the random variable that represents the cost of our vertex cover, C . Then $\mathbf{E}[\text{Cost}] \leq 2\text{OPT}$.*

Our primary goal will be to show that for any $u \in U$, the total expected charge on u due to vertices in $\text{helper}(u)$ is at most $1 - 2\epsilon_u$. Before doing so, we will first show that our preprocessing step (of removing capacity 1 vertices) is justifiable:

Lemma 1. *Let R be the set of vertices of capacity 1 removed from a graph G_o in the pre-processing step (Step 1). Let G_n be the new graph that has the same vertices and edges as G_o except that the capacities of the vertices in R is reduced to 0. Let $\text{OPT}(G_o)$ and $\text{OPT}(G_n)$ represent the optimal solutions in G_o and G_n respectively. Then $\text{OPT}(G_o) = \text{OPT}(G_n)$. This implies that the LP solution to G_n is a lower bound on $\text{OPT}(G_o)$.*

Proof. Let $\text{OPT}(G_o)$ be an optimal solution to G_o that uses a minimum number of vertices from R . If $\text{OPT}(G_o) \cap R = \emptyset$ then the claim follows trivially. Now consider the case when $\text{OPT}(G_o) \cap R \neq \emptyset$. Let $v \in \text{OPT}(G_o) \cap R$. Construct a directed graph H having the same vertex set as G_o . Include an edge (a, b) in H iff edge (a, b) in G_o is covered by a in $\text{OPT}(G_o)$ and by b in $\text{OPT}(G_n)$. H may contain some cycles. Since v has in-degree zero in H , v cannot be part of any cycle. Contract every cycle of H . Now consider a maximal path, Q , starting from v . Let w be the last vertex in the path. Note that w does not have any outgoing edges, otherwise Q is not maximal. Consider the solution $\text{OPT}(G_o) \setminus \{v\} \cup \{w\}$ in which the edges of Q have the same assignment as in $\text{OPT}(G_n)$. We will now show that this new assignment does not violate capacity constraints of any of the vertices. The only vertices that are affected are the vertices in Q . The assignment of edges to all other vertices remain the same as in $\text{OPT}(G_o)$. In H , since w has one incoming edge and no outgoing edges, w covers one more edge in $\text{OPT}(G_n)$ than it covers in $\text{OPT}(G_o)$. Since $w \notin R$, the capacity of w is the same in G_o and in G_n . Thus w covers at most $k_w - 1$ edges in $\text{OPT}(G_o)$. Thus in $\text{OPT}(G_o)$, w has a spare capacity of at least 1 that it uses to cover its incoming edge in Q . Every other vertex whose covering is different than in $\text{OPT}(G_o)$ is an internal vertex of Q . Each such vertex uncovers one edge (outgoing edge in Q) and covers a new edge (incoming edge in Q), hence its capacity constraints are not affected. This cost of this solution is the same as $\text{OPT}(G_o)$ and it uses one fewer vertex from R , thus contradicting the assumption that $\text{OPT}(G_o)$ used minimum number of vertices from R .

Lemma 2. *Every vertex in $\bar{U}$ has capacity at least 2.*

Proof. If any vertex v has capacity 1 then $x_v = 1$ (Step 1). Hence, all capacity 1 vertices belong to U .

Lemma 3. *Let $e = (u, v)$ and $v \in \text{helper}(u)$. Then $y'_{ev} = 1$. In other words, vertex v contributes 1 towards h_u .*

Proof. Since $v \in \text{helper}(u)$, $y'_{ev} = \min\{1, k_v - \sum_{f \in E'(v) \setminus \{e\}} y'_{fv}\}$. To prove our claim, we must show that $k_v - \sum_{f \in E'(v) \setminus \{e\}} y'_{fv} \geq 1$. L.H.S evaluates to $k_v - \sum_{f \in E'(v) \setminus \{e\}} y_{fv} \geq k_v - \sum_{f \in E'(v)} y_{fv} = k_v - \sum_{f \in E(v)} y_{fv}$. Using constraint (1), we get L.H.S $\geq k_v - k_v x_v \geq k_v - k_v/2 = k_v/2 \geq 1$.

Lemma 4. *Each vertex $u \in P$ is charged at most $\lceil h_u \rceil$ by vertices in I' , i.e., $|\text{helper}(u)| \leq \lceil h_u \rceil$.*

Remark: Observe that if $x_u = 1/2$, we are done since $E'(u) = \emptyset$. Hence, whenever we need to calculate the expected cost of a vertex $u \in U$, we can assume $0 \leq \epsilon_u < 1/2$.

Lemma 5. *Let $u \in U$. Let Z_u be the random variable that denotes the help received by vertex u in Step 4 of the algorithm, i.e., $Z_u = \sum_{e=(u,v):v \in I} y_{ev}/x_v$. Then $\mu_u = \mathbf{E}[Z_u] \geq 2h_u(1 - \epsilon_u)/(1 - 2\epsilon_u)$.*

Proof. Recall that $h_u = \sum_{e=(u,v) \in E'(u)} (y_{ev} - \epsilon_u)/(1 - \epsilon_u)$ and $d_u = |E'(u)|$. By definition of expectation, we have

$$\begin{aligned} \mu_u &= \sum_{e=(u,v) \in E'(u)} (y_{ev}/x_v) 2x_v \\ &= 2 \sum_{e=(u,v) \in E'(u)} y_{ev} \end{aligned} \quad (2)$$

$$\begin{aligned} &= 2(1 - \epsilon_u)h_u + 2\epsilon_u d_u \\ &= 2h_u + 2\epsilon_u(d_u - h_u) \end{aligned} \quad (3)$$

Since $\mu_u \leq d_u$, we have $d_u \geq 2h_u + 2\epsilon_u(d_u - h_u)$. This gives us $d_u - h_u \geq h_u/(1 - 2\epsilon_u)$. Combining this inequality with (3), we get $\mu_u \geq 2h_u + 2\epsilon_u h_u/(1 - 2\epsilon_u) = 2h_u(1 - \epsilon_u)/(1 - 2\epsilon_u)$.

Notation: From now on, let $\exp(x)$ denote e^x .

Lemma 6. *If X_u is the random variable denoting the charge on a vertex $u \in U$ due to vertices in I' , then $\mathbf{E}[X_u] \leq \sum_{i=0}^{\lceil h_u \rceil} (\exp(-\delta_i)/(1 - \delta_i)^{(1-\delta_i)})^{\mu_u}$, for some $\delta_i \in [\frac{1}{2(1-\epsilon)} + \frac{i(1-2\epsilon)}{2h_u(1-\epsilon)}, 1]$ and $\mu_u = \mathbf{E}[Z_u]$. When $\delta_i = 1$, we evaluate the summand in the limit as $\delta_i \rightarrow 1$: this limit is $\exp(-\mu_u)$.*

Proof. Note that X_u can be any integer between 0 and $\lceil h_u \rceil$. By definition of expectation, we have $\mathbf{E}[X_u] = \sum_{i=1}^{\lceil h_u \rceil} i \Pr X_u = i = \sum_{i=0}^{\lceil h_u \rceil} \Pr X_u \geq i + 1 \leq \sum_{i=0}^{\lceil h_u \rceil} \Pr Z_u \leq \lceil h_u \rceil - (i + 1)$. Thus we get

$$\mathbf{E}[X_u] \leq \sum_{i=0}^{\lceil h_u \rceil} \Pr Z_u \leq h_u - i \quad (4)$$

Since Z_u is a sum of independent random variables each lying in $[0, 1]$, we get using the Chernoff-Hoeffding bound that

$$\Pr Z_u \leq \mu_u(1 - \delta_i) \leq \left(\exp(-\delta_i)/(1 - \delta_i)^{(1-\delta_i)} \right)^{\mu_u}$$

The value of δ_i can be obtained as follows.

$$1 - \delta_i = \frac{h_u - i}{\mu_u} \quad (5)$$

Combining (5) with Lemma 5, we get $\delta_i \geq \frac{1}{2(1-\epsilon_u)} + \frac{i(1-2\epsilon_u)}{2h_u(1-\epsilon_u)}$.

Lemma 7. For $0 \leq \delta < 1$, the function $f(\delta) = 1/(1-\delta)^{(1-\delta)}$ attains a maximum value of $\exp(1/e)$ at $\delta = 1 - 1/e$.

Lemma 8. For any vertex $u \in U$, if $h_u \geq 2$ then $\mathbf{E}[X_u] \leq 1 - 2\epsilon_u$.

Proof. From Lemma 6 and Lemma 7, we get $\mathbf{E}[X_u] \leq \sum_{i=0}^{\lfloor h_u \rfloor} (\exp(1/e - \delta_i))^{\mu_u}$. From Lemma 6, we know that $\forall i \geq 0$, $\delta_i \geq 1/2$. Hence, $1/e - \delta_i$ is always negative. Also, μ_u is always positive. Hence, the summand is maximized when μ_u and δ_i are minimized. Thus, we get

$$\begin{aligned} \mathbf{E}[X_u] &\leq \sum_{i=0}^{\lfloor h_u \rfloor} \left(\exp \left(\frac{1}{e} - \frac{h_u + i(1-2\epsilon_u)}{2h_u(1-\epsilon_u)} \right) \right)^{\frac{2h_u(1-\epsilon_u)}{1-2\epsilon_u}} \\ &= \sum_{i=0}^{\lfloor h_u \rfloor} \exp(p - i) \quad \left(\text{where } p = \frac{2h_u(1-\epsilon_u)}{e(1-2\epsilon_u)} - \frac{h_u}{1-2\epsilon_u} \right) \\ &\leq \frac{e}{e-1} \cdot \exp(p) \cdot (1 - \exp(-h_u - 1)). \end{aligned} \quad (6)$$

We will now show that $f(h_u) = \exp(p)(1 - \exp(-h_u - 1))$ is a decreasing function of h_u . $f'(h_u) = \exp(p) \left(\frac{2(1-\epsilon_u)}{e(1-2\epsilon_u)} - \frac{1}{1-2\epsilon_u} \right) - \exp(p - h_u - 1) \left(\frac{2(1-\epsilon_u)}{e(1-2\epsilon_u)} - \frac{1}{1-2\epsilon_u} - 1 \right)$. The expression $\left(\frac{2(1-\epsilon_u)}{e(1-2\epsilon_u)} - \frac{1}{1-2\epsilon_u} \right)$ is negative since $2(1-\epsilon_u)/e < 1$. Since the first term dominates the second term $f'(h_u)$ is negative. Thus $f(h_u)$ is decreasing and is maximized when h_u is minimized. When $h_u = 2$,

$$p = \frac{4(1-\epsilon_u)}{e(1-2\epsilon_u)} - \frac{2}{1-2\epsilon_u} = \frac{2}{e} - \frac{K_1}{1-2\epsilon_u},$$

where K_1 is the positive constant $(2e - 2)/e$. Thus, from (6), it is sufficient to show that

$$\forall \epsilon \in [0, 1/2), \quad K_2 \cdot \exp(-K_1/(1-2\epsilon)) \leq 1 - 2\epsilon,$$

where K_2 is the constant $\frac{e^2 + e + 1}{e^2} \cdot \exp(2/e)$. Making the substitution $\psi = \frac{1}{1-2\epsilon}$ and taking the natural logarithm on both sides, it suffices to show:

$$\forall \psi \geq 1, \quad -\ln \psi + K_1 \psi - \ln K_2 \geq 0.$$

The inequality holds for $\psi = 1$. Also, for $\psi > 1$, the function $\psi \mapsto -\ln \psi + K_1 \psi - \ln K_2$ has derivative $K_1 - 1/\psi$; since $K_1 = 2 - 2/e$ is greater than 1, the function increases for $\psi > 1$, and so we are done.

Lemma 9. *For any vertex $u \in U$, if $0 < h_u < 1$ then $\mathbf{E}[X_u] \leq 1 - 2\epsilon_u$.*

Proof. Recall that $d_u = |E'(u)|$. Consider the case when $d_u = 1$. Let $e = (u, v) \in E'(u)$. Thus, $h_u \leq \epsilon_u \leq y_{ev} \leq x_v$. Thus, with a probability of $2x_v \geq 2\epsilon_u$, $v \in I$ and u receives the help h_u . Hence, the probability with which u participates in Step 5, i.e., $u \in P$, is $1 - 2\epsilon_u$. In that case, $|\text{helper}(u)| \leq 1$. Hence, $\mathbf{E}[X_u] \leq 1 - 2\epsilon_u$. Now consider when $d_u = 2$. Let $e_1 = (u, v)$ and $e_2 = (u, w)$ be the edges in $E'(u)$. Since $\mu_u = 2(y_{e_1v} + y_{e_2w}) \geq 4\epsilon_u$. From (3), we know that $\mu_u \geq 2h_u$. Hence, either $h_{e_1v} \geq h_u$ or $h_{e_2w} \geq h_u$. Without loss of generality, let $h_{e_1v} \geq h_u$. Since $x_v \geq \epsilon_u$, the probability of u receiving help of h_u in the randomized rounding step (Step 4) is at least $2\epsilon_u$. Hence, u participates in Step 5 (Alteration Step) of the algorithm with a probability of at most $2(1 - \epsilon_u)$. Thus, we get $\mathbf{E}[X_u] \leq 1 - 2\epsilon_u$. For the remainder of the lemma we assume that $d_u \geq 3$.

From inequality (4), we know that $\mathbf{E}[X_u] \leq \Pr Z_u \leq h_u$. Recall that Z_u is the random variable that represents the amount of help that u receives in Step 4 (Randomized Rounding) of the algorithm. Let $Z_u = \sum_{e=(u,v) \in E'(u)} Z_{ev}$, where Z_{ev} is the random variable that denotes the amount of help that v provides to u in Step 4 of the algorithm. Next suppose X is a random variable with mean μ and variance σ^2 ; suppose $a > 0$. Then, the well-known Chebyshev's inequality states that $\Pr |X - \mu| \geq a$ is at most σ^2/a^2 . We will need stronger tail bounds than this, but only on X 's deviation *below* its mean. The Chebyshev-Cantelli inequality shows that

$$\Pr X \leq \mu - a \leq \sigma^2/(\sigma^2 + a^2). \quad (7)$$

Define

$$y_u = \frac{\sum_{e=(u,v) \in E'(u)} y_{ev}}{d_u},$$

and note that

$$\epsilon_u \leq y_u \leq 1/2. \quad (8)$$

We will use (7) to bound $\Pr Z_u \leq h_u$. Thus, setting $\mu_u - a = h_u$ and using (2) we get $a = \mu_u - h_u = 2 \sum_{e=(u,v) \in E'(u)} y_{ev} - \sum_{e=(u,v) \in E'(u)} (y_{ev} - \epsilon_u)/(1 - \epsilon_u) = 2d_u y_u - (d_u y_u - d_u \epsilon_u)/(1 - \epsilon_u)$. This gives us

$$a = d_u \left(2y_u - \frac{y_u - \epsilon_u}{1 - \epsilon_u} \right) \quad (9)$$

Let σ_u^2 and σ_{ev}^2 denote the variance of the random variables Z_u and Z_{ev} respectively. Since Z_u is the sum of the *independent* random variables Z_{ev} , we get $\sigma_u^2 = \sum_{e=(u,v) \in E'(u)} \sigma_{ev}^2 = \sum_{e=(u,v) \in E'(u)} \mathbf{E}[Z_{ev}^2] - \mathbf{E}[Z_{ev}]^2$. This gives us

$$\sigma_u^2 = \sum_{e=(u,v) \in E'(u)} \frac{2y_{ev}^2}{x_v} - 4y_{ev}^2 \quad (10)$$

For a fixed a , the R.H.S. of (7) is maximized when σ^2 is maximized. We know that $\epsilon_u \leq y_{ev} \leq x_v < 1/2$. The R.H.S. of (10) is maximized when x_v is minimized.

Also, for a fixed value of $\sum_{e=u, v \in E'(u)} y_{ev}$, $\sum_{e=(u,v) \in E'(u)} y_{ev}^2$ is minimized when $y_{ev} = y_{e'v'} = y_u$, $\forall e = (u, v) \in E'(u)$ and $e' = (u, v') \in E'(u)$. Note that we are not changing the value of $\sum_{e=(u,v) \in E'(u)} y_{ev}$. Substituting $y_{ev} = y_u$ and $x_v = y_{ev} = y_u$ in the above inequality, we get $\sigma_u^2 \leq \sum_{e=(u,v) \in E'(u)} 2y_u(1-2y_u) \leq 2d_u y_u(1-2y_u)$. Using (7), the value of a from (9), and the value of σ_u^2 obtained above, we get $\mathbf{E}[X_u] \leq \Pr Z_u \leq h_u \leq \sigma_u^2/(\sigma_u^2 + a^2) \leq (2d_u y_u(1-2y_u))/(2d_u y_u(1-2y_u) + d_u^2 (2y_u - (y_u - \epsilon_u)/(1 - \epsilon_u))^2)$. This gives us

$$\mathbf{E}[X_u] \leq \frac{2y_u(1-2y_u)}{2y_u(1-2y_u) + 3\left(2y_u - \frac{y_u - \epsilon_u}{1 - \epsilon_u}\right)^2} \quad (11)$$

We will analyze the cost by considering the following two cases.

Case I: $\epsilon > 3y_u/4$.

We would like to upper-bound the value of $(y_u - \epsilon_u)/(1 - \epsilon_u)$ in (11). For $\epsilon_u \leq y_u < 4\epsilon_u/3$, we will calculate a value $c \in [0, 1]$ such that $(y_u - \epsilon_u)/(1 - \epsilon_u) \leq cy_u$. $c \geq (y_u - \epsilon_u)/(y_u(1 - \epsilon_u)) = 1/(1 - \epsilon_u) - \epsilon_u/(y_u(1 - \epsilon_u)) \geq 1/(1 - \epsilon_u) - \epsilon_u/((4\epsilon_u/3)(1 - \epsilon_u)) = 1/(1 - \epsilon_u) - 3/(4(1 - \epsilon_u)) = 1/(4(1 - \epsilon_u)) \geq 1/(4(1 - 1/2)) = 1/2$. Thus, substituting the value of $(y_u - \epsilon_u)/(1 - \epsilon_u)$ as $y_u/2$ in (11), we get $\mathbf{E}[X_u] \leq (2y_u(1-2y_u))/(2y_u(1-2y_u) + 3(2y_u - y_u/2)^2) = (2y_u(1-2y_u))/(2y_u - 4y_u^2 + (27y_u^2/4)) \leq (2y_u(1-2y_u))/2y_u = 1 - 2y_u$.

Case II: $\epsilon \leq 3y_u/4$.

We want to show that $\mathbf{E}[X_u] \leq 1 - 2\epsilon_u$. Thus, it is sufficient to show that R.H.S of (11) is at most $1 - 2\epsilon_u$ which means that it is sufficient to show that

$$2y_u(1-2y_u) - 2y_u(1-2y_u)(1-2\epsilon_u) \leq 3\left(2y_u - \frac{y_u - \epsilon_u}{1 - \epsilon_u}\right)^2 (1-2\epsilon_u) \quad (12)$$

We will consider the L.H.S and R.H.S of (12) separately. L.H.S = $2y_u(1-2y_u) - 2y_u(1-2y_u)(1-2\epsilon_u) = 2y_u(1-2y_u)(2\epsilon_u) = 4\epsilon_u y_u(1-2y_u)$. Since $\epsilon_u \leq 3y_u/4$, we get

$$\text{L.H.S} \leq 3y_u^2(1-2y_u) \quad (13)$$

R.H.S evaluates to $3(y_u/(1 - \epsilon_u) + \epsilon_u(1 - 2y_u)/(1 - \epsilon_u))^2 (1 - 2\epsilon_u)$. Since $y_u < 1/2$, we get

$$\text{R.H.S} \geq 3y_u^2(1-2\epsilon_u) \quad (14)$$

From (13) and (14) we conclude that L.H.S $\leq$ R.H.S and $\mathbf{E}[X_u] \leq 1 - 2\epsilon_u$.

Lemma 10. For any vertex $u \in U$, if $1 \leq h_u < 2$ then $\mathbf{E}[X_u] \leq 1 - 2\epsilon_u$.

Proof. We will use the notation $d_u, \mu_u, y_u, \sigma_u^2$ etc. as in the proof of Lemma 9. As in that proof, we have $\sigma_u^2 \leq 2d_u y_u(1-2y_u)$. Recall that $h_u = d_u(y_u - \epsilon_u)/(1 - \epsilon_u)$. Thus, by Chebyshev-Cantelli,

$$\begin{aligned} \mathbf{E}[X_u] &\leq \Pr Z_u \leq h_u - 1 + \Pr Z_u \leq h_u \\ &\leq \frac{2d_u y_u(1-2y_u)}{2d_u y_u(1-2y_u) + (\mu_u - h_u + 1)^2} + \frac{2d_u y_u(1-2y_u)}{2d_u y_u(1-2y_u) + (\mu_u - h_u)^2} \end{aligned}$$

$$\begin{aligned}
&= \frac{2d_u y_u (1 - 2y_u)}{2d_u y_u (1 - 2y_u) + (2d_u y_u - \frac{d_u(y_u - \epsilon_u)}{1 - \epsilon_u} + 1)^2} + \\
&\quad \frac{2y_u (1 - 2y_u)}{2y_u (1 - 2y_u) + d_u(2y_u - \frac{y_u - \epsilon_u}{1 - \epsilon_u})^2}
\end{aligned} \tag{15}$$

Now fix ϵ_u and y_u arbitrarily (subject to the constraints $0 \leq \epsilon_u < y_u \leq 1/2$), and consider an adversary who wishes to maximize (15) subject to the constraint that d_u is a real number for which $d_u(y_u - \epsilon_u)/(1 - \epsilon_u) \geq 1$. It is then sufficient to show that the maximum value (achievable by the adversary) is at most $1 - 2\epsilon_u$; we will do so now. It can be shown that (15) is maximized when $d_u(y_u - \epsilon_u)/(1 - \epsilon_u) = 1$. Making the substitution $z = 2d_u y_u$, we make some observations. Since $z = 2y_u(1 - \epsilon_u)/(y_u - \epsilon_u)$ where $0 \leq \epsilon_u < y_u$, we have $z \geq 2$; also, $\epsilon_u = \frac{y_u(z-2)}{z-2y_u}$. So, to show that (15) is at most $1 - 2\epsilon_u$, we need to show that $\frac{2d_u y_u (1 - 2y_u)}{2d_u y_u (1 - 2y_u) + (2d_u y_u)^2} + \frac{2d_u y_u (1 - 2y_u)}{2d_u y_u (1 - 2y_u) + (2d_u y_u - 1)^2} \leq 1 - 2\epsilon_u$; i.e., $\frac{1 - 2y_u}{1 - 2y_u + z} + \frac{z(1 - 2y_u)}{z(1 - 2y_u) + (z - 1)^2} \leq 1 - \frac{2y_u(z-2)}{z - 2y_u}$. That is, we want to show that

$$\frac{z}{1 - 2y_u + z} - \frac{z(1 - 2y_u)}{z(1 - 2y_u) + (z - 1)^2} \geq \frac{2y_u(z - 2)}{z - 2y_u}. \tag{16}$$

Substitute $p = 1 - 2y_u$, and note that $p \in [0, 1]$. Simplifying (16), we want to show that $z \cdot (z + p - 1) \cdot ((z - 1)^2 - p^2) \geq (1 - p) \cdot (z - 2) \cdot (z + p) \cdot ((z - 1)^2 + pz)$. Since $z \geq 2$ and $0 \leq p \leq 1$, all the factors in this last inequality are non-negative; so, it suffices to show that $z \geq (1 - p) \cdot (z + p)$, and $(z + p - 1) \cdot ((z - 1)^2 - p^2) \geq (z - 2) \cdot ((z - 1)^2 + pz)$. The first inequality reduces to $zp \geq p(1 - p)$, which is true since $z \geq 2 > 1 - p$. The second inequality reduces to $-p^3 - p^2(z - 1) + p + (z - 1)^2 \geq 0$. For a fixed p , the derivative of the l.h.s. (w.r.t. z) is easily seen to be non-negative for $z \geq 2$. Thus, it suffices to check that $-p^3 - p^2(z - 1) + p + (z - 1)^2 \geq 0$ is non-negative when $z = 2$, which follows from the fact that $p \in [0, 1]$.

Acknowledgments. We thank Seffi Naor for helpful discussions. Thanks also to the ICALP referees for their useful comments.

References

1. R. Bar-Yehuda and S. Even. *A Local-Ratio Theorem for Approximating The Weighted Vertex Cover Problem*. Annals of Discrete Mathematics, 25:27–45, 1985.
2. J. Bar-Ilan, G. Kortsarz and D. Peleg. *Generalized submodular cover problems and applications*. Theoretical Computer Science, 250, pages 179–200, 2001.
3. V. Chvátal. *A Greedy Heuristic for the Set Covering Problem*. Mathematics of Operations Research, vol. 4, No 3, pages 233–235, 1979.
4. R. D. Carr, L. K. Fleischer, V. J. Leung and C. A. Phillips. *Strengthening Integrality Gaps For Capacitated Network Design and Covering Problems*. In Proc. of the 11th ACM-SIAM Symposium on Discrete Algorithms, pages 106–115, 2000.
5. J. Chuzhoy and J. Naor. *Covering Problems with Hard Capacities*. Proc. of the Forty-Third IEEE Symp. on Foundations of Computer Science, 481–489, 2002.

6. G. Dobson. *Worst Case Analysis of Greedy Heuristics For Integer Programming with Non-Negative Data*. Math. of Operations Research, 7(4):515–531, 1980.
7. R. Gandhi, S. Khuller, S. Parthasarathy and A. Srinivasan. *Dependent Rounding in Bipartite Graphs*. In Proc. of the Forty-Third IEEE Symposium on Foundations of Computer Science, pages 323–332, 2002.
8. S. Guha, R. Hassin, S. Khuller and E. Or. *Capacitated Vertex Covering with Applications*. Proc. ACM-SIAM Symp. on Discrete Algorithms, pages 858–865, 2002.
9. D. S. Johnson, *Approximation algorithms for combinatorial problems*. J. Computer and System Sciences, 9, pages 256–278, 1974.
10. E. Halperin. *Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs*. In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California, pages 329–337, 2000.
11. D. S. Hochbaum. *Approximation Algorithms for the Set Covering and Vertex Cover Problems*. SIAM Journal on Computing, 11:555–556, 1982.
12. D. S. Hochbaum. *Heuristics for the fixed cost median problem*. Mathematical Programming, 22:148–162, 1982.
13. D. S. Hochbaum (editor). *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1996.
14. S. G. Kolliopoulos and N. E. Young. *Tight Approximation Results for General Covering Integer Programs*. In Proc. of the Forty-Second Annual Symposium on Foundations of Computer Science, pages 522–528, 2001.
15. L. Lovász, *On the ratio of optimal integral and fractional covers*. Discrete Math., 13, pages 383–390, 1975.
16. M. Pál, É. Tardos and T. Wexler. *Facility Location with Nonuniform Hard Capacities*. In Proc. Forty-Second Annual Symposium on Foundations of Computer Science, 329–338, 2001.
17. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.
18. L. A. Wolsey. *An analysis of the greedy algorithm for the submodular set covering problem*. Combinatorica, 2:385–393, 1982.
19. N. E. Young. *K-medians, facility location, and the Chernoff-Wald bound*. ACM-SIAM Symposium on Discrete Algorithms, pages 86–95, 2000.

Approximation Schemes for Degree-Restricted MST and Red-Blue Separation Problem

Sanjeev Arora^{1*} and Kevin L. Chang²

¹ Princeton University, Princeton, NJ.
`arora@cs.princeton.edu`.

² Yale University, New Haven, CT.
`kevin.chang@yale.edu`.

Abstract. We develop a quasi-polynomial time approximation scheme for the Euclidean version of the Degree-restricted MST by adapting techniques used previously for approximating TSP. Given n points in the plane, $d = 2$ or 3 , and $\epsilon > 0$, the scheme finds an approximation with cost within $1 + \epsilon$ of the lowest cost spanning tree with the property that all nodes have degree at most d . We also develop a polynomial time approximation scheme for the Euclidean version of the Red-Blue Separation Problem.

1 Introduction

In the *degree-restricted spanning tree problem* we are given n points in $\mathbb{R}^2$ (more generally, $\mathbb{R}^k$) and a degree bound $d \geq 2$ and have to find the spanning tree of lowest cost in which every node has degree at most d . The case $d = 2$ is equivalent to traveling salesman and hence NP-hard. Papadimitriou and Vazirani [15] showed NP-hardness for $d = 3$ in the plane, and conjectured that the problem remains NP-hard for $d = 4$. The problem can be solved in polynomial time for $d = 5$ since a minimum spanning tree has degree at most 5. We are interested in approximation algorithms for the difficult cases, both in $\mathbb{R}^2$ and $\mathbb{R}^k$. (In $\mathbb{R}^k$ the degree bound on the optimum spanning tree is of the form $\exp(\Theta(k))$, so all degrees less than that are interesting cases for the problem.) This problem is the most basic of a family of well-studied problems about finding degree-constrained structures; see Raghavachari's survey [16].

An *approximation scheme* for an NP-minimization problem is an algorithm that can, for every $\epsilon > 0$, compute a solution whose cost is at most $(1 + \epsilon)$ times the optimum. If the running time is polynomial for every fixed ϵ , then we say that the approximation scheme is a *Polynomial Time Approximation Scheme* (PTAS), and if the running time is not quite polynomial but $n^{\text{poly}(\log n)}$ then we say the approximation scheme is a *Quasipolynomial Time Approximation Scheme* (QPTAS).

* Supported by David and Lucille Packard Fellowship, and NSF Grants CCR-0098180 and CCR-009818. Work done partially while visiting the CS Dept at UC Berkeley.

We now know that many geometric problems have PTASs. Arora [1], and independently, Mitchell [14] showed the existence of PTASs for many geometric problems in $\mathbb{R}^2$, including Traveling Salesman, Steiner Tree, k -TSP, k -MST. (Arora’s algorithms also extend to instances in $\mathbb{R}^k$ for any fixed k .) Later, the running time of many of these algorithms was improved to nearly linear (Arora [2] and Rao-Smith [17]). Similar PTASs were later designed for many other geometric problems. For a current survey of approximation schemes for geometric problems, see Arora [3].

The above-mentioned survey notes that all these results use similar methods. Underlying the design of the PTAS is a “Structure Theorem” about the problem, demonstrating the existence of a near-optimal solution with very local structure—namely, it is possible to give a recursive geometric dissection of the plane such that the solution crosses each square in the dissection very few times. (A simple dynamic programming can optimize over such solutions.) The Structure Theorem is proved by showing that an optimum solution can be modified—by breaking it up locally and “patching it up” so as to not greatly increase cost—to have such local structure.

The survey goes on to notice that this method of proving the Structure Theorem breaks down when the optimum solution has to satisfy some topological conditions. Two examples were given: the degree restricted spanning tree, and minimum weight Steiner triangulation. Attempts to break the optimum solution and patch up the effects seems to have a rippling effect as we try to reimpose the topological constraint (the degree constraint, or the fact that the solution is a triangulation); the ensuing nonlocal changes are difficult to reason about¹.

In this paper we present a QPTAS for degree restricted spanning tree problem in $\mathbb{R}^2$. The running time in $\mathbb{R}^2$ is $n^{O(\log^5 n)}$ (we know how to reduce it to $n^{O(\log^3 n)}$ with a more complicated argument). For any $d > 2$, our algorithm, generalized to $\mathbb{R}^k$, runs in time $n^{O(\log^{2k+1} n)}$. The previous best algorithms were due to Khuller, Raghavachari and Young [10], who gave a 1.5-approximation for $d = 3$ and a 1.25-approximation for $d = 4$ in $\mathbb{R}^2$. For $d = 3$, they gave a $5/3$ approximation in $\mathbb{R}^k$, but have no such result for $d > 3$. We also present a PTAS for another problem with a topological flavor, the *Red Blue Separation* problem. In this problem we are given a set of points, some Red and some Blue. We desire the simple polygon with lowest edge length that contains all the Red points and no Blue points. This PTAS is easier to design.

2 Degree Restricted Spanning Tree

We start by defining the recursive scheme for dissecting the instance, namely, a $1/3 : 2/3$ *tiling*. It differs from the dissection used in other geometric PTASs in that it is not picked randomly at the start. Instead, the algorithm searches for a

¹ We note that the salesman problem also involves a topological constraint on the solution, namely, all degrees are 2. However, then the solution is 2-connected and a simple Patching Lemma—see Lemma 11—holds. This fact does not generalize to degree restricted spanning tree, where we allow degrees of 1.

suitable tiling using dynamic programming. This search goes hand-in-hand with the dynamic programming used to find the nearoptimum solution.

Let the *bounding box* be the smallest square around our perturbed instance, and let L denote the length of each side. We assume by rescaling distances that L is an integer and is n^3 . Clearly, the optimum tree has cost $\text{OPT} \geq L$.

In any rectangle of length l and width w where $l \geq w$, a *line separator* is a straight line segment parallel to the shorter edge of the rectangle that lies in the middle 1/3rd (i.e., its distance from each of the shorter edges is at least $l/3$). Below, we use only line separators with integer coordinates.

The $1/3 : 2/3$ tiling of our instance is a binary tree of rectangles, whose root is the bounding box. Each tree node is a rectangle, and its two children are obtained by dividing the rectangle using some line separator. We stop the partitioning when the rectangle's larger side has length at most L/n^2 . (Recall, $L = n^3$ is the size of the root square.) Clearly, the depth of the tree is $O(\log n)$. Note that the number of distinct $1/3 : 2/3$ tilings could be exponential because the partitioning is free to pick any line separator at each step. We will also associate with the tiling a set of *portals*. We designate an integer $m > 0$ as the *portal parameter*. Along each line separator used in the tiling, we place m evenly spaced points with integer coordinate known as *portals*. A set of edges (for example, a spanning tree or a salesman tour) is called *portal respecting* if the edges cross each line separator only at its portals. The set of edges is (m, r) -light with respect to a $1/3:2/3$ -tiling, if it is portal respecting, and each line separator is crossed by at most r edges.

Throughout, d denotes the degree bound; in the plane the interesting cases are $d = 3, 4$.

Theorem 1 (Structure Theorem for DRMST) *There exists a $1/3 : 2/3$ -tiling and a degree-restricted spanning tree with cost at most $(1 + \epsilon)\text{OPT}$, such that the tree is (m, r) -light with respect to this $1/3:2/3$ -tiling. Here $m = O(\log n/\epsilon)$ and $r = O(\log^5 n)$. Furthermore, every line separator used in defining the $1/3 : 2/3$ -tiling has integer coordinates.*

This theorem immediately leads to a dynamic programming algorithm that is a QPTAS for the problem. We sketch this dynamic programming here and defer details to the complete paper. (We also omit the Structure Theorem for $\mathbb{R}^k$ and its QPTAS.)

First, observe that there are only $O(n^3)$ choices for the horizontal and vertical lines used as line separators in the $1/3:2/3$ tiling, since they have integer coordinates in $[0, L]$ and $L = O(n^3)$. Since every rectangle used in the tiling is bounded by either one of the sides of the bounding box or one of the above $O(n^3)$ lines, there are only $O(n^{12})$ possible rectangles that could occur as nodes in the tiling. The basic subproblems solved by the dynamic programming involve the following inputs: (a) A rectangle. (b) A set of k edges ($k \leq 4r$) out of all possible $\binom{n}{2}$ edges $(u_1, v_1), (u_2, v_2), \dots, (u_k, v_k)$ which cross the boundary of the rectangle. In other words, u_i 's lie inside the rectangle and v_i 's are outside. Any u_i, v_i could occur in multiple edges subject of course to the degree constraint. (c)

A tree on $\{u_1, \dots, u_k\} \cup \{v_1, v_2, \dots, v_k\}$ that includes the edges in b . This tree forms a “template” for how the final solution must connect up the edges. The solution to this instance is the portion of the desired d -restricted spanning tree that lies inside the rectangle. The total number of subproblems (and hence the size of the dynamic programming table) is at most $O(n^{12}) \times (\sum_{k=1}^{4r} \binom{n^2}{r}) \times (8r)^{8r}$, which is at most $n^{O(\log^5 n)}$.

To solve a subproblem we proceed in the usual bottom-up manner. The base case is rectangles whose each side is at most L/n^2 . We solve these arbitrarily (subject to the constraints imposed by the template of course). For any other rectangle, we try all possibilities for the line separator; for each such partition into two smaller rectangles we try all possible templates involving edges that cross the line separator, look up the solutions computed already for those subproblems, and use the line separator and the template that minimizes the cost.

The correctness of this follows from the Structure Theorem and the fact that solving the instance arbitrarily in the base case can only affect the cost by $(n-1) \times L/n^2 < L/n < \epsilon \text{OPT}$ since the tree has only $n-1$ edges.

2.1 Proof of Structure Theorem

The theorem is of course similar to known results for other geometric problems. The proof is also somewhat similar: we start with an optimum solution and whenever it crosses the tiling too often, we modify it so that it doesn't. To reason about cost increases, we will use some fixed optimum solution, say T_0 , rooted at some arbitrary node. We let d denote the degree bound. We will use the following objects extensively.

Definition 2 (d -forest) *A d -forest in the instance is a collection $(T_1, u_1), \dots, (T_k, u_k)$ where the T_i 's are node-disjoint trees of degree at most d which together contain all nodes, and $u_i \in T_i$ is the representative node for T_i . We require that u_1 has degree at most $d-1$ and $u_2, u_3, \dots, u_k$ have degree at most $d-2$. We call u_1 the start node.*

Definition 3 (Rearrangeable Path) *If T is any degree-restricted spanning tree, a free node is a node with degree at most $d-1$. For any tree edge e , the rearrangeable path, p , associated with e is the path from e to a descendent free node such that p has the least number of edges of all such paths (in case of ties, pick the rightmost path).*

Remark 1. A rearrangeable path has at most $\lceil \log_2 n \rceil$ edges, since otherwise the subtree rooted at this edge would have degree 3 for depth at least $\log_2 n$, and thus more than n nodes. (Our convention is to include e in the path.) If two tree edges have rearrangeable paths that contain a common edge, then one must be a descendent of the other and hence one path is contained in the other.

Suppose T is a degree d tree rooted at v_0 and $e = (v_1, v_2)$ is some edge whose rearrangeable path is $(v_1, v_2, \dots, v_l)$, where v_l is a free node. Deleting every edge on this path partitions the nodes of T into subtrees $T_1, T_2, \dots, T_l$, where v_j lies in T_j . Furthermore, each of $v_2, v_3, \dots, v_l$ now has degree at most $d - 2$ (since we deleted two edges incident to each $v_1, \dots, v_{l-1}$, and one edge incident to v_l) and v_1 has degree at most $d - 1$. Thus $(T_1, v_1), (T_2, v_2), \dots, (T_l, v_l)$ is a d -forest, where each v_i for $i \geq 2$ is both the root and the representative node for T_i .

Now we are ready to prove a crucial property of the optimum tree. The following lemma relies on the fact that the shortest salesman path on k nodes in a square of sidelength M has length $O(\sqrt{k}M)$. The well-known *salesman strip tour* is a construction that achieves this bound. We will use it often.

Lemma 4 (Crossing Lemma for DRMST) *For any constant c there is a constant $c' > 0$ such that the following is true for any optimum degree-restricted minimum spanning tree T_0 . Let S be any straight line segment of length $s \geq 1$. Then T_0 has at most $c' \log^5 n$ edges that cross S and have rearrangeable paths longer than $s/(c \log n)$.*

Proof: Assume for contradiction's sake that the number of such edges exceeds $c' \log^5 n$. We modify the tree to produce one of lower cost, contradicting optimality.

Recall that the bounding box has length $L = n^3$, so rearrangeable paths have length at most $O(n^3 \log n)$. Partition the edges with rearrangeable paths of length greater than $s/c \log n$ into at most $N \leq 4 \log n$ different categories, $C_0, C_1, C_2, \dots, C_N$ where C_j contains edges whose rearrangeable paths have length from $s2^j/c \log n$ to $s2^{j+1}/c \log n$. The pigeonhole principle implies that some category, say C_i , contains $k' > \frac{c'}{4} \log^4 n$ crossing edges. Since a rearrangeable path contains at most $\log n$ edges and two paths intersect iff one lies inside the other, we can pick $k = k'/\log n$ paths from this category that are pairwise edge-disjoint. Note that the paths have lengths between $M/2$ and M , where $M = s2^{j+1}/c \log n$.

Remove all edges in all k paths; this gives a d -forest whose cost is lower by at least $kM/2$. Since each path removal gives rise to at most $\log n$ subtrees, the d -forest has at most $k \log n + 1$ subtrees.

Connect the representative nodes of the $\leq k \log n + 1$ trees with the shortest salesman path, whose length is at most $O(\sqrt{k \log n + 1}(2M + s))$. Since $k \geq \frac{c'}{4} \log^3 n$ and $M > 2s/c \log n$ the total added cost is $O(\sqrt{c'} \log^2 n (M + s))$, which for large enough c' is lower than the cost $kM/2$ saved earlier when we removed edges. Thus we have lowered total cost, while keeping all degrees at most d . $\square$

The proof below uses reasoning similar to Lemma 4. We start with the optimum degree- d spanning tree T_0 and repeatedly remove a rearrangeable path from a degree- d spanning tree to obtain a d -forest, and then add some salesman path starting at u_1 and visiting all of $u_2, u_3, \dots, u_k$ (not necessarily in this order), which gives a degree- d spanning tree again. We refer to this process of reconnecting with a salesman path as a *patching action*.

The power of this approach comes from the fact that *any* salesman path suffices, and that Arora's Structure Theorem for TSP shows the existence of nearoptimal salesman paths that cross the tiling boundaries "not too often." Thus the optimum tree can be gradually modified using patching actions to be (m, r) -light.

However, this iterative approach has a complication; even after one patching action the modified tree is no longer optimum and hence Lemma 4 does not apply to it. What saves the situation is that the patching actions leave most of the original tree untouched. This motivates the next few definitions and lemmas.

Definition 5 (Core) *Let T be some tree of degree at most d and T_0 be the optimum tree. A core of T is a set of edges $C \subseteq T_0 \cap T$ such that for every edge e in C , the rearrangeable path for e (with respect to T) also lies in C and this rearrangeable path is at most as long as the rearrangeable path for e in T_0 .*

Remark 2. Put another way, if T is derived from successive modifications of an optimum tree T_0 , a core is a set of edges in T_0 (with certain restrictions on the rearrangeable paths) that were not removed in any of the modifications. Cores are not unique, but there is a largest core that contains all other cores. We denote the largest core of a tree T as $C^*(T)$.

The next corollary explains our interest in cores; it immediately follows from Lemma 4.

Corollary 6 (to Lemma 4) *For any constant c there is a constant $c' > 0$ such that the following is true for a core C of any degree-restricted spanning tree. Let S be any straight line segment of length s . Then C has at most $c' \log^5 n$ edges that cross S and have rearrangeable paths longer than $s/(c \log n)$.*

Now consider what happens to $C^*(T)$ after a patching action on T that results in a tree T' . What is the largest core $C^*(T')$ of the modified tree? Lemma 9, whose relatively straightforward proof has been omitted, answers this question. We need two definitions.

Definition 7 (Upstream Edge) *If T is a degree- d tree and u is a node, then an edge e in T whose rearrangeable path passes through u is called an upstream edge associated with u, T .*

Remark 3. The set of all upstream edges associated with node u forms a path. Consider two nodes whose rearrangeable paths p_1 and p_2 pass through u ; then either p_1 is contained in p_2 or vice versa. Thus, there exists a largest path that contains all such upstream edges. Such a path contains at most $\log_2 n$ edges, since all rearrangeable paths have at most $\log_2 n$ edges. One can think of this path as the path of all nodes that are "upstream" from the node u . These observations lead to the following definition:

Definition 8 (Upstream Path) *If T is a degree- d tree and u is a node, then the path of edges that are upstream with respect to u , T is called the upstream path associated with u , T .*

The next Lemma shows that a patching action does not greatly affect the core (as noted, the upstream path has length at most $\log_2 n$).

Lemma 9 *Suppose tree T' results from a patching action on tree T that involved the removal of rearrangeable paths $p_1, \dots, p_k$ and the addition of a salesman path p that began at node u . If f is the upstream path associated with u , T , then*

$$C^*(T') \supseteq C^*(T) \setminus (f \cup \cup_i p_i).$$

Call the modification of $C^*(T)$ to $C^*(T')$ after a patching action a *core update*. Now we are ready to prove the main theorem.

Proof: (Of Structure Theorem) The proof consists of describing a procedure that, over $N = O(\log n)$ phases, produces a $1/3 : 2/3$ tiling of the bounding box, while simultaneously converting an optimum degree-restricted spanning tree, T_0 , into a degree-restricted spanning tree T_N that is (m, r) -light with respect to this tiling. The i 'th phase refines the depth i tiling into a depth $i + 1$ tiling and transforms T_i into T_{i+1} , where T_{i+1} is (m, r_{i+1}) -light with respect to the refined tiling and satisfies degree constraints, where $r, r_{i+1} = O(\log^5 n)$. At each phase we update the core of the tree as described earlier.

We maintain the following four invariants:

- (a) The tree T_{i+1} consists of three types of edges: the core $C^*(T_{i+1})$, a set of salesman paths $p_1, \dots, p_j$ (these are edge-disjoint but need not be vertex-disjoint) that became part of the tree during patching actions in previous iterations, and a set of *fixed edges* $\mathcal{F}_i \subseteq T_0$ that will not be removed from the tree in subsequent phases. We note that $\mathcal{F}_i$ is a union of some paths $f_1, \dots, f_j$ that were upstream paths consisting of edges from T_0 whose rearrangeable paths were found to intersect the start node of the salesman paths created in an earlier phase. (Definition 8). Thus we have

$$T_{i+1} = C^*(T_{i+1}) \cup (\cup p_j) \cup (\cup f_j)$$

As noticed in the remarks preceding Definition 8, each upstream path f_j has only $\log_2 n$ edges; these edges may also be contained in the core, but such cautious overaccounting will not hurt our result.

- (b) Each level $i + 1$ rectangle contains edges from at most $i + 1$ of the aforementioned salesman paths.
- (c) T_{i+1} crosses each level $i + 1$ line separator at most $r_{i+1} = O(\log^5 n)$ times.
- (d) T_{i+1} is portal-respecting, with m as the portal parameter, declared below.

Note that invariants (c) and (d) imply that T_{i+1} is (m, r_{i+1}) -light.

We will use the following three quantities extensively: the portal parameter m , which satisfies $m > 72 \log n$ and $m = O(\log n / \epsilon)$, the constant c (it may

depend on ϵ), which satisfies $c > 2m/\log n$, and t , the bound on the number of edges with rearrangeable paths of length at least $s/c \log n$ that cross a line segment of length s from Corollary 6, which satisfies $t = O(\log^5 n)$. Recall that t depends only on c and not on s .

We will prove that the costs of the trees satisfy the recurrence

$$\text{cost}(T_{i+1}) \leq \text{cost}(T_i) + \frac{11\text{cost}(T_i)}{m}.$$

After all $N = O(\log n)$ phases, this implies that $\text{cost}(T_N) \leq (1 + \frac{11}{m})^N \text{OPT} \leq (1 + \epsilon)\text{OPT}$, since $m = O(\log n/\epsilon)$.

DETAILS OF PROCEDURE: Phase i considers level i rectangles of the current tiling one by one, divides the rectangle into two by picking a suitable line separator, and modifies the tree in such a way that it still maintains our four invariants.

Let R be a level i rectangle whose longer side has length W . Let $T_i|_R$ denote the portion of T_i that lies in R . For each action of Phase i , we first describe the modifications, then show how the actions contribute to maintaining the invariants, and lastly bound the cost of T_{i+1} .

PHASE i 'S ACTION IN R , FIRST STEP: Theorem 5 from [1] implies that a random line separator of R is crossed by T_i at most $C = 3\text{cost}(T_i|_R)/W$ times. If $C < t$, we do not modify $T_i|_R$ at all, and just pick such a separator to get two level $i+1$ rectangles.

On the other hand, suppose $C \geq t$. Regardless of which separator we pick, we may need to modify the tree in order to reduce the number of edges that cross the line separator. Let CE be the set of edges that cross our chosen line separator. We partition the edges of CE into six categories:

$$CE = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6,$$

where the categories are defined below:

1. E_1 consists of crossing edges belonging to the i salesman paths in R .
2. E_2 consists of crossing edges belonging to the core, whose rearrangeable paths are shorter than $W/c \log n$ and are completely contained in R .
3. E_3 consists of crossing edges belonging to the core, whose rearrangeable paths are longer than $W/c \log n$.
4. E_4 consists of crossing fixed edges e_f whose rearrangeable paths in T_0 are of length greater than $W/c \log n$.
5. E_5 consists of crossing fixed edges whose rearrangeable paths in T_0 are of length less than $W/c \log n$ and are completely contained in R .
6. E_6 consists of crossing fixed or core edges whose rearrangeable paths in T_0 and T_i , respectively, are of length less than $W/c \log n$, but are not contained in R . i.e. their rearrangeable paths each contain at least one edge with an endpoint outside of R .

Step 1 proceeds by first choosing a line separator S for rectangle R and then modifying the tree so that the total number of edges crossing S is small and invariants (a), (b), and (c) are maintained.

The following six claims show how to bound the number of edges in each successive category, or how to locally modify the tree —namely, leaving it unchanged outside rectangle R — in order to reduce the crossings in a way that is consistent with invariants (a), (b), and ultimately (c). It is important to note that the following bounds/modifications on E_1, E_2, E_3, E_4 and E_5 do not depend on which line separator is chosen.

CLAIM 1 (MODIFICATION): *Given any line separator, the tree can be modified so that the number of crossing edges in E_1 is reduced to $2i$. The modification preserves invariants (a) and (b), and results in a cost increase of at most $O(iw)$.*

Proof: We can break these edges and then apply Lemma 11 at most i times to reconnect the freed nodes, thus raising cost by at most $O(iW)$, and reducing the number of crossings from these salesman paths to $2i$. $\square$

CLAIM 2 (MODIFICATION): *Given any line separator, the tree can be modified so that all edges in E_2 are removed and replaced with a salesman path that crosses the line separator at most twice. This preserves invariants (a) and (b).*

Proof: We remove all the edges of E_2 along with their rearrangeable paths, thus obtaining a d -forest F . We then execute a patching action on F , but use a salesman path p that crosses S at most twice (Lemma 11 shows such a path exists), and update the core. The upstream path associated with the start node of p now becomes a set of fixed edges. Clearly, invariants (a) and (b) are maintained since the added salesman path lies entirely in R . $\square$

CLAIM 3 (BOUND): *For any line separator, E_3 contains at most t crossing edges.*

Proof: Apply Corollary 6. $\square$

CLAIM 4 (BOUND): *For any line separator, E_4 contains at most t edges.*

Proof: Since each edge in E_4 corresponds to a unique crossing edge in T_0 with long rearrangeable path, there are at most t of these by Lemma 4. $\square$

CLAIM 5 (BOUND): *For any line separator, E_5 contains at most $i \log_2 n$ edges.*

Proof: A fixed edge $e \in E_5$ must be associated with some salesman path that lies partly in R . Since at most $\log_2 n$ upstream edges (and hence $\log_2$ fixed edges) are associated with each salesman path and by invariant (b) there are at most i salesman paths in R , the total number of edges in E_5 cannot exceed $i \log_2 n$. $\square$

We do not modify the tree in order to remove edges in E_6 , but rather prove the existence of a line separator S such that the number of crossing edges of type E_6 is small. Fortunately, all the modifications and bounds we proved for the sets E_1, E_2, E_3, E_4 and E_5 are independent of the line separator and thus we are free to choose any S we wish, without fear of destroying these other bounds.

CLAIM 6 (BOUND): *There exists a line separator S of R such that the number of edges in category E_6 is at most $r_i/2$.*

Proof: The following Lemma is similar to Theorem 5 from [1]

Lemma 10 *For a random line separator, the expected number of crossing edges in E_6 is at most $r_i/6$.*

Proof: Recall that invariant (d) guarantees that the tree T_i is portal respecting. We had already previously set the portal parameter m to satisfy $m > 72 \log n$. By invariant (c), there are at most r_i edges of T_i that cross into R . For appropriately chosen c we have $c \log n > 2m$. Then for any given line separator S , the crossing edges in E_6 can originate from at most two portals, since the rearrangeable paths of the crossing edges are shorter than $s/c \log n$. By averaging, we know that the expected number of edges that enter these two portals of a random line separator is at most $r_i/12 \log n$. Since each of these rearrangeable path can induce at most $\log_2 n$ crossings in E_6 , the expected number of such crossings is at most $r_i/6$. $\square$

Let $\mu = 3C$. If we pick a random line separator, with probability at least $2/3$, the number of total crossings is at most μ (by Theorem 5 in [1] and with probability at least $2/3$, the number of crossings edges in E_6 is at most $r_i/2$. Therefore a line separator S exists for which both events occur, and we choose S to divide the rectangle. $\square$

We have thus shown how to modify the tree in order to bound each type of edge, and that the modifications preserve invariants (a) and (b). We now show that these modifications preserve invariant (c).

BOUNDING THE TOTAL NUMBER OF EDGES CROSSING S : We have modified the tree in order to bound each type of crossing edge: E_1, E_2, E_3, E_4, E_5 and E_6 . In order to see that invariant (c) is maintained, note that the total number of times the modified tree crosses S is at most $r_{i+1} = 2 + 2i + 3t + i \log_2 n + r_i/2$. Since $r_1 \leq 2t$, this recurrence relation solves to $r_{i+1} \leq 4t \sum_{0 \leq j \leq i+1} 1/2^j \leq 8t = O(\log^5 n) \forall i$.

BOUNDING THE COST INCREASE: As proved in the discussion of Claim 2, the cost of removing edges in E_1 is at most $O(iW) = WO(\log n)$. For n sufficiently large, this cost is $< W\mu/9m < \text{cost}(T_i|_R)/m$. As proved in the discussion of Claim 3, the cost of removing edges in E_2 is the cost of the salesman path that visited at most $\mu \cdot \log_2 n$ nodes. We know such a salesman path costs at most $O(W\sqrt{\mu \log_2 n + 1})$. For n sufficiently large, this cost is $< W\mu/m \leq \text{cost}(T_i|_R)/m$.

PHASE i 'S ACTION ON R , SECOND STEP: The second step of Phase i modifies the tree so that the result is portal respecting (invariant (d)). The step involves moving all $\leq 3C$ crossings to portals. This action is accomplished by adding two vertical detours of length at most $W/2(m-1)$. Thus at the end all crossings are portal-respecting. For n sufficiently large, the total cost of moving crossings to portals is then at most $3CW/(m-1) \leq 9\text{cost}(T_i|_R)/(m-1)$.

Call the tree resulting from Phase i Steps 1 and 2 on all level i rectangles T_{i+1} . We have shown that T_{i+1} satisfies all four invariants, and that $\text{cost}(T_{i+1})$ satisfies the recurrence relation $\text{cost}(T_{i+1}) \leq \text{cost}(T_i) + \frac{11\text{cost}(T_i)}{m}$. Furthermore, each T_i for $i = O(\log n)$ is (m, r) -light, where $r = r_N = O(t)$. $\square$

Now we state a Lemma that was used above.

Lemma 11 (Patching Lemma for TSP [1,2]) *Let S be any line segment of length s and P be any salesman path that crosses S at least three times. Then*

we can break the path in all but two of these places, and add to it line segments lying on S of total length at most $3s$ such that P changes into a salesman path P' that crosses S at most twice.

3 Red Blue Separation

In order to prove the existence of a PTAS for RBSP, we simply need to prove a patching lemma for RBSP. The rest of the proof follows the treatment in [2], with only some very straightforward modifications. This gives an algorithm running in $n(\log n)^{O(1/\epsilon)}$ time. Joe Mitchell has mentioned to us (private communication) that the techniques of [14] give a PTAS as well though not as efficient.

Lemma 12 (Patching Lemma for RBSP) *Let S be any line segment of length s and P be the boundary of a simple separating polygon that crosses S at least three times. Then we can break P at all but two of these places and add segments lying on S of total length at most $4s$, so that we have a separating polygon whose boundary P' crosses S in at most two places.*

Proof: We assume without loss of generality that S is a horizontal grid line. We describe a four step patching algorithm that converts P into P' that satisfies the conditions of the lemma:

STEP 1: Label the $t > 2$ crossings of S , from left to right, as the points $x_1, x_2, \dots, x_t$. We know that one side of each line of P lies on the “inside” and the other lies on the “outside” of the polygon. Since these inside and outside regions must alternate, we have one of two scenarios:

- A. Segments $\overline{x_1x_2}, \overline{x_3x_4}, \overline{x_5x_6}, \dots$ are all contained in the interior of the polygon (and $\overline{x_2x_3}, \overline{x_4x_5}, \dots$ are not).
- B. Segments $\overline{x_2x_3}, \overline{x_4x_5}, \overline{x_6x_7}, \dots$ are all contained in the interior of the polygon (and $\overline{x_1x_2}, \overline{x_3x_4}, \dots$ are not).

STEP 2: If we have Scenario A, break P at $x_1, x_2, x_3, \dots, x_{t-2}$. If t is odd, break P at x_{t-1} as well. If we have Scenario B, break P at $x_2, x_3, \dots, x_{t-1}$. If t is odd, break P at x_t as well.

STEP 3: For Scenario A, if t is odd, add line segments $\overline{x_1x_2}, \overline{x_3x_4}, \overline{x_5x_6}, \dots, \overline{x_{t-2}x_{t-1}}$ to both sides of S . If t is even, add line segments $\overline{x_1x_2}, \overline{x_3x_4}, \overline{x_5x_6}, \dots, \overline{x_{t-3}x_{t-2}}$. For Scenario B, add line segments $\overline{x_2x_3}, \overline{x_4x_5}, \overline{x_6x_7}, \dots$.

Step 3 has patched the open ends of the original simple polygon, so that we have a union of connected components with well defined insides and outsides. All points of one color are inside and all points of the other color are outside. The patching has not introduced any intersections of the new boundary of the polygons with itself, nor any intersections of the new boundary with S .

STEP 4: **do** the following, first for connected components that intersect the region above S , then for all components that remain:

Let the connected components of interest be $P_1, P_2, \dots, P_k$. Let y_i be the point x_j , such that j is the largest j such that P_i touches x_j . Let $y'_i = x_{j+1}$.

Note for exactly one i , y'_i does not exist. Assume wlog that this $i = k$. Then, add two copies of the edges $y_1\overline{y'_1}$, $y_2\overline{y'_2}$, $\dots$, $y_{k-1}\overline{y'_{k-1}}$. The proofs of the following two claims are straightforward and have been deferred to the complete paper.

CLAIM 1: After Step 3, each connected component created by the algorithm is in fact a simple polygon.

CLAIM 2: After Step 4, all the simple separating polygons created in Step 3 have been “merged” into a single simple separating polygon. $\square$

4 Conclusions

Can we design approximation schemes for other geometric problems that involve complicated topology? Minimum weight Steiner triangulation seems like the next obvious candidate. We only know of a 316-approximation due to Eppstein. For a survey of this and other problems with topological constraints, see Bern and Eppstein [7]. For all of these problems a first step would be to prove theorems about the structure of the optimum solutions (analogous to Lemma 4).

We do not currently see a way to reduce the running time of our approximation scheme from quasipolynomial to polynomial.

References

1. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Proceedings of 37th IEEE Symp. on Foundations of Computer Science*, 1996.
2. S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *JACM* **45**(5) 753–782, 1998.
3. S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Math Programming*, 2003 (to appear). Available from www.cs.princeton.edu/~arora.
4. S. Arora and G. Karakostas. Approximation schemes for minimum latency problems. *Proc. ACM Symposium on Theory of Computing*, 1999.
5. S. Arora, P. Raghavan, and S. Rao. Approximation schemes for the Euclidean k -medians and related problems. In *Proc. 30th ACM Symposium on Theory of Computing*, pp 106–113, 1998.
6. J. Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Proc. Cambridge Philos. Soc.* **55**:299–327, 1959.
7. M. Bern and D. Eppstein. Approximation algorithms for geometric problems. In [9].
8. A. Czumaj and A. Lingas. A polynomial time approximation scheme for Euclidean minimum cost k -connectivity. *Proc. 25th Annual International Colloquium on Automata, Languages and Programming*, LNCS, Springer Verlag 1998.
9. D. Hochbaum, ed. Approximation Algorithms for NP-hard problems. PWS Publishing, Boston, 1996.
10. S. Khuller, B. Raghavachari, and N. Young. Low degree spanning tree of small weight. *SIAM J. Computing*, **25**:355–368, 1996. Preliminary version in *Proc. 26th ACM Symposium on Theory of Computing*, 1994.

11. S. G. Kolliopoulos and S. Rao. A nearly linear time approximation scheme for the Euclidean k -median problem. *LNCS*, vol.1643, pp 378–387, 1999.
12. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys. The traveling salesman problem. John Wiley, 1985
13. C.S. Mata and J. Mitchell. Approximation Algorithms for Geometric tour and network problems. In *Proc. 11th ACM Symp. Comp. Geom.*, pp 360–369, 1995.
14. J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple PTAS for geometric k -MST, TSP, and related problems. *SIAM J. Comp.*, **28**, 1999. Preliminary manuscript, April 30, 1996. To appear in *SIAM J. Computing*.
15. C. H. Papadimitriou and U. V. Vazirani. On two geometric problems related to the traveling salesman problem. *J. Algorithms* **5**(1984), pp. 231–246.
16. B. Raghavachari. Algorithms for finding low degree structures. In [9]
17. S. Rao and W. Smith. Approximating geometric graphs via “spanners” and “banyans.” In *Proc. 30th ACM Symposium on Theory of Computing*, pp. 540–550, 1998.

Approximating Steiner k -Cuts

Chandra Chekuri¹, Sudipto Guha², and Joseph (Seffi) Naor³

¹ Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974

`chekuri@research.bell-labs.com`,

² Dept. of Computer & Information Science, University of Pennsylvania,
Philadelphia, PA 19104

`sudipto@cis.upenn.edu`,

³ Computer Science Dept., Technion, Haifa 32000, Israel

`naor@cs.technion.ac.il`

Abstract. We consider the Steiner k -cut problem, which is a common generalization of the k -cut problem and the multiway cut problem: given an edge-weighted undirected graph $G = (V, E)$, a subset of vertices $X \subseteq V$ called *terminals*, and an integer $k \leq |X|$, the objective is to find a minimum weight set of edges whose removal results in k disconnected components, each of which contains at least one terminal. We give two approximation algorithms for the problem: a $2 - \frac{2}{k}$ -approximation based on Gomory-Hu trees, and a $2 - \frac{2}{|X|}$ -approximation based on LP rounding. The latter algorithm is based on rounding a generalization of a linear programming relaxation suggested by Naor and Rabani [8]. The rounding uses the Goemans and Williamson primal-dual algorithm (and analysis) for the Steiner tree problem [4] in an interesting way and differs from the rounding in [8]. We use the insight from the rounding to develop an exact bi-directed formulation for the global minimum cut problem (the k -cut problem with $k = 2$).

Keywords: Multiway Cut, k -Cut, Steiner tree, minimum cut, primal-dual.

1 Introduction

Two fundamental graph partitioning problems are the k -cut problem and the multiway cut problem. In both problems we are given an undirected edge-weighted graph $G = (V, E)$ with $w(e)$ denoting the weight of edge $e \in E$. In the k -cut problem the goal is to find a minimum weight set of edges to separate the graph into at least k disconnected components. In the multiway cut problem we are given a set of k terminals, $X \subseteq V$, and the goal is to find a minimum weight set of edges to separate the graph into components, such that each terminal is in a different connected component. In this paper we define a common generalization of the two problems that we call the Steiner k -cut problem. We are given an undirected weighted graph G , a set of *terminals* $X \subseteq V$, and an integer $k \leq |X|$. The goal is to find a minimum weight cut that separates the graph into k components with vertex sets $V_1, V_2, \dots, V_k$, such that $V_i \cap X \neq \emptyset$

for $1 \leq i \leq k$. If $X = V$, we obtain the k -cut problem. If $|X| = k$ we obtain the multiway cut problem.

The k -cut problem can be solved in polynomial time for fixed k [5,6], but is NP-hard when k is part of the input [5]. In contrast, the multiway cut problem is NP-hard for all $k \geq 3$ [2]. It follows that the Steiner k -cut problem is NP-hard for all $k \geq 3$. For the multiway cut problem Calinescu, Karloff and Rabani [1] gave a $1.5 - 1/k$ approximation using an interesting geometric relaxation. Karger et al. [7] improved the analysis of the integrality gap of this relaxation and obtained an approximation ratio of $1.3438 - \epsilon_k$, where ϵ_k tends to 0 as k tends to ∞ . For the k -cut problem Saran and Vazirani [10] gave a $2 - \frac{2}{k}$ approximation algorithm using a greedy algorithm. Recently, two different 2-approximations for the k -cut problem were obtained. The algorithm of Naor and Rabani [8] is based on rounding a linear programming formulation of the problem and the algorithm of Ravi and Sinha [9] is based on the notion of network strength.

1.1 Results

We provide two approximation algorithms for the Steiner k -cut problem. The first algorithm we present is combinatorial and achieves a factor of $2 - \frac{2}{k}$. The algorithm is based on choosing cuts from the Gomory-Hu tree and it is very similar to approximation algorithms developed for the k -cut problem and the multiway cut problem [11]. Our main result is a 2-approximation algorithm for the Steiner k -cut problem which is based on rounding a linear programming formulation. Although our formulation is a generalization of the formulation in [8] (for the k -cut problem), our rounding scheme differs substantially. The rounding in [8] exploits the properties of the optimal solution to the LP relaxation. These properties do not hold for the relaxation of the Steiner k -cut problem. Our rounding is based on the primal dual algorithm and analysis of Goemans and Williamson [4] for the Steiner tree problem. As a consequence, our rounding algorithm extends to any feasible solution of the linear programming formulation. We believe that this interesting new connection would have future applications.

We conclude with a bi-directed formulation for the global minimum cut problem and prove that the linear relaxation of this formulation is exact. The formulation and analysis are inspired by machinery developed for the Steiner k -cut problem.

2 Combinatorial $(2 - \frac{2}{k})$ -Approximation Algorithm

We assume without loss of generality that the given graph G is connected. A natural greedy algorithm for the Steiner k -cut problem is the following iterative algorithm. At each step, find a minimum weight cut that increases the number of distinct components that contain a terminal. This algorithm has been shown to achieve a $2 - \frac{2}{k}$ approximation algorithm for both the k -cut problem and the multiway cut problem (e.g., [11]). However, the analysis of this algorithm for the k -cut problem is non-trivial. As in [10,11], we consider an alternative algorithm

which is based on the Gomory-Hu tree representation of the minimum cuts in a graph. Given G , let $T = (V_T, E_T)$ be a Gomory-Hu tree of the graph G . Let c denote the weight function defined on the edges of T . In the Gomory-Hu tree T , for all $(u, v) \in E_T$, $c(u, v)$ is the weight of the minimum cut separating u and v in G . We run the natural greedy algorithm mentioned above on the tree T : we iteratively pick the smallest weight edge in T that would separate a pair of terminals that are not already separated. It is easy to see that we will pick $k - 1$ edges in T . We take the union of the cuts associated with these edges and this is our solution for the Steiner k -cut problem in G .

Theorem 1. *There is a $2 - \frac{2}{k}$ -approximation algorithm for the Steiner k -cut problem that runs in time required to build a Gomory-Hu tree representation of the minimum cuts of the input graph G .*

Proof. The proof is along the same lines as the proof of Theorem 4.8 in [11, Page 42] for the k -cut problem. Fix an optimal solution A to the Steiner k -cut problem and let $V_1, \dots, V_k$ be the partitioning of V in A . Clearly, each set V_i ($i = 1, \dots, k$) contains at least one terminal from X . From each set V_i we choose a terminal t_i contained in V_i . Define cuts $A_i = (V_i, V \setminus V_i)$ for $i = 1, \dots, k$, and let $w(A_i)$ denote the weight of cut A_i . Suppose without loss of generality that $w(A_1) \leq w(A_2) \leq \dots \leq w(A_k)$. Since each edge of the optimum participates in exactly two cuts, the weight of the optimal solution A is $w(A) = \sum_{i=1}^k w(A_i)/2$. Let $B_1, \dots, B_{k-1}$ denote the $k - 1$ lightest cuts chosen by the algorithm from the Gomory-Hu tree T . We first prove that

$$\sum_{i=1}^{k-1} w(B_i) \leq \sum_{i=1}^{k-1} w(A_i). \quad (1)$$

Let $T' = (V_{T'}, E_{T'}) \subseteq T$ be a minimal tree that spans $t_1, \dots, t_k$. Observe that any edge in T' corresponds to a minimum cut that separates a pair of terminals from $t_1, \dots, t_k$.

Consider the graph H with vertex set $v_1, \dots, v_k$ corresponding to $V_1, \dots, V_k$. Define v_i and v_j to be adjacent if there exist vertices $a, b \in V_{T'}$, where $a \in V_i$ and $b \in V_j$, and $(a, b) \in E_{T'}$. Clearly, H is a connected graph since the tree T' spans $V_{T'}$. Let R be a spanning arborescence of H obtained by taking a spanning tree of H and directing all edges towards the root v_k . Consider an edge $(v_i, v_j) \in R$ directed from v_i to v_j . Since $c(a, b)$ is the weight of the minimum cut separating $a \in V_i$ from $b \in V_j$ in the graph G , and since A_i is one such cut, it follows that $c(a, b) \leq w(A_i)$. Thus, the weights of the cuts A_i , $1 \leq i \leq k - 1$, can be charged to the $k - 1$ edges of R . Since the cuts $B_1, \dots, B_{k-1}$ correspond to the $k - 1$ lightest edges in T , we get that (1) holds. Since $w(A_1) \leq w(A_2) \leq \dots \leq w(A_k)$, we get that

$$\sum_{i=1}^{k-1} w(A_i) \leq \left(1 - \frac{1}{k}\right) \sum_{i=1}^k w(A_i) \leq 2 \left(1 - \frac{1}{k}\right) w(A)$$

completing the proof. $\square$

3 Linear Programming Formulation and a 2-Approximation

We assume without loss of generality that the input graph G is complete: if edge (u, v) is not in the original graph, we can add it with zero weight. Consider the following integer programming formulation for the problem. For each edge e we have a binary variable $d(e)$ which is 1 if the edge e belongs to the cut and 0 otherwise.

Consider any Steiner tree T on the terminal set X in G . In any feasible Steiner k -cut, at least $k - 1$ edges of T have to be cut. Based on this we obtain the following integer program for the Steiner k -cut problem.

$$\begin{aligned}
 (K) \quad & \min \sum_e w(e) \cdot d(e) \quad \text{subject to :} \\
 & \sum_{e \in T} d(e) \geq k - 1 \quad \forall T : T \text{ Steiner tree on } X \\
 & d(e) \in \{0, 1\} \quad \forall e
 \end{aligned}$$

A relaxation of this integer program is obtained by allowing the variables $d(e)$ to assume values in $[0, 1]$. The variables $d(e)$ are to be interpreted as inducing a semi-metric on V . Our formulation above is a straight forward extension of the formulation of Naor and Rabani [8] for the k -cut problem. In the k -cut problem $X = V$, and hence [8] considers only spanning trees of G .

Unfortunately, we do not know how to solve the above linear program in polynomial time. Consider, for example, the separation oracle required for running the Ellipsoid algorithm. Given a vector $\vec{d}$, the separation oracle has to check whether $\vec{d}$ induces a feasible solution, which means checking that the minimum cost Steiner tree on X in G , with edge weights defined by $\vec{d}$, is of cost at least $k - 1$. Since the Steiner tree problem is NP-hard, this problem is intractable. Note that for the k -cut problem, a polynomial time separation oracle is available because the *minimum spanning tree* (MST) can be computed efficiently.

We can use an approximate separation oracle based on the MST heuristic for the Steiner tree problem. Given a vector $\vec{d}$, let G_X be the complete graph on the terminal set X , where the weight of edge $(u, v) \in G_X$ is the weight of the shortest path from terminal u to terminal v (in G) with respect to $\vec{d}$. The oracle computes the MST on G_X . If the MST is of weight at least $k - 1$, the oracle concludes that $\vec{d}$ is feasible. If the weight of the MST is less than $k - 1$, it is easy to find a corresponding Steiner tree on X whose weight is less than $k - 1$. Note that we are assuming here that d induces a semi-metric on the $V(G)$. In other words, we are solving the following relaxation:

$$(K') \quad \min \sum_e w(e) \cdot d(e) \quad \text{subject to :}$$

$$\sum_{e \in T} d(e) \geq k - 1 \quad \forall T : T \text{ spanning tree in } G_X \quad (2)$$

$$\bar{d} \text{ induces a semi-metric on } V(G) \quad (3)$$

$$d(e) \in [0, 1] \quad \forall e \quad (4)$$

The next lemma follows from the discussion.

Lemma 1. *The linear program K' is a valid relaxation for the Steiner k -cut problem and it can be solved optimally in polynomial time.*

For the multiway cut problem we note that the linear program K' is equivalent to a linear program that constrains the terminals to be at distance at least 1 from each other. This latter linear program has been shown to have an integrality gap of $2(1 - 1/k)$ [2]. We will obtain the same result as well for the Steiner k -cut problem. We now prove a property of feasible solutions to K' that will be useful later.

Lemma 2. *Let $\bar{d}$ be any feasible solution to K' . Then, there is $X' \subseteq X$ such that $|X'| \geq k$, and for any two distinct vertices u and v in X' , $d(u, v) > 0$.*

Proof. For any two, not necessarily distinct, vertices u and v in X , define a relation R as follows: uRv iff $d(u, v) = 0$. Since d is a metric, this clearly defines an equivalence relation on X . We need to prove that the number of equivalence classes in R is at least k . Suppose this is not the case. For any two vertices a and b in V , $d(a, b) \leq 1$. Hence, there is a spanning tree on X of cost at most $\ell - 1$, where ℓ is the number of distinct equivalence classes. If $\ell < k$, then we get a contradiction to the the feasibility of $\bar{d}$. $\square$

3.1 Rounding the Linear Program

We show how to round a solution to (K') to yield a 2-approximation to the Steiner k -cut problem. In [8], for the k -cut problem, it is shown that there exists an optimal solution to (K) that defines an ultra metric on the vertices. The ultra metric property is used crucially to identify a family of cuts that pack into the metric $\bar{d}$. Then, the algorithm in [8] picks $k - 1$ cuts from this family using a probability distribution. In contrast, we do not use properties of the optimal solution. We use the Goemans and Williamson primal-dual approximation algorithm for the Steiner tree problem [4] (henceforth referred to as the GW algorithm) as a way of finding a set of cuts.

Let $\bar{d}$ be any feasible solution to linear program (K') . Then, $\bar{d}$ defines a weight function on the edges of G . Let G_d denote the resulting edge weighted graph. We run the GW primal-dual algorithm on the graph G_d to create a Steiner tree on X . To find a minimum Steiner on X in G_d , the GW algorithm uses the following cut based LP relaxation of the Steiner tree problem. Let $x(e)$ be 1 if e is in the Steiner tree and 0 otherwise. Then, every cut that separates the terminal set has to be covered by at least one edge. This yields the following linear program

where the variables are relaxed to be in $[0, 1]$. Note that in the programs below, the variables $d(e)$ are treated as constants obtained from a solution to K' .

$$(STP) \quad \min \sum_e d(e) \cdot x(e) \quad \text{subject to :}$$

$$\begin{aligned} \sum_{e \in \delta(S)} x(e) &\geq 1 & \forall S : S \text{ separates } X \\ x(e) &\in [0, 1] & \forall e \end{aligned}$$

The dual of this linear program is the following.

$$(STD) \quad \max \sum_S y(S) \quad \text{subject to :}$$

$$\begin{aligned} \sum_{S: e \in \delta(S)} y(S) &\leq d(e) & \forall e \\ y(S) &\geq 0 & \forall S : S \text{ separates } X \end{aligned}$$

The GW algorithm is a primal-dual algorithm that incrementally grows a dual solution while maintaining feasibility and computes a corresponding feasible primal Steiner tree such that the cost of the Steiner tree computed is at most twice the value of the dual solution found. Let y' be the dual solution produced by the GW algorithm upon termination and let T be the Steiner tree returned by the algorithm. Then the following properties are true for y' and T [4].

1. y' is a feasible solution to (STD).
2. T is a tree that spans the terminal set X .
3. $\sum_{e \in T} d(e) \leq 2(1 - 1/|X|) \sum_S y'(S)$.
4. The set of cuts S with $y'(S) > 0$ forms a laminar family.
5. Let $u \in X$ be a terminal such that for all $v \in X, v \neq u, d(v, u) > 0$. Then, there exists a cut S such that $y'(S) > 0$ and $S \cap X = \{u\}$.

With the above discussion in place, we are ready to describe our rounding procedure. For a cut S , let $w(S) = \sum_{e \in \delta(S)} w(e)$ denote the weight of S in G . We solve (K') to obtain a solution $\bar{d}$. We then run the GW algorithm on G_d with X as the set of terminals. Let y' be the dual solution obtained by the GW algorithm and let $\mathcal{S} = \{S \mid y'(S) > 0\}$ be the set of all cuts that have non-zero dual values in y' . We first argue that there are at least k cuts in $\mathcal{S}$.

Lemma 3. *Let $\bar{d}$ be a feasible solution to K' . Let y' be a feasible dual solution constructed by the GW algorithm when run on G_d . Then, there are at least k distinct cuts $S_1, S_2, \dots, S_k$, such that $y'(S_i) > 0$ for $1 \leq i \leq k$ and $S_i \cap X \neq S_j \cap X$ for $i \neq j$.*

Proof. Follows from Lemma 2 and Property (5) of y' . $\square$

Now we describe how we choose the cuts from $\mathcal{S}$. We partition $\mathcal{S}$ into classes $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_j$ such that two cuts S and S' are in the same class $\mathcal{S}_i$ if and only if $S \cap X = S' \cap X$. From Lemma 3 we have that $j \geq k$. For a class $\mathcal{S}_i$, let C_i be the least weight cut in $\mathcal{S}_i$. Let $\mathcal{C}$ be the collection of C_i , $1 \leq i \leq j$. Our algorithm simply outputs the union of the $k - 1$ cheapest cuts from $\mathcal{C}$.

We first argue about the correctness of the algorithm. Since the family of cuts $\mathcal{S}$ is laminar, so is the family of cuts in $\mathcal{C}$. By construction, for any two cuts $C_i, C_j \in \mathcal{C}$, $C_i \cap X \neq C_j \cap X$. Hence, picking $k - 1$ cuts from $\mathcal{C}$ results in at least $k - 1$ components, each of which contains a terminal from X .

We now analyze the performance of the algorithm. Our first observation is the following.

Claim. $\sum_{S \in \mathcal{S}} y'(S)w(S) \leq \sum_e w(e)d(e)$.

Proof. We have the following:

$$\begin{aligned} \sum_{S \in \mathcal{S}} y'(S)w(S) &= \sum_{S \in \mathcal{S}} y'(S) \sum_{e \in \delta(S)} w(e) \\ &= \sum_e w(e) \sum_{S: e \in \delta(S)} y'(S) \\ &\leq \sum_e w(e)d(e). \end{aligned}$$

The final inequality follows from the fact that y' is a feasible solution to (STD). $\square$

Claim. $2(1 - 1/|X|) \sum_{S \in \mathcal{S}} y'(S) \geq (k - 1)$.

Proof. The GW algorithm guarantees that $2(1 - 1/|X|) \sum_S y'(S) \geq \sum_{e \in T} d(e)$. Since T is an MST on X , from the feasibility of $\bar{d}$ for (K') , $\sum_{e \in T} d(e) \geq k - 1$. The claim follows. $\square$

Let $y'(\mathcal{S}_i)$ denote $\sum_{S \in \mathcal{S}_i} y'(S)$. Then, $\sum_i y'(\mathcal{S}_i) = \sum_{S \in \mathcal{S}} y'(S)$.

Claim. If for all e , $d(e) \leq 1$ then for all i , $1 \leq i \leq j$, $y'(\mathcal{S}_i) \leq 1/2$.

Proof. Let $X_i = S \cap X$ for some $S \in \mathcal{S}_i$. Then, it follows that for any $S' \in \mathcal{S}_i$, $S' \cap X = X_i$. Let $v \in X - X_i$ and $u \in X_i$. Then, the edge $(u, v) \in \delta(S)$ for all $S \in \mathcal{S}_i$. In the GW algorithm the cuts containing each terminal are grown simultaneously and at the same rate. Hence the set of cuts containing u can grow to at most $d(u, v)/2$ before meeting the cuts around v . Since $d(u, v) \leq 1$ it follows that $y'(\mathcal{S}_i) \leq \frac{1}{2}$. $\square$

We now can lower bound $\sum_{S \in \mathcal{S}} y'(S)w(S)$ as follows.

$$\begin{aligned}
 \sum_{S \in \mathcal{S}} y'(S)w(S) &= \sum_{1 \leq i \leq j} \sum_{S \in \mathcal{S}_i} y'(S)w(S) \\
 &\geq \sum_{1 \leq i \leq j} \sum_{S \in \mathcal{S}_i} y'(S)w(C_i) \\
 &= \sum_{1 \leq i \leq j} w(C_i) \sum_{S \in \mathcal{S}_i} y'(S) \\
 &= \sum_{1 \leq i \leq j} w(C_i)y'(\mathcal{S}_i).
 \end{aligned}$$

Putting the above together with Claim 3.1, we obtain the following.

$$\sum_{1 \leq i \leq j} w(C_i)y'(\mathcal{S}_i) \leq \sum_e w(e)d(e).$$

Assume without loss of generality that $w(C_1) \leq w(C_2) \leq \dots \leq w(C_j)$. Then, the algorithm outputs the union of the cuts $C_1, C_2, \dots, C_{k-1}$. Let $A = \sum_{i=1}^{k-1} w(C_i)$. Since the C_i -s are in increasing order of weight and $y'(\mathcal{S}_i) \leq 1/2$ for all i , we have the following.

$$\begin{aligned}
 \sum_{i=1}^j y'(\mathcal{S}_i)w(C_i) &\geq \sum_{i=1}^{k-1} w(C_i)/2 + w(C_k) \left(\sum_{i=1}^j y'(\mathcal{S}_i) - (k-1)/2 \right) \\
 &\geq A/2 + w(C_k) \frac{k-1}{2} (1/(1-1/|X|) - 1) \\
 &\geq A/2 + A/2 \cdot (1/(1-1/|X|) - 1) \\
 &\geq A/2 \cdot (1/(1-1/|X|)).
 \end{aligned}$$

Hence, we obtain that

$$A \leq 2(1-1/|X|) \sum_{i=1}^j y'(\mathcal{S}_i)w(C_i) \leq 2(1-1/|X|) \sum_e w(e)d(e).$$

From the above we get the following theorem.

Theorem 2. *The integrality gap of linear program (K') is at most $2(1-1/|X|)$.*

3.2 Lower Bound on the Integrality Gap

The integrality gap of (K') (and (K)) is not better than $2(1-1/|X|)$ even for the case where $k = 2$ and $X = V$, i.e., the global minimum cut problem. Consider the unit weight cycle on n vertices. Clearly, the optimal integral solution has to cut at least two edges to separate the cycle into two components. However, $d(e) = 1/(n-1)$ on each edge of the cycle and the induced shortest path distances on the rest of the edges, is feasible for both (K) and (K') , and the value of this solution is $n/(n-1)$. Hence, the integrality gap is $2(1-1/n)$.

4 An Exact Formulation for the Global Minimum Cut Problem

In the previous section we saw that linear program (K) has an integrality gap of $2 - 1/n$ for the 2-cut problem, i.e., for the global minimum cut problem. The authors of this paper are not aware of any exact formulation for the global minimum cut problem that does not rely on enumerating over all possible $s - t$ cuts in the graph. Here we give a bi-directed formulation of the global minimum cut problem. Given an undirected weighted graph $G = (V, E)$ let $G^b = (V, A)$ be the directed graph obtained by replacing each edge $e \in E$ between u and v by two directed arcs (u, v) and (v, u) . The weights of both (u, v) and (v, u) in G^b are set to $w(e)$. Let r be any vertex in $V(G)$. An *arborescence* in a directed graph rooted at a vertex r is a spanning out-tree from r (also known as a *branching*). Our formulation is based on G^b . For an arc $a \in A$, let $d(a) = 1$ if a is chosen to the cut, and let $d(a) = 0$ otherwise. The following is a valid integer program for the global minimum cut problem. The root r is chosen arbitrarily.

$$\begin{aligned}
 (B) \quad & \min \sum_a w(a) \cdot d(a) \quad \text{subject to :} \\
 & \sum_{a \in T} d(a) \geq 1 \quad \forall T : T \text{ arborescence rooted at } r \text{ in } G^b \\
 & d(a) \in \{0, 1\} \quad \forall a \in A
 \end{aligned}$$

Although the above integer program is similar to integer program (K) , we remark that it is not a valid formulation for the k -cut problem for $k > 2$.

We obtain a linear program by relaxing each variable $d(a)$ to be in $[0, 1]$. We show that the value of the linear program is exactly equal to the global minimum cut of the graph G . Note that linear program (B) can be solved in polynomial time by the Ellipsoid algorithm, since the separation oracle needed is the minimum cost arborescence problem in directed graphs, and this problem can be solved in polynomial time as shown by Edmonds [3]. In fact, Edmonds [3] showed that the arborescence polytope is integral and we use this to show that (B) is exact for the minimum cut problem. The proof is similar in outline to the one in Section 3, however, we use arborescences in place of spanning trees, and the result of Edmonds [3] on the integrality of the arborescence polytope in place of the GW algorithm. Let $\bar{d}$ be an optimal solution to (B) . Let G_d^b be the graph G^b equipped with $\bar{d}$ as costs on the edges of G^b . We find a minimum cost arborescence in G_d^b using the following formulation. For each arc a , variable $x(a) = 1$ if a belongs to the arborescence and 0 otherwise.

$$(AP) \quad \min \sum_a d(a) \cdot x(a) \quad \text{subject to :}$$

$$\begin{aligned} \sum_{a \in \delta(S)} x(a) &\geq 1 & \forall S : S \neq V \text{ and } r \in S \\ x(a) &\in [0, 1] & \forall a \end{aligned}$$

The dual of the above linear program is the following.

$$\begin{aligned} (AD) \quad \max \quad & \sum_S y(S) & \text{subject to :} \\ \sum_{S: a \in \delta(S)} y(S) &\leq d(a) & \forall a \\ y(S) &\geq 0 & \forall S : S \neq V \text{ and } r \in S \end{aligned}$$

Let $\bar{x}^*$ and $\bar{y}^*$ be optimal primal and dual solutions to (AP) and (AD) on the graph G_d^b . From the feasibility of $\bar{d}$, it follows that $\sum_a d(a)x^*(a) \geq 1$. From weak duality we therefore also obtain that $\sum_S y^*(S) \geq 1$. Let $\mathcal{S} = \{S \mid y^*(S) \geq 0\}$ be the set of all cuts with strictly positive dual values. Let $C \in \mathcal{S}$ be a cut such that $w(S)$ is the cheapest cut. We pick C as our solution. We now show that $w(C) \leq \sum_a w(a)d(a)$ which shows that the weight of the cut is at most the value of the optimal solution to (B). We see that

$$\begin{aligned} \sum_s y^*(S)w(S) &= \sum_S \sum_{a \in \delta(S)} w(a) \\ &= \sum_a w(a) \sum_{S: a \in \delta(S)} y^*(S) \\ &\leq \sum_a w(a)d(a). \end{aligned}$$

The last inequality follows from the feasibility of y^* . We have that $\sum_s y^*(S)w(S) \leq \sum_a w(a)d(a)$ and $\sum_S y^*(S) \geq 1$. Therefore, the weight of the cheapest cut is no more than $\sum_a w(a)d(a)$.

5 Conclusions

Our study of linear programming relaxations for the Steiner k -cut problem was partly motivated by the goal of obtaining an approximation algorithm for the k -cut problem with a ratio better than 2. This has been accomplished for the multiway cut problem by a strengthened LP relaxation [1]. Our results show that the available approximation techniques for the k -cut problem extend to the Steiner k -cut problem. In the process we have shown an interesting connection between laminar cut families obtained from the primal-dual algorithm of Goemans and Williamson [4] and their use in analyzing the LP relaxation for the Steiner k -cut problem. We hope that our ideas will be useful in developing and analyzing stronger LP relaxations that have integrality gap strictly smaller than 2 for the k -cut problem.

References

1. G. Călinescu, H. Karloff, and Y. Rabani. An improved approximation algorithm for multiway cut. *Journal of Computer and System Sciences*, 60:564–574, 2000.
2. E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. on Computing*, 23:864–894, 1994.
3. J. Edmonds. Optimum branchings. *J. Res. Nat. Bur. Standards*, B71:233–240, 1967.
4. M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM J. on Computing*, 24:296–317, 1995.
5. O. Goldschmidt and D. Hochbaum. Polynomial algorithm for the k -cut problem. *Mathematics of Operations Research*, 19:24–37, 1994 .
6. D. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43:601–640, 1996.
7. D. Karger, P. Klein, C. Stein, M. Thorup, and N. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. In *Proceedings of the 29th ACN Symposium on Theory of Computing*, pp. 668–678, 1999.
8. J. Naor and Y. Rabani. Approximating k -cuts. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 26–27, 2001.
9. R. Ravi and A. Sinha. Approximating k -cuts via Network Strength. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 621–622, 2002.
10. H. Saran and V.V. Vazirani. Finding k -cuts within twice the optimal. *SIAM J. on Computing*, 24:101–108, 1995.
11. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

MAX k -CUT and Approximating the Chromatic Number of Random Graphs

Amin Coja-Oghlan^{1*}, Cristopher Moore^{2**}, and Vishal Sanwalani²

¹ Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
`coja@informatik.hu-berlin.de`

² University of New Mexico, Albuquerque NM 87131
`{vishal,moore}@cs.unm.edu`

Abstract. We consider the MAX k -CUT problem in random graphs $G_{n,p}$. First, we estimate the probable weight of a MAX k -CUT using probabilistic counting arguments and by analyzing a simple greedy heuristic. Then, we give an algorithm that approximates MAX k -CUT within expected polynomial time. The approximation ratio tends to 1 as $np \rightarrow \infty$. As an application, we obtain an algorithm for approximating the chromatic number of $G_{n,p}$, $1/n \leq p \leq 1/2$, within a factor of $O(\sqrt{np})$ in polynomial expected time, thereby answering a question of Krivelevich and Vu, and extending a result of Coja-Oghlan and Taraz. We give similar algorithms for random regular graphs $G_{n,r}$.

1 Introduction and Results

Let $G = (V, E)$ be a graph, and let $k \geq 2$ be an integer. A k -cut of G is a partition of V into k sets $V_1, \dots, V_k$. The *weight* of a k -cut is the number of edges crossing the cut, i.e. connecting vertices in V_i and V_j for $i \neq j$. The MAX k -CUT problem asks for a k -cut of maximum weight. Though MAX k -CUT is NP-hard, a simple greedy algorithm achieves a $(1 - 1/k)$ -approximation.

The case $k = 2$, which is simply called the MAX CUT problem, has received the most attention. Goemans and Williamson [12] gave a semidefinite programming (“SDP”) algorithm with an approximation ratio of 0.87856. Håstad [16] proved that no polynomial-time algorithm can approximate MAX CUT within a factor > 0.9412 unless $P = NP$. Extending the methods of Goemans and Williamson, Frieze and Jerrum [9] achieved an approximation algorithm for MAX k -CUT for all $k \geq 2$, and proved an approximation guarantee tending to $1 - 1/k + 2(\log k)/k^2$ for large k . For $k = 3$, Goemans and Williamson gave an 0.836008-approximation algorithm [13].

In contrast to MAX k -CUT, no polynomial time algorithm is known that achieves a constant approximation ratio for the *Graph Coloring* problem. A *coloring* of a graph G is an assignment of colors to the vertices of G such that

* Research supported by the Deutsche Forschungsgemeinschaft (DFG FOR 413/1-1)

** Supported by NSF PHY-0071139 and Los Alamos National Laboratory.

adjacent vertices receive different colors. The *chromatic number* $\chi(G)$ is the minimum number of colors needed to color G . It is well-known that computing $\chi(G)$ is NP-hard. Indeed, Feige and Kilian [6] proved that, unless $\text{coRP}=\text{NP}$, no polynomial time algorithm can approximate $\chi(G)$ within $n^{1-\varepsilon}$.

Since these non-approximability results restrict what we can hope for in the worst case, it is reasonable to ask for algorithms that perform well in the average case, i.e. for random instances. In this paper, we will mainly consider the $G_{n,p}$ model of random graphs, pioneered by Erdős and Rényi. Given $p = p(n)$, $0 < p < 1$, the random graph $G_{n,p}$ is formed by including each of the $\binom{n}{2}$ possible edges independently with probability p . For example, the case $p = 1/2$ gives the uniform distribution over all graphs of order n . Though $G_{n,p}$ may fail to model some types of input instances appropriately, the combinatorial structure and algorithmic theory of $G_{n,p}$ are of fundamental interest (e.g. [10,17,19]).

The MAX k -CUT problem. In contrast to the chromatic number $\chi(G_{n,p})$, little is known about MAX k -CUT in $G_{n,p}$ (except in the case $k = 2$, i.e. the MAX CUT problem [2,4,18]). First, we give a sharp concentration result; its proof is via a standard application of Talagrand's inequality (cf. [17, p. 40]), and is omitted. We denote by $\text{MC}_k(G)$ the weight of a maximum k -cut in G .

Theorem 1. *Let $\mu = \mathbb{E}[\text{MC}_k(G_{n,p})]$. Then, for any $\xi > 0$, $\Pr[\text{MC}_k(G_{n,p}) \leq \mu - \xi] \leq 2 \exp(-\xi^2/(8\mu))$, and $\Pr[\text{MC}_k(G_{n,p}) \geq \mu + \xi] \leq 2 \exp(-\xi^2/(8(\mu + \xi)))$.*

The following theorem generalizes results in [2,4,18] by bounding $\text{MC}_k(G_{n,p})$ in the dense case $np \rightarrow \infty$ and in the sparse case $p = d/n$ where d is a large constant independent of n . Note that since a random k -cut includes $(1 - 1/k)$ of the edges, the constants A_k below tell us how much better the maximum k -cut is than a random one. We say that $G_{n,p}$ has a property P *with high probability* (w.h.p.) if $\lim_{n \rightarrow \infty} \Pr[G_{n,p} \text{ has property } P] = 1$.

Theorem 2. *Let k be a constant. If $p = p(n)$ such that $\lim_{n \rightarrow \infty} np(n) = \infty$, then w.h.p. $\text{MC}_k(G_{n,p}) = (1 - 1/k)n^2p/2 + A_k n^{3/2} \sqrt{p(1-p)} + o(n^{3/2} p^{1/2})$ where*

$$\frac{\sqrt{8}}{3} \sqrt{\frac{\log k}{k}} \left(1 - O\left(\frac{\log \log k}{\log k}\right) \right) \leq A_k \leq \sqrt{\frac{\log k}{k}} \sqrt{1 - \frac{1}{k}}. \quad (1)$$

If $p(n) = d/n$, then w.h.p. $\text{MC}_k(G_{n,p}) = (1 - 1/k)(dn/2) + A_{k,d} n \sqrt{d} + o(n)$, where $\lim_{d \rightarrow \infty} A_{k,d}$ obeys the same bounds as A_k in Eq. (1).

Since $\sqrt{8}/3 \approx 0.9428$, Theorem 2 w.h.p. determines $\text{MC}_k(G_{n,p})$ to within roughly 6% in the limit of large (but constant) k . We note that these upper bounds hold for all k , and that explicit lower bounds can be given for a given fixed k using the same proof techniques (see Section 2).

We turn now to computational complexity. There are two different types of algorithms for NP-hard random graph problems. First, there are algorithms that *always* run in polynomial time, and *almost always* output a good solution. We shall refer to such algorithms as *heuristics*. For example, the proof of Theorem 2 is based on the analysis of a simple greedy heuristic. On the other hand, there

are algorithms that guarantee some approximation ratio on *any* input instance, and which have a polynomial *expected* running time when applied to $G_{n,p}$. Here we say that an algorithm A runs in *polynomial expected time* if there is a constant $l > 0$ such that $\sum_G R_A(G) \Pr[G_{n,p} = G] = O(n^l)$, where $R_A(G)$ is the running time of A on input G and the sum ranges over all graphs G of order n . (In a sense, Levin’s concept [24] of polynomial average running time is a relaxation of the aforementioned concept.)

Theorem 3. *Suppose that $p \geq Ck^2x^2/n$, where $C > 0$ is a sufficiently large constant (independent of n, k) and $x \geq 1$. There exists an algorithm **ApproxMkC** that finds a k -cut of weight $\geq (1 - 1/(kx))\text{MC}_k(G)$ for any input graph G , and which runs in polynomial expected time.*

Note that in the dense case $np \rightarrow \infty$, the approximation ratio of **ApproxMkC** tends to 1. To achieve this approximation ratio, **ApproxMkC** combines a greedy heuristic that produces sufficiently good solutions on “typical” instances, with an SDP-relaxation SDP_k of MAX k -CUT due to Frieze and Jerrum [9] (see Section 3 for details). The value of SDP_k indicates whether the input graph G is “typical” or not. In the exceptional case, **ApproxMkC** takes superpolynomial time in order to find a solution that is within the desired approximation ratio. The main technical result of this paper is the following bound on the probable value of $\text{SDP}_k(G_{n,p})$, and may be of independent interest.

Theorem 4. *Suppose that $p \geq 1/n$. There exists a constant $\lambda > 0$ (independent of p, n , and k) such that w.h.p. $\text{SDP}_k(G_{n,p}) \leq (1 - 1/k)n^2p/2 + \lambda n^{3/2}p^{1/2}$.*

The case $k = 2$, $p = 1/2$ of Theorem 4 has previously been treated in [5]. Since $\text{SDP}_k(G) \geq \text{MC}_k(G)$ for all graphs G , the lower bound in Theorem 2 yields a lower bound on $\text{SDP}_k(G_{n,p})$. Our proof that **ApproxMkC** runs in polynomial expected time relies on the following concentration result on SDP_k .

Theorem 5. *Suppose that $n^2p \rightarrow \infty$, and let μ be a median of $\text{SDP}_k(G_{n,p})$. Then for any $\xi \geq \max\{\sqrt{\mu}, 10\}$ the following estimates hold.*

$$\begin{aligned} \Pr[\text{SDP}_k(G_{n,p}) \geq \mu + \xi] &\leq 30 \exp(-\xi^2/(4\mu + 8\xi)) \\ \Pr[\text{SDP}_k(G_{n,p}) \leq \mu - \xi] &\leq 3 \exp(-\xi^2/(8\mu)) \end{aligned}$$

We also obtain an algorithm that approximates the MAX k -CUT of *random regular graphs*. Let $G_{n,r}$ denote the set of all r -regular graphs of order n , equipped with the uniform distribution.

Theorem 6. *Suppose that $Ck^2x^2 \leq r = o(n^{1/2})$, where $C > 0$ is some constant and $x \geq 1$. There exists an algorithm **RegMkC** that on any input graph G computes a k -cut of weight $\geq (1 - 1/(kx))\text{MC}_k(G)$, and which runs in polynomial expected time on $G_{n,r}$.*

The graph coloring problem. Bollobas and Łuczak computed the probable value of $\chi(G_{n,p})$: With high probability,

$$\chi(G_{n,1/2}) \sim \frac{n}{2 \log_2 n} \text{ and } \chi(G_{n,p}) \sim \frac{np}{2 \log(np)} \text{ for } C/n \leq p = o(1), \quad (2)$$

where $C > 0$ denotes some large constant (cf. [17]). Moreover, Achlioptas and Friedgut [1] showed the following sharp threshold result: for any constant k there exist constants d_k^+, d_k^- such that for $p > (1 + \varepsilon)d_k^+/n$, the random graph $G_{n,p}$ is w.h.p. not k -colorable, whereas for $p < (1 - \varepsilon)d_k^-/n$, $G_{n,p}$ is w.h.p. k -colorable.

Concerning algorithms, it is known that (e.g. in the case $p = 1/2$) a linear-time greedy heuristic exists which w.h.p. uses at most $(1 + o(1))n/\log_2 n$ colors. Hence, w.h.p. it achieves a 2-approximation (cf. [17]). However, since this greedy heuristic does not compute any lower bound on the chromatic number, it cannot distinguish between such input graphs G with $\chi(G)$ large (as in (2)) and input graphs with $\chi(G)$ much smaller. Therefore, it fails to guarantee any non-trivial approximation ratio.

Graph coloring algorithms with polynomial expected running time which provide a performance guarantee on all instances have been proposed by several authors (see [22] for a recent survey). However, only [21,3] treat the $G_{n,p}$ model. Similarly to the case of MAX k -CUT, the crucial step is to exhibit an efficiently computable lower bound on $\chi(G)$. Employing spectral techniques, Krivelevich and Vu [21] achieved an approximation ratio of $O(\sqrt{np}/\log(np))$, provided $p > n^{-1/2}$. Moreover, they ask [21,22] whether an algorithm with similar performance exists for smaller values of p . Lower-bounding the chromatic number via the Lovász number ϑ , Coja-Oghlan and Taraz [3] gave an $O(\sqrt{np})$ -approximation for the case $p > (\log^7 n)/n$, and an exact algorithm for $p \leq 1.01/n$. Our main result on graph coloring closes the remaining gap.

Theorem 7. *Let $1/n \leq p \leq 1/2$. There exists an algorithm $\text{ApproxColor}(G)$ that for any input graph G finds a coloring with at most $C'(np)^{1/2}\chi(G)$ colors, and which runs in polynomial expected time. Here C' denotes a constant independent of n and p .*

Note that the approximation ratio $C'(np)^{1/2}$ gets better as p decreases. In order to lower-bound $\chi(G)$, $G = (V, E)$, ApproxColor makes use of the following (rather obvious) connection between MAX k -CUT and graph coloring: If $\chi(G) \leq k$, then $\text{SDP}_k(G) \geq \text{MC}_k(G) = |E|$. Indeed, the analysis of ApproxColor relies on our bound on $\text{SDP}_k(G_{n,p})$ (Theorem 4). Once more, we obtain a result on random regular graphs:

Theorem 8. *Let $r \geq r_0$ for a certain constant r_0 . There exists an algorithm $\text{RegColor}(G)$ that for any r -regular graph G finds a coloring in $\leq C'r^{1/2}\chi(G)$ colors, and which runs in polynomial expected time on $G_{n,r}$. Again, C' denotes a constant independent of n and p .*

2 Proof of Theorem 2

The proof of the upper bound uses the first moment method, and is omitted. The proof of the lower bound is similar to those for MAX CUT in [4,18], and relies on analyzing a simple greedy heuristic **Greedy**. Initially all vertices are uncolored. At the beginning of the t 'th step, t vertices are already colored and **Greedy** chooses an uncolored vertex v at random. Let $u_i(t)$ for $i \in \{1, \dots, k\}$

denote the number of v 's neighbors which are colored with color i just before the t 'th step, and $u(t) = \sum_i u_i(t)$. Then **Greedy** assigns to v the color j such that $u_j(t) = \min_i u_i(t) \equiv m(t)$, breaking ties randomly. Clearly, each step of **Greedy** adds $u(t) - m(t) \equiv z(t)$ edges to the cut. By linearity of expectation, **Greedy** produces a cut of expected size $\sum_{t=0}^{n-1} \mathbb{E}[z(t)]$ where $\mathbb{E}[z(t)] = \mathbb{E}[u(t)] - \mathbb{E}[m(t)]$.

Since every vertex is connected to v independently with probability p , we have $\mathbb{E}[u(t)] = pt$. Calculating $\mathbb{E}[m(t)]$ is slightly harder. The worst case for **Greedy** is if, at the beginning of each step t , an equal number t/k of vertices are colored with each of the k colors. In that case $m(t)$ is the minimum of k independent random variables $u_i(t)$, each of which is binomially distributed as the sum of t/k trials with probability of success p .

Let α be some variable (dependent on np), such that $\alpha np/k \rightarrow \infty$ and assume that $t \geq \alpha n$. Then we have $pt/k \rightarrow \infty$. Therefore, we can use the normal approximation to the binomial distribution. The expected value of the minimum of k independent and identically distributed normal variables with mean μ and variance σ^2 is $\mu - \sigma r_k$ where r_k is the expected maximum of k independent normal variables with mean 0 and variance 1. Since for the $u_i(t)$ we have $\mu = pt/k$ and $\sigma^2 = p(1-p)t/k$,

$$\mathbb{E}[m(t)] \leq \frac{pt}{k} - r_k \sqrt{\frac{p(1-p)t}{k}}$$

so

$$\mathbb{E}[z(t)] \geq \left(1 - \frac{1}{k}\right) pt + r_k \sqrt{\frac{p(1-p)t}{k}}. \quad (3)$$

Ignoring the first αn steps, clearly **Greedy** finds a cut whose expected size is at least $Z \equiv \sum_{t=\alpha n}^{n-1} \mathbb{E}[z(t)]$. Since $\mathbb{E}[z(t)]$ is positive and increases monotonically, we can replace this sum by an integral with an additive error equal to its largest term, which is $O(np)$. Setting $\alpha = o(1/\sqrt{np})$ and integrating Eq. (3) over t gives

$$\mathbb{E}[Z] \geq \left(1 - \frac{1}{k}\right) \frac{n^2 p}{2} + \frac{2r_k}{3\sqrt{k}} n^{3/2} \sqrt{p(1-p)} - o(n^{3/2} p^{1/2})$$

so $A_k \geq 2r_k/(3\sqrt{k})$. From [20] we have

$$r_k \geq \sqrt{2 \log k} \left(1 - \frac{\log \log k + \log 4\pi}{4 \log k}\right) + O(1/\log k)$$

which yields the lower bound of Eq. (1). For $k \leq 5$ it is possible to express r_k exactly in terms of elementary functions, yielding $A_2 = (1/3)\sqrt{2/\pi}$, $A_3 = 1/\sqrt{3\pi}$, $A_4 = (\pi + 2 \sin^{-1} \frac{1}{3})/(2\pi^{3/2})$, and $A_5 = \sqrt{5}(\pi + 6 \sin^{-1} \frac{1}{3})/(6\pi^{3/2})$. Finally, Theorem 1 implies that w.h.p. $\text{MC}_k(G_{n,p}) \geq \mathbb{E}[Z] - (n \log \log n) p^{1/2} = \mathbb{E}[Z] - o(n^{3/2} p^{1/2})$. Setting $p = d/n$ yields the bounds for the sparse case.

3 The Large Deviation Result on SDP_k

We will use the following notation in the remainder of the paper. If A is a symmetric $n \times n$ matrix, then we let $\lambda_1(A) \geq \dots \geq \lambda_n(A)$ be the eigenvalues

of A . By $\|x\| = ({}^t x x)^{1/2}$ we denote the L^2 -norm. If $X = (x_{ij})_{i,j}$ is an $n \times n$ matrix, then $\text{diag}(X) = ({}^t x_{11}, \dots, x_{nn}) \in \mathbb{R}^n$ is the diagonal of X . Conversely, if $x \in \mathbb{R}^n$, then $\text{diag}(x)$ denotes the matrix with diagonal x and all off-diagonal entries = 0. We let $\mathbf{1}$ be the vector with all entries equal to 1 (in any dimension). If $A = (a_{ij}), B = (b_{ij})$ are $n \times n$ matrices, then we let $(A|B) = \sum_{i,j=1}^n a_{ij} b_{ij}$.

Though our main interest is in graphs, we define the SDP-relaxation SDP_k of MC_k in terms of multigraphs. Let G be an (undirected) multigraph with vertex set $V = \{1, \dots, n\}$. The *adjacency matrix* of G is the symmetric matrix $A = A(G) = (a_{ij})_{i,j \in V}$, where a_{ij} is the number of edges connecting i and j in G if $i \neq j$, and a_{ii} is twice the number of self-loops at vertex i . The following semidefinite program SDP_k is due to Frieze and Jerrum [9]:

$$\max \sum_{i < j} a_{ij} \frac{k-1}{k} (1 - {}^t v_i v_j) \text{ s.t. } \|v_i\| = 1, {}^t v_i v_j \geq -\frac{1}{k-1}, \quad (4)$$

where $v_1, \dots, v_n$ range over $\mathbb{R}^n$. If we let $L = L(G) = \text{diag}(A\mathbf{1}) - A$ denote the *Laplacian* of G , then in matrix notation SDP_k reads

$$\max \frac{k-1}{2k} (L|X) \text{ s.t. } \text{diag}(X) = \mathbf{1}, x_{ij} \geq -\frac{1}{k-1}, X \geq 0, \quad (5)$$

where X ranges over $n \times n$ positive semidefinite matrices. Clearly, $\text{SDP}_k(G) \geq \text{MC}_k(G)$ for all G . We shall apply *Talagrand's inequality* (cf. [17, p. 44]):

Theorem 9. *Let $A_1, \dots, A_N$ be probability spaces, and $\Lambda = A_1 \times \dots \times A_N$. Let $A, B \subset \Lambda$ be measurable sets such that for some $t \geq 0$ the following condition is satisfied: For every $b \in B$ there is $\alpha = (\alpha_1, \dots, \alpha_N) \in \mathbb{R}^N \setminus \{0\}$ such that for all $a \in A$ we have $\sum_{a_i \neq b_i} \alpha_i \geq t \sqrt{\sum_{i=1}^N \alpha_i^2}$, where a_i (respectively b_i) denotes the i 'th coordinate of a (respectively b). Then $\Pr[A] \Pr[B] \leq \exp(-t^2/4)$.*

The following lemma is the key ingredient to the proof of Theorem 5.

Lemma 10. *Let μ be such that $\Pr[\text{SDP}_k \leq \mu] \geq \varepsilon$. Let $\mu_0, \xi > 0$. Then $\Pr[\mu + \xi \leq \text{SDP}_k(G_{n,p}) \leq \mu_0] \leq \exp(-\xi^2/(5\mu_0))/\varepsilon$.*

Proof. Let the random variable $A_{ij} : G_{n,p} \rightarrow \{0, 1\}$, $1 \leq i < j < n$, take the value 1 if the edge $\{i, j\}$ is present in $G_{n,p}$, and 0 otherwise. Since $(A_{ij})_{i < j}$ are mutually independent, we can identify $G_{n,p}$ with the product space $\Lambda = \prod_{i < j} A_{ij}$. Clearly, A_{ij} is the ij -entry of the adjacency matrix of $G_{n,p}$. Let $A = \{G \mid \text{SDP}_k(G) \leq \mu\}$ and $B = \{G \mid \mu + \xi \leq \text{SDP}_k(G) \leq \mu_0\}$. Let $H \in B$, and let $(b_{ij})_{i,j}$ be the adjacency matrix of H . Let $\alpha_{ij} = b_{ij} \frac{k-1}{k} (1 - {}^t v_i v_j)$, where $v_1, \dots, v_n$ are feasible vectors that maximize $\text{SDP}_k(H)$. Then $0 \leq \alpha_{ij} \leq 1$. Therefore, $\sum_{i < j} \alpha_{ij}^2 \leq \sum_{i < j} \alpha_{ij} = \text{SDP}_k(H) \leq \mu_0$. Now let $G \in A$, and let (a_{ij}) be the adjacency matrix of G . Let $\beta_{ij} = a_{ij} \alpha_{ij}$. Then $\sum_{i < j} \beta_{ij} \leq \text{SDP}_k(G) \leq \mu$. On the other hand, $\sum_{i < j} \alpha_{ij} = \text{SDP}_k(H) \geq \mu + \xi$, whence $\xi \leq \sum_{i < j} \alpha_{ij} - \sum_{i < j} \beta_{ij} = \sum_{i < j, a_{ij} \neq b_{ij}} \alpha_{ij}$. Let $t = \xi/\sqrt{\mu_0}$. Then $\sum_{a_{ij} \neq b_{ij}} \alpha_{ij} \geq t \cdot (\sum_{i < j} \alpha_{ij}^2)^{1/2}$, whence by Talagrand's inequality, $\Pr[A] \Pr[B] \leq \exp(-t^2/4) = \exp(-\xi^2/(4\mu_0))$. The lemma follows from the fact that $\Pr[A] \geq \varepsilon$. $\square$

By the lemma, we obtain the upper tail bound in Theorem 5 by estimating the geometric sum $\Pr[\mu + \xi \leq \text{SDP}_k(G_{n,p})] \leq 2 \sum_{l=1}^{\infty} \Pr[\mu + l\xi \leq \text{SDP}_k \leq \mu + (l+1)\xi]$. Similar arguments prove the lower tail bound.

4 The Upper Bound on SDP_k

The standard method to upper-bound a relaxation of a maximization problem such as SDP_k is to transform the SDP into a minimization problem via SDP duality. In this way we obtain an upper bound on SDP_k in terms of eigenvalues. Since we can estimate the eigenvalues of “dense” random graphs quite well, this approach rather immediately gives the desired bound in the case $p > (\log^7 n)/n$.

Bounding SDP_k via eigenvalues. Let G be a multigraph with Laplacian $L = L(G)$. The following SDP-relaxation SMC of MAX CUT is due to Goemans and Williamson [12]: $\max(L/4|X|)$ s.t. $\text{diag}(X) = \mathbf{1}$, $X \geq 0$, i.e. X ranges over $n \times n$ positive semidefinite matrices. The dual semidefinite program DSMC is

$$\min^t \mathbf{1}u \text{ s.t. } L/4 + Z - \text{diag}(u) \geq 0, \quad u \in \mathbb{R}^n, \quad Z \geq 0.$$

By (weak) SDP-duality, the value of DSMC is an upper bound on the value of SMC. Poljak and Rendl [26] gave an equivalent formulation of DSMC:

Lemma 11. *We have $\min_{u \in \mathbb{R}^n, \mathbf{1} \perp u} n\lambda_1(\text{diag}(u) + L/4) = \text{DSMC}(G)$.*

From this we obtain the following spectral bound on SDP_k .

Lemma 12. *Suppose that the multigraph $G = (V, E)$ does not contain multiple loops. Then $\text{SDP}_k(G) \leq (1 - 1/k)(|E| + n - (n/2)\lambda_n(A(G)))$.*

Sketch of proof. Since G does not contain multiple loops, Lemma 11 entails that $\text{SMC}(G) \leq (n/4)(d - \lambda_n(A) + 2)$, where $d = 2|E|/n$. Let $\text{SDP}'_k = 2(1 - 1/k)\text{SMC}$. Comparing the constraints of SDP_k and SMC, we find $\text{SDP}_k \leq \text{SDP}'_k$. $\square$

To prove Theorem 4 in the case $p > (\log^7 n)/n$, note that the entries a_{ij} , $i < j$, of the adjacency matrix A of $G_{n,p}$ are independent random variables. If $p > (\log^7 n)/n$, the arguments of [11] apply, and show that with high probability $\max_{i \geq 2} |\lambda_i(A)| \leq 4\sqrt{np}$. Since for $G_{n,p}$ $|E|$ is concentrated about its mean $\binom{n}{2}p$, our bound follows from Lemma 12. The following proposition, however, shows that the above approach breaks down in the sparse case.

Proposition 13. *Let $p = d/n$ for some constant $d > 1$. Then w.h.p.*

$$-\lambda_n(A(G_{n,p})) = \Omega(\sqrt{\log n / \log \log n}).$$

Hence, the bound in Lemma 12 is $> n^2 p/2$.

Sparse random graphs. Proposition 13 shows that we need a different approach in the sparse case. Let us assume that $C/n \leq p = o(n^{-1/2})$, for some large constant C . To bound $\text{SDP}_k(G_{n,p})$ for small values of p , we shall exhibit a class $\mathcal{R} \subset G_{n,p}$ of graphs G with the following properties.

- A.** $\Pr[G_{n,p} \in \mathcal{R}] \geq \exp(-30n)$
- B.** For all $G \in \mathcal{R}$ we have $\text{SDP}_k(G) \leq (1 - 1/k)n^2 p/2 + \lambda n^{3/2} p^{1/2}$, where $\lambda > 0$ denotes some constant.

Before we sketch how to construct $\mathcal{R}$, let us observe that Theorem 4 immediately follows from the existence of some $\mathcal{R}$ with A. and B. and Theorem 5. For let μ be a median of $\text{SDP}_k(G_{n,p})$, and let $\lambda' > 0$ be a sufficiently large constant. Assume for contradiction that $\mu > (1 - 1/k)n^2p/2 + (\lambda + \lambda')n^{3/2}p^{1/2}$. Then

$$\begin{aligned} \exp(-30n) &\leq \Pr[\text{SDP}_k(G_{n,p}) \leq (1 - 1/k)n^2p/2 + \lambda n^{3/2}p^{1/2}] \\ &\leq \Pr[\text{SDP}_k(G_{n,p}) < \mu - \lambda' n^{3/2}p^{1/2}] < \exp(-30n) , \end{aligned} \quad (6)$$

provided λ' is large enough. Thus, we have $\mu \leq (1 - 1/k)n^2p/2 + \lambda'' n^{3/2}p^{1/2}$ for some constant λ'' . Invoking Theorem 5 once more completes the proof of Theorem 4.

Our construction of the class $\mathcal{R}$ is probabilistic. More precisely, we shall prove that most graphs with certain degree sequences satisfy property B. Let $\bar{d}$ be an even integer satisfying $|\bar{d} - np| \leq 1$. Let $\Delta = (\Delta_1, \dots, \Delta_n) \in \mathbb{Z}^n$. We define a sequence $d = d(\Delta) = (d_1, \dots, d_n)$ as follows. Let $b_0 = 0$, $b_i = i\bar{d} + \Delta_i$, $i = 1, \dots, n-1$, and $b_n = \bar{d}n$, and set $d_i = b_i - b_{i-1}$. Then $\sum_{i=1}^n d_i = b_n - b_0 = \bar{d}n$. We call a sequence $d(\Delta)$ *almost regular*, if $|\Delta_i| \leq \sqrt{\bar{d}}$ for all i . Then $|d_i - \bar{d}| \leq 2\sqrt{\bar{d}}$.

If G is a multigraph, and d_i is the degree of vertex $i \in V = \{1, \dots, n\}$, then we call $(d_v)_{v \in V}$ the *degree sequence* of G . Let us call a (simple) graph *almost regular*, if its degree sequence is almost regular (i.e. $d = d(\Delta)$, $|\Delta_i| \leq \sqrt{\bar{d}}$). The following lemma shows that there are many almost regular graphs.

Lemma 14. *We have $\Pr[G_{n,p} \text{ is almost regular}] \geq \exp(-25n)$.*

Sketch of proof. Let $d = d(\Delta)$ be an almost regular degree sequence. Using the formulas derived in [25], one can compute that the number of graphs with degree sequence d is $\geq \exp(-2n) \#G_{n,\bar{d}}$. Hence, there are

$$\geq (4\bar{d})^{\frac{n-1}{2}} \exp(-2n) \#G_{n,\bar{d}} \geq (4\bar{d})^{\frac{n-1}{2}} \exp(-3n) \left(\frac{e}{\bar{d}}\right)^{\bar{d}n/2} (2\pi\bar{d})^{-n/2} n^{\bar{d}n/2} \quad (7)$$

almost regular graphs. Comparing Eq. (7) with the number of all graphs with $\bar{d}n/2$ edges and estimating $\Pr[|E| = \bar{d}n/2]$ proves the lemma. $\square$

Let us fix an almost regular degree sequence d . By $G_{n,d}$ we denote the set of all graphs with degree sequence d , equipped with the uniform distribution. We shall prove that most $G \in G_{n,d}$ satisfy property B. To study the random graph $G_{n,d}$ with degree sequence d , we invoke the *configuration model* (cf. [25]). Let $W = W(d) = \{(v, t) \mid v \in V, t = 1, \dots, d(v)\}$; the elements of W are called *half edges*. A configuration σ is a partition of W into $\bar{d}n/2$ pairs. Thus, to each half-edge (u, v) , σ assigns another half-edge $\sigma(u, v) \neq (u, v)$, and $\sigma^2 = \text{id}$. We say that (u, v) and $\sigma(u, v)$ form an *edge*. By $\mathcal{C} = \mathcal{C}(d)$ we denote the set of all configurations w.r.t. d . For brevity, we let $\mathcal{C}(\bar{d}) = \mathcal{C}(\bar{d}, \dots, \bar{d})$. Then $|\mathcal{C}(d)| = (\bar{d}n - 1)!!$ depends only on the average degree $\bar{d}$.

To each $\sigma \in \mathcal{C}(d)$, the canonical map $\pi : W \rightarrow V$ assigns a multigraph $\pi(\sigma)$ with degree sequence d . If we equip $\mathcal{C} = \mathcal{C}(d)$ with the uniform distribution, then conditional on $G_{n,d}$, π induces the uniform distribution. Let $G_{n,d}^*$ denote the set of all multigraphs with degree sequence d , equipped with the distribution induced by π (in general, this is *not* the uniform distribution).

Before we come to simple graphs $G_{n,d}$, let us bound $\text{SDP}_k(G_{n,d}^*)$. Our argument relies on the spectral bound in Lemma 12 (note that w.h.p. $G_{n,d}^*$ has no multiple loops). In order to estimate $\lambda_n(A(G_{n,d}^*))$, we need the following lemma.

Lemma 15. *Let d be an almost regular degree sequence. Then there is a constant $\gamma > 0$ such that with high probability the adjacency matrix $A = A(\pi(\sigma))$, $\sigma \in \mathcal{C} = \mathcal{C}(d)$, satisfies $|{}^t(Ax)y| \leq \gamma\sqrt{\bar{d}}$, for all vectors $x \perp \mathbf{1}$, $y \perp \mathbf{1}$, $\|x\|, \|y\| \leq 1$.*

The proof of Lemma 15 goes along the lines of the estimate on the eigenvalues of random regular graphs by Kahn and Szemerédi [8]. Using some linear algebra, we can estimate the eigenvalues of $A = A(\pi(\sigma))$ as follows.

Lemma 16. *Let σ be such that for all vectors $x \perp \mathbf{1}$, $y \perp \mathbf{1}$, $\|x\|, \|y\| \leq 1$, we have $|{}^t(Ax)y| \leq \gamma\sqrt{\bar{d}}$. Then $\max_{i \geq 2} |\lambda_i(A)| \leq (\gamma + 2)\sqrt{\bar{d}}$.*

Combining Lemmata 12, 15 and 16, we conclude that there is some constant $\lambda > 0$ such that w.h.p. $\text{SDP}_k(G_{n,d}^*) \leq (1 - 1/k)\bar{d}n/2 + \lambda\sqrt{\bar{d}n}$. If $\bar{d} = O(1)$, then we immediately obtain a bound on $\text{SDP}_k(G_{n,d})$, since $\Pr[G_{n,d}^* \text{ is simple}] = \Omega(1)$. However, since $\Pr[G_{n,d}^* \text{ is simple}] = o(1)$ in the case $\bar{d} \rightarrow \infty$, we need another large deviation result. The proof of the following lemma relies on martingales.

Lemma 17. *Let μ be the expectation of SDP_k over $G_{n,d}^*$. Then for any $t > 0$,*

$$\Pr[|\text{SDP}_k(G_{n,d}^*) - \mu| > t] \leq 2 \exp(-t^2/(64\bar{d}n)) .$$

By [25], in the case $\bar{d} = o(n^{1/2})$, we have $\Pr[G_{n,d}^* \text{ is simple}] \geq \exp(-n)$. Hence, using Lemma 17 instead of Theorem 5, a similar estimate as Eq. (6) shows that w.h.p.

$$\text{SDP}_k(G_{n,d}) \leq (1 - 1/k)\bar{d}n/2 + \lambda'\sqrt{\bar{d}n}, \quad (8)$$

for some constant λ' . Finally, we let our class $\mathcal{R}$ consist of all graphs $G_{n,d}$ that satisfy Eq. (8), where d ranges over all almost regular degree sequences. Then property B. is satisfied by construction, and A. follows from Lemma 14.

Remark 18. An alternative way to prove Theorem 4 would be to combine spectral techniques as recently proposed by Feige and Ofek [7] with Theorem 5. Indeed, in [7] the weight of a MAX CUT of $G = G_{n,p}$ is bounded via the smallest eigenvalue of the adjacency matrix A' of the graph G' obtained from G by removing all vertices of degree $\geq (1 + \varepsilon)np$. Bounding $\lambda_n(A')$ using the method of Kahn and Szemerédi [8], Feige and Ofek obtain a heuristic that with probability $1 - \exp(-\Omega(np))$ achieves a similar approximation ratio as our algorithm ApproxMkC below.

5 Approximating MAX k -CUT

In Section 2, we analyzed a simple algorithm **Greedy** on random graphs $G_{n,p}$. It is not hard to see that on any input graph G the greedy heuristic finds a k -cut of weight $\geq (1 - 1/k)\text{MC}_k(G)$. To obtain an algorithm as claimed in Theorem 3, we shall combine the greedy procedure with the upper bound SDP_k . Moreover, we need an exact algorithm for MAX k -CUT.

Lemma 19. *For any k there is an algorithm **DynamicCut** that runs in time $O(3^{(1+o(1))n})$ and on any input graph outputs a maximum k -cut.*

The algorithm **DynamicCut** is based on dynamic programming. Our approximation algorithm for MAX k -CUT is as follows.

Algorithm 20. **ApproxMkC**(G)

Input: A graph $G = (V, E)$. *Output:* A k -cut of G .

1. If $|E| < \frac{n^2 p}{2} - 2n^{3/2} p^{1/2}$, then go to 3. Otherwise, run the greedy algorithm on input G . Let $\mathcal{C}$ be the resulting k -cut.
2. If $\text{SDP}_k(G) > \frac{(k-1)n^2 p}{2k} + cn^{3/2} p^{1/2}$ for a certain constant c (independent of k , G , n , and p), then go to 3. Otherwise, terminate with output $\mathcal{C}$.
3. Run **DynamicCut**(G, k) and output the resulting MAX k -CUT.

The first two steps of **ApproxMkC** are polynomial, since SDP_k can be solved in polynomial time within sufficient precision (e.g. via the ellipsoid method [15]).

Lemma 21. *Suppose that $c_0/n \leq p \leq 1/2$ for some sufficiently large constant c_0 . Then **ApproxMkC** has a polynomial expected running time on $G_{n,p}$.*

Sketch of proof. By Chernoff bounds, the probability that step 1 branches to step 3 is $o(4^{-n})$. Theorem 5 entails that $\Pr[\text{SDP}_k(G) > \frac{(k-1)n^2 p}{2k} + cn^{3/2} p^{1/2}] \leq \exp(-2n) = o(4^{-n})$, if c is large enough. Since step 3 consumes time $o(4^n)$, the assertion follows. $\square$

The fact that **ApproxMkC** guarantees the desired approximation ratio follows by estimating the quotient $\frac{(1-1/k)n^2 p/2 - 2n^{3/2} p^{1/2}}{(1-1/k)n^2 p/2 + cn^{3/2} p^{1/2}}$. The algorithm **RegMkC** is similar to **ApproxMkC**; its analysis relies on Eq. (8) and Lemma 17, instead of Theorem 4 and Theorem 5.

6 Approximate Graph Coloring

Choosing the constant C' mentioned in Theorem 7 large enough, we may assume that $p \geq c_0/n$ for some constant $c_0 > 0$. Our graph coloring algorithm makes use of a procedure **CoreColor** established in [3].

Lemma 22. *There is an algorithm **CoreColor** that on input $G = G_{n,p}$ finds a coloring in $\leq \max\{10np, \chi(G)\}$ colors, and runs in linear expected time.*

As mentioned in the introduction, the coloring algorithm **ApproxColor** uses SDP_k to lower bound the chromatic number. The idea of lower-bounding the chromatic number via MAX k -CUT first occurs in [14], where an algorithm for deciding 3-colorability is given. However, the algorithm [14] relies on the (worst-case) approximation guarantee by Frieze and Jerrum [9] instead of a bound on the probable value of SDP_k . It is easily seen that this approach leads to an $O(np)^{3/4}$ -approximation, but seems not to give the $O(np)^{1/2}$ stated in Theorem 7.

Algorithm 23. **ApproxColor**(G)

Input: A graph $G = (V, E)$. *Output:* A coloring of G .

1. Run **CoreColor**(G). Let $\mathcal{C}$ be the resulting coloring of G .
2. Let $k = c_1(np)^{1/2}$ for some (small) constant $c_1 > 0$. If $\text{SDP}_k(G) < |E| - 1$, then terminate with output $\mathcal{C}$.
3. Find an optimal coloring using Lawler's algorithm [23] (in time $o(\exp(n))$).

The following lemma is a consequence of Theorem 4 and Theorem 5, and implies that **ApproxColor** has a polynomial expected running time.

Lemma 24. *Let $G = (V, E)$ be distributed as $G_{n,p}$. Suppose that $c_0/n \leq p \leq 1/2$ for a certain constant c_0 . If $c_1 > 0$ is small enough, then $\Pr[\text{SDP}_k(G_{n,p}) < |E| - 1] \geq 1 - \exp(-2n)$.*

Let us prove that **ApproxColor** achieves an approximation ratio of $O(\sqrt{np})$. Let $G = (V, E)$ be any graph. If $\chi(G) \leq k = c_1(np)^{1/2}$, then clearly $\text{SDP}_k(G) \geq \text{MC}_k(G) = |E|$. Thus, if $\text{SDP}_k(G) < |E|$, then $\chi(G) \geq k$. Since **CoreColor**(G) uses at most $\max\{10np, \chi(G)\}$ colors, **ApproxColor** guarantees an approximation ratio of $10(np)^{1/2}/c_1$.

To obtain an algorithm **RegColor** as in Theorem 8, replace **CoreColor** with the well-known greedy procedure for graph coloring, and let $k = c_2r^{1/2}$ for some small constant $c_2 > 0$. (On r -regular graphs, the greedy algorithm uses $\leq 2r$ colors.) In the analysis, use Eq. (8) and Lemma 17 instead of Theorem 4 and Theorem 5.

Acknowledgement. The first author is grateful to M. Krivelevich and Till Nierhoff for helpful discussions.

References

1. Achlioptas, D. and Friedgut, E.: A sharp threshold for k -colorability. *Random Structures & Algorithms* **14** (1999) 63–70.
2. Bertoni, A., Campadelli, P., Posenato, R.: An upper bound for the maximum cut mean value. *Springer LNCS* **1335** (1997) 78–84.
3. Coja-Oghlan, A. and Taraz, A.: Colouring random graphs in expected polynomial time. *Proc. 20th STACS* (2003) 487–498.
4. Coppersmith, D., Gamarnik, D., Hajiaghayi, M., Sorkin, G.B.: Random MAX SAT, random MAX CUT, and their phase transitions. *Proc. 14th SODA* (2003) 329–337.
5. Delorme, C. and Poljak, S.: Laplacian eigenvalues and the maximum cut problem. *Math. Programming* **62** (1993) 557–574.
6. Feige, U. and Kilian, J.: Zero knowledge and the chromatic number. *Proc. 11th IEEE Conf. Comput. Complexity* (1996) 278–287.
7. Feige, U., Ofek, E.: Spectral techniques applied to sparse random graphs, report MCS03-01, Weizmann Institute of Science (2003).
8. Friedman, J., Kahn, J., Szemerédi, E.: On the second eigenvalue in random regular graphs. *Proc. 21st ACM Symp. Theory of Computing (STOC)* (1989) 587–598.
9. Frieze, A. and Jerrum, M.: Improved approximation algorithms for MAX k -CUT and MAX BISECTION. *Algorithmica* **18** (1997) 61–77.
10. Frieze, A. and McDiarmid, C.: Algorithmic theory of random graphs. *Random Structures and Algorithms* **10** (1997) 5–42.
11. Füredi, Z. and Komlós, J.: The eigenvalues of random symmetric matrices. *Combinatorica* **1** (1981) 233–241.

12. Goemans, M.X. and Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. of the ACM* **42** (1995) 1115–1145.
13. Goemans, M.X. and Williamson, D.P.: Approximation algorithms for MAX-3-CUT and other problems via complex semidefinite programming. *Proc. 33rd ACM Symp. Theory of Computing (STOC)* 2001, 443–452.
14. Goerdts, A. and Jurdzinski, A.: Some results on random unsatisfiable k -Sat instances and approximation algorithms applied to random structures. *Proc. Mathematical Foundations of Computer Science (MFCS)* 2002, 280–291.
15. Grötschel, M., Lovász, L., Schrijver, A.: *Geometric algorithms and combinatorial optimization*. Springer 1988.
16. Hastad, J.: Some optimal inapproximability results. *Proc. 29th ACM Symp. Theory of Computing (STOC)* 1997, 1–10.
17. Janson, S., Luczak, T., Ruciński, A.: *Random Graphs*, Wiley 2000.
18. Kalapala, V. and Moore, C.: MAX-CUT on sparse random graphs. University of New Mexico Technical Report TR-CS-2002-24.
19. Karp, R.: The probabilistic analysis of combinatorial optimization algorithms. *Proc. Intl. Congress of Mathematicians* (1984) 1601–1609.
20. Kinnison, Robert: *Applied Extreme Value Statistics*. Battele Press. (1985) 54–57.
21. Krivelevich, M., Vu, V.H.: Approximating the independence number and the chromatic number in expected polynomial time. *J. Combin. Opt.* **6** (2002) 143–155.
22. Krivelevich, M.: Coloring random graphs – an algorithmic perspective. *Proc. 2nd Coll. on Math. and Comp. Sci.*, B. Chauvin et al. eds., Birkhäuser (2002) 175–195.
23. Lawler, E.L.: A note on the complexity of the chromatic number problem. *Information Processing Letters* **5** (1976) 66–67.
24. Levin, L.: Average case complete problems. *Proc. 16th STOC* (1984) 465
25. McKay, B. D. and Wormald, N. C.: Asymptotic enumeration by degree sequence of graphs with degrees $o(n^{1/2})$. *Combinatorica* **11**(4) (1991) 369–382.
26. Poljak, S. and Rendl, F.: Nonpolyhedral relaxations of graph-bisection problems. *SIAM J. Optimization* **5** (1995) 467–487.

Approximation Algorithm for Directed Telephone Multicast Problem

Michael Elkin^{1*} and Guy Kortsarz²

¹ School of Mathematics, Institute for Advanced Study, Princeton, NJ, USA, 08540.

elkin@ias.edu

² Computer Science Department, Rutgers University, Camden, NY, USA.

guyk@crab.rutgers.edu

Abstract. Consider a network of processors modeled by an n -vertex *directed* graph $G = (V, E)$. Assume that the communication in the network is synchronous, i.e., occurs in discrete “rounds”, and in every round every processor is allowed to pick one of its neighbors, and to send him a message. The *telephone k -multicast* problem requires to compute a schedule with minimal number of rounds that delivers a message from a given single processor, that generates the message, to all the processors of a given set $\mathcal{T} \subseteq V$, $|\mathcal{T}| = k$. The processors of $V \setminus \mathcal{T}$ may be left uninformed.

The telephone multicast is a basic primitive in distributed computing and computer communication theory. In this paper we devise an algorithm that constructs a schedule with $O(\max\{\log k, \frac{\log n}{\log k}\} \cdot br^* + k^{1/2})$ rounds for the *directed k -multicast* problem, where br^* is the value of the optimum solution. This significantly improves the previously best-known approximation ratio of $O(k^{1/3} \cdot \log n \cdot br^* + k^{2/3})$ due to [EK03].

We show that our algorithm for the directed multicast problem can be used to derive an algorithm with a similar ratio for the *directed minimum poise Steiner arborescence* problem, that is, the problem of constructing an arborescence that spans a collection $\mathcal{T}$ of terminals, minimizing the sum of height of the arborescence plus maximum out-degree in the arborescence.

1 Introduction

Consider a network of processors modeled by an n -vertex graph $G = (V, E)$. Assume that the communication in the network is synchronous, i.e., occurs in discrete “rounds”, and in every round every processor is allowed to pick one of its neighbors, and to send him a message. The *telephone k -multicast* problem requires to compute a schedule with minimal number of rounds that delivers a message from a given single processor, that generates the message, to all the

* This material is based upon work supported by the National Science Foundation under agreement No. DMS-9729992. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

processors of a given set $\mathcal{T} \subseteq V$, $|\mathcal{T}| = k$. The processors of $V \setminus \mathcal{T}$ may be left uninformed. The case $\mathcal{T} = V$ is called *broadcast* problem.

The telephone multicast and broadcast are basic primitives in distributed computing and computer communication theory, and are used as building blocks for various more complicated tasks in these areas (cf. [HHL88]). The optimization variants of the multicast and broadcast primitives were intensively studied during the last decade. Most of the research focused on undirected graphs [BGNS98, KP95, R94, S00, F01]. Several approximation algorithms with a polylogarithmic ratio were suggested for the undirected minimum time multicast problem [BGNS98, R94, EK03].

A related notion is the *poise* of a directed graph [R94]. Let G be a graph and T a spanning arborescence in the graph. The poise of an arborescence is defined to be $\maxdeg(T) + h(T)$ with $\maxdeg$ being the maximum out-degree in the arborescence and $h(t)$ the height (maximum number of edges in a root to leaf path) of the arborescence. The poise of a graph G is the minimal poise of some spanning tree T of G .

1.1 Preliminaries

The input in the minimum time directed telephone multicast problem consists of a directed graph $G(V, E)$, a source vertex s and a collection $\mathcal{T} \subseteq V$ of terminals. Throughout the run of the broadcast protocol, the vertices V are split to the *informed* subset I and the *uninformed* vertices U . We denote $UT = U \cap \mathcal{T}$. At the start time of the broadcast protocol $I = \{s\}$, $U \leftarrow V \setminus \{s\}$ and $UT \leftarrow \mathcal{T}$.

A *round* is a *directed matching* of a subset $X \subset I$ to a subset $Y \subseteq U$. Thus, the matching edges are directed from X to Y . After the round, the Y vertices become informed, namely, $I \leftarrow I \cup Y$ and $U \leftarrow U \setminus Y$. More importantly, $UT \leftarrow UT \setminus Y$. Thus, a schedule is an ordered tuple of rounds. The goal is to use the fewest possible number of rounds and inform all of $\mathcal{T}$. Namely, at the end of the protocol we must have $UT = \emptyset$.

Throughout the paper we denote $|\mathcal{T}| = k$, and the optimum value for the instance at hand is denoted by br^* . As the number of informed vertices can at most double at every round, $br^* \geq \lceil \log_2 k \rceil$. In addition, since at every round we can inform at least one additional vertex, we may assume that $br^* \leq n - 1$. We assume that the value of br^* is known and can be used by the algorithm. Indeed, the correct br^* value can be found by binary search between the maximum possible value of $n - 1$ and the minimum possible value of $\lceil \log_2 k \rceil$.

Given a subgraph G' , the distance (minimum number of edges in a shortest path) between u and v is denoted by $dist(u, v, G')$. Given an arborescence T , its height (the largest distance from a root to a leaf) is denoted by $h(T)$. The set of neighbors of u (vertices at distance 1 from u) is denoted by $N(u)$.

The graph induced by a set of nodes U is denoted by $G(U)$.

We denote by $deg(v)$ the out-degree of a vertex v . The out-degree of v in a subgraph G' is denoted by $deg(v, G')$.

For a vertex v and positive integer $\ell = 1, 2, \dots$, the ℓ -out-neighborhood in G is the vertex set $\{u \mid d_G(v, u) \leq \ell\}$.

A leaf in a rooted arborescence is a vertex that has no children.

An (r, t) approximation algorithm for a minimization problem P is an algorithm that given an instance α of P returns a solution of value at most $r \cdot \text{opt}(\alpha) + t$, where $\text{opt}(\alpha)$ is the value of the optimal solution of the problem P on the instance α .

The busy schedule procedure. One of the tools that are used in our multicast procedure is the well known busy schedule. In the busy schedule, an informed node u considers its set of neighbors. If there exists an uninformed neighbor of u , then u chooses an *arbitrary* uninformed neighbor and sends it the message. This schedule is called *non lazy* (the terminology is due to [BGNS98]).

Thus, the busy schedule is the simple greedy strategy that essentially makes sure that whenever possible no informed vertex is “idle”.

Finally, we use the following lemma due to [EK02].

Lemma 1. [EK02] *Let Q be an arborescence rooted at s with leaf set L and depth h . Assume that s knows the message. Then the busy schedule is a multicast scheme to all the vertices in Q in no more than $h + |L|$ rounds.*

1.2 Previous Research

The optimization variants of the broadcast and multicast primitives were intensively studied during the last decade. See, for example, [BGNS98,BK94,KP95,M93,R94,S00,F01,EK02,EK03]. Table 1 summarizes the previous research.

Table 1. Known positive and negative approximability results for the telephone broadcast and multicast problems. The positive results appear in columns marked “u.b.” (stands for “upper bounds”), and the negative results appear in columns marked “l.b.” (stands for “lower bounds”).

	Broadcast		Multicast	
	<i>u.b.</i>	<i>l.b.</i>	<i>u.b</i>	<i>l.b</i>
Directed	$\log n$ [EK02]	$\sqrt{\log n}$ [EK02]	$(k^{1/3} \cdot \log n, k^{2/3})$ [EK03]	$\log k$ [F01]
Undirected	$\log n / \log \log n$ [EK03]	3 [EK02]	$O(\log k / \log \log k)$ [EK02]	3 [EK02]

1.3 Our Results

For the directed k -multicast problem, currently the best-known approximation is $(k^{1/3} \cdot \log n, k^{2/3})$ due to [EK03]. We improve upon this and devise an algorithm that given input with optimal schedule of length br^* constructs a schedule of length $O(br^* \cdot \max\{\log k, \frac{\log n}{\log k}\} + k^{1/2})$ (that is, the approximation is

$(\max\{\log k, \frac{\log n}{\log k}\}, k^{1/2})$). For the special case of $br^* = \Omega(\sqrt{k})$ (for example, graphs with high diameter) the ratio is $O(\log n)$. It is easy to show that for $k = \Omega(n^\epsilon)$ with $\epsilon > 0$ a (possibly small) constant and $br^* = \Omega(\sqrt{k})$, the achieved ratio (that is, $O(\log n)$) is the best possible up to a constant factor, unless $P = NP$. The running time of our algorithm is $\tilde{O}(n^3)$ (the notation $\tilde{O}$ ignores polylogarithmic factors).

In the more general *postal model* (introduced in [BGNS98]) each vertex v has a delay number $0 \leq \mu(v) \leq 1$. The vertex sending a message is “busy” only at the first μ time units starting from the initial sending time. After μ time units, v is free to send another message. In addition, every edge e has a delay number ℓ_e representing the time required to send the message over e . We extend our algorithm to this more general problem, and show that this problem admits the same approximation ratio.

We also derive the first non-trivial approximation algorithm for the minimum poise Steiner arborescence problem. The approximation ratio that we achieve for this problem is $(O(\log n), O(\sqrt{k}))$. This results can be considered a bicriteria approximation result [HRR+]; Our algorithm accepts as input a graph that has a spanning arborescence of height h^* and maximum out-degree d^* , and produces a directed Steiner arborescence of height $O(h^*)$ and maximum out-degree $O(\log n \cdot d^*) + O(\sqrt{k})$.

Comparing our algorithm with the algorithm of [EK03] (the latter algorithm achieves approximation ratio of $(O(k^{1/3} \log n), k^{2/3})$), our algorithm is significantly simpler. Specifically, the algorithm of [EK03] makes a heavy use of broadcasting through *jungles*. The jungle (introduced in [EK03]) is a collection of not necessarily vertex-disjoint trees, that, nevertheless, possess some other useful properties. Broadcast through jungles was shown in [EK03] to be a useful technique that enables to achieve the only known today *sublogarithmic* approximation ratio for *undirected* k -multicast problem. This technique was applied in [EK03] to the *directed* k -multicast problem, achieving the first (and the only known prior to the current paper) *sublinear* approximation ratio for this problem. In this paper we show that in the case of *directed* k -multicast problem, the jungles can be replaced by ordinary forests, achieving a significantly better approximation ratio, and simplifying the algorithm.

1.4 Overview of the Algorithm

The multicast schedule that we define can be roughly divided into two parts. In the first part of the schedule, the goal is to construct a partition of the vertex set V into two disjoint subsets I and U . The set I will be referred as the set of *informed* vertices, and the set U will be referred as the set of *uninformed* ones. This partition has to satisfy certain properties. First, there should be a polynomially-computable schedule for informing the vertices of the set I . Second, no uninformed vertex (that is, a vertex that belongs to U) is allowed to have in its vicinity “many” uninformed terminals (vertices of the set $UT = U \cap \mathcal{T}$). These vicinities are defined with respect to the graph $G(U)$ induced by the vertex set

U . These properties will be captured formally by the notion of $\sqrt{k}$ degree-height partition, that will be defined later on.

Roughly speaking, given a partition that satisfies these properties, the schedule that informs UT can be efficiently constructed. In other words, the vertices of the set UT of yet uninformed terminals can be informed within $O(\max\{\log k, \log_k n\}) \cdot br^*$ additional rounds.

Note that the number of rounds used in this part of the schedule is essentially the smallest possible (unless $P = NP$) as indicated by the lower bound of [F01]. However, to construct a complete schedule we still need to inform the vertices of the set U . In our schedule this task requires $2\sqrt{k}$ rounds. Moreover, after the first part of our schedule, the number of terminals at distance br^* from $u \in U$ can be as high as $\sqrt{k}$. It follows that any improvement in constructing the subschedule for informing the vertices of the set U , would imply an analogous improvement of the approximation guarantee of the entire algorithm.

In Section 2 we describe the algorithm for constructing the partition with the desired properties. This is done by a greedy partitioning technique that uses minimum distance arborescences.

We then explain how to complete the multicast given the partition (I, U) . For this end, we reduce this task to certain generalization of the set-cover problem, that we call *multiple set-cover* problem. We present a logarithmic approximation algorithm for this problem that uses linear programming. We believe that both the multiple set-cover problem and our approximation algorithm for it are of independent interest.

The second part of the algorithm uses the algorithm for the multiple set-cover problem and has two stages: in the first stage, it informs a subset $D \subseteq U$ of vertices, that satisfies that every vertex $u \in UT \setminus D$ has a nearby vertex $w(u) \in D$. In the second stage, the shortest path arborescences are used to complete the task of informing $UT \setminus D$.

In this extended abstract we outline the proof of a slightly weaker approximation guarantee, specifically, $(O(\log n), O(\sqrt{k}))$. Most proofs are omitted.

2 Constructing the Partition and Informing the Vertices of U

2.1 Computing a $\sqrt{k}$ Partition

We start with the definition of the $\sqrt{k}$ degree-height partition.

Definition 1. A $\sqrt{k}$ degree-height partition (henceforth, $\sqrt{k}$ partition) is a partition of V into two disjoint sets $V = I \cup U$, $I \cap U = \emptyset$, $S \in I$, such that in the graph $G(U)$ induced by U no vertex $u \in U$ can reach via a directed path of length at most br^* more than $\sqrt{k}$ terminals in $U \cap \mathcal{T} = UT$.

Again, intuitively, the vertices of the set I are informed. It remains to inform the terminals of UT .

Intuition: To understand why a $\sqrt{k}$ partition is useful, assume that we are given a $\sqrt{k}$ partition and that the remaining goal is to inform UT .

In Section 3.2 it will be shown that a subset D of U that has the property that each terminal $u \in UT$ has a nearby vertex $v \in D$ (the distance is measured in $G(U)$) can be efficiently informed.

Then, we define a collection of arborescences and use it to send the message to the uninformed terminals of UT via this collection. For each vertex $u \in D$, let T_u denote the arborescence rooted in u . The sets of leaves of each multicast arborescence T_u are contained in the set $UT \setminus D$. These properties are used to guarantee that the trees T_u are relatively shallow, and that each of them has at most $\sqrt{k}$ leaves.

The algorithm: The following procedure starts with $U = V \setminus \{s\}$ and $I = \{s\}$ and informs more vertices (changing U and I) so as to reach a $\sqrt{k}$ degree-height partition (U, I) with all I informed.

We say that $u \in U$ is $\sqrt{k}$ -good if its br^* -out-neighborhood in $G(U)$ contains at most $\sqrt{k}$ terminals.

Input: A graph $G = (V, E)$, a set $\mathcal{T}$ of terminals, and a source $s \in V$.

Output: A $\sqrt{k}$ partition (I, U) of V .

Procedure *Comp – Par*

1. $I \leftarrow \{s\}; \quad U \leftarrow V \setminus \{s\};$
2. $Roots \leftarrow \emptyset; \quad /* \text{The roots are later used to inform the new vertices of } I. *$
*/
3. **While** $G(U)$ admits a $\sqrt{k}$ -good vertex u **Do**:
 - a) Let $Cl(u)$ be the $\lfloor \sqrt{k} \rfloor$ uninformed terminals closest to u , in $G(U)$.
 - b) Let T_u be the shortest path arborescence leading from u to $Cl(u)$ in $G(u)$.
 - c) $I \leftarrow I \cup V(T_u), \quad U \leftarrow U \setminus V(T_u), \quad Roots \leftarrow Roots \cup \{u\}.$
4. Output $(I, U, Roots)$

Properties of the algorithm: The following claim is derived directly from the algorithm.

Claim. (1) The pair (I, U) output by Procedure *Comp – Par* is a $\sqrt{k}$ partition. (2) The set $Roots$ has cardinality $|Roots| \leq \sqrt{k} + 2$. (3) It is possible to use $2br^* + 2\sqrt{k} + 2$ rounds or less in order to transform $U = V \setminus \{s\}, \quad I = \{s\}$ into a new partition (I, U) so that I is the set of informed vertices and (U, I) is a $\sqrt{k}$ degree-height partition.

3 From a $\sqrt{k}$ Degree-Height Partition to a Complete Schedule

3.1 An Algorithmic Tool: The Multiple Set-Cover Problem

Consider the following problem called the multiple set-cover problem. Let $G(V_1, V_2, E)$ be a bipartite graph. Recall that a set $S \subseteq V_1$ is called a *set-cover* of

V_2 if $N(S, G) = V_2$, namely, every $v_2 \in V_2$ has at least one neighbor in S . The minimum set-cover problem is to find a set-cover $S \subset V_1$ of V_2 with minimum cardinality.

The multiple set-cover problem is defined as follows.

Input: A bipartite graph $G(V_1, V_2, E)$ with $|V_1| + |V_2| = n$. The set V_1 is partitioned into a disjoint union of sets $V_1 = \bigcup_{j=1}^d A_j$.

Output: A set cover $S \subset V_1$ of V_2 .

Definition 2. For a cover S , let the value of S be $val(S) = \max\{|S \cap A_i|\}_{i=1}^d$.

Optimization goal: Minimize $val(S)$.

Observe that the usual set-cover problem is a special case of the multiple set-cover problem with $d = 1$. The case of general d appears to be somewhat more complex. In particular, it is not clear to the authors whether a natural greedy algorithm achieves a logarithmic approximation ratio. However, it can be shown that randomized rounding [RT87] can be employed to achieve this approximation ratio. Note that this is tight up to a constant factor in view of the logarithmic lower bound of [RS97] on the approximation threshold of the set-cover problem.

Theorem 1. *There exists a deterministic polynomial-time $O(\log n)$ -approximation algorithm for the multiple set-cover problem.*

3.2 An Application of the Multiple Set Cover Problem for a Multicasting Task

Next, we present a reduction from the directed multicast problem to the multiple set-cover problem. This reduction, along with our approximation algorithm for the multiple set-cover problem, described in Section 3.1, is used to obtain the approximation algorithm for the directed multicast problem.

The problem of informing a distance- br^* dominating set D : A dominating set D in a directed graph G is a set D so that $D \cup N(D) = V$, that is, a vertex that is not in D must have an incoming edge from a vertex of D . A distance- br^* dominating set in G is a set D that satisfies that for every $v \in V$ there is a vertex $u \in D$ such that $dist(u, v) \leq br^*$. Finally, D is a distance- br^* dominating set with respect to a subset W if the above condition holds for every vertex in W (that is, the vertices of W are of distance at most br^* from D , but the vertices from $V \setminus W$ may be at larger than br^* distance from D).

Let I, U be a disjoint partition of V such that the source s belongs to the set I . Suppose that all the vertices of the set I are already informed. It remains, therefore, to inform the vertices of $UT = U \cap \mathcal{T}$. We relax this problem as follows. Let $P_{I,U}$ be the problem of informing some distance- br^* dominating set D of UT using minimum number of rounds. This is the first phase in the second part of the algorithm. Informing of the vertices of $UT \setminus D$ once the vertices of D are informed is the second phase.

We show how to use the solution for the multiple set-cover problem to approximate the $P_{I,U}$ problem.

Consider the following bipartite graph $\mathcal{B}_{I,U}(\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$. Let

$$\mathcal{V}_1 = \{x_{(v,u)} \mid v \in I, u \in U, (v,u) \in E\}.$$

Intuitively, the set $\mathcal{V}_1$ is the set of edges (v,u) of the original graph G , with v in I , and u in U . Observe that, in a sense, each vertex $u \in U$ is “duplicated” many times. That is, if the vertex u has d incoming edges, it will appear in d different vertices $x_{v,u}$.

Let $A_v = \{x_{(v,u)} \mid u \in N_G(v)\}$. The disjoint partition of $\mathcal{V}_1$ that is provided as input for the multiple set-cover problem is $\mathcal{V}_1 = \bigcup_{v \in V} A_v$.

The set $\mathcal{V}_2$ is set to $\mathcal{V}_2 \leftarrow UT$ (the set of terminals that are still uninformed).

We now define the edge set $\mathcal{E}$ of the graph $\mathcal{B}_{I,U}(\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$. There is an edge between a vertex $x_{(v,u)}$ to a vertex $w \in UT$ if there is a directed path of length at most br^* from u to w in $G(U)$. See Figure 1 for an illustration.

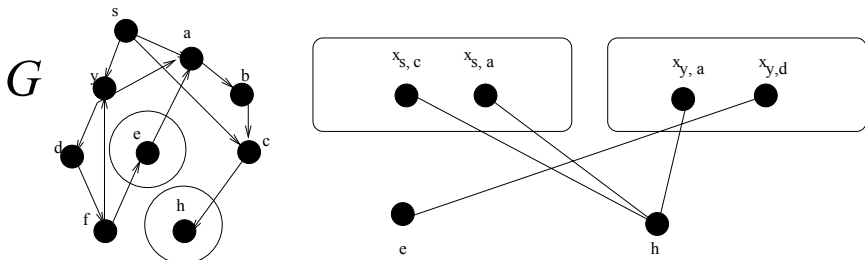


Fig. 1. An example of the application of the multiple set-cover problem. The terminals e and h are depicted by cycles around the vertices. It is easy to see that for the graph G , $br^* = 4$. The disjoint partition of the vertex set V into $I \cup U$ that is obtained after one round is $I = \{s, y\}$ and $U = V \setminus I$. This partition induces the bipartite graph $\mathcal{B}_{I,U}$ that is depicted on the right side. For example, the vertex $x_{s,a}$ is connected to h as the distance between a and h in the graph $G(U)$ induced by U is $3 \leq br^*$. In contrast, the respective distance of h from d is greater than br^* . Consequently, there is no edge between $x_{y,d}$ and h in the bipartite graph $\mathcal{B}_{I,U}$.

Lemma 2. *The multiple set-cover instance $\mathcal{B}$ admits a solution $S^* \subseteq \mathcal{V}_1$ such that $\text{val}(S^*) \leq br^*$.*

Now, assume that we compute the instance $\mathcal{B}_{I,U}$ of the multiple set-cover instance of I and U as described above. By Theorem 1 there exists a polynomial-time $O(\log n)$ -approximation algorithm for the multiple set-cover problem. Hence, the algorithm returns a solution D for the multiple set-cover problem with value $O(br^* \log n)$.

If $x_{v,u}$ is chosen into the cover, we say that the vertex u is assigned to the vertex v . Observe that a vertex $u \in U$ can be assigned many times in the solution. In other words, it may happen that several variables $x_{v_1,u}, x_{v_2,u}, \dots$ are rounded

to 1. In this case it is enough to choose an arbitrary variable $x_{v_i, u}$, and set it to 1, while setting to zero all the others. Thus, each vertex u is assigned to a unique vertex v_i .

Lemma 3. $\max_{v \in V_1} |D \cap A_v| = O(\log n) \cdot br^*$.

Lemma 4. *There is a polynomially computable schedule using $O(\log n \cdot br^*)$ rounds for $P_{I, U}$.*

3.3 Informing the Set UT

We start with a $\sqrt{k}$ degree-height partition (I, U) and elaborate on the following task: how to send the message from I to UT using as small number of rounds as possible. The algorithm has two phases. The first phase applies the solution for $P_{I, U}$. The second phase defines a collection of arborescences in the graph $(G(U)$ induced by U and uses this collection in order to inform the terminals of UT .

For simplicity, we are first going to describe a solution for phase 2 that uses a collection of not necessarily disjoint arborescences. Later, we explain a fix that allows to maintain the useful properties of the defined arborescences, while reducing the collection of arborescences into a vertex-disjoint collection.

1. Compute the $P_{I, U}$ instance of U and I and compute its approximate solution D .
2. **Phase 1:** Every v such that $D \cap A_v \neq \emptyset$ sends the message in arbitrary order to all the vertices in $\{u \mid x_{v, u} \in D \cap A_v\}$.
3. **Assignment Phase:** The assignment is performed by the following procedure called Procedure *Assign*:

Procedure *Assign*

- a) Initialize all the vertices of $UT \setminus D$ as “unassigned”.
- b) **While** there is an unassigned vertex in $UT \setminus D$ **do**:
 - i. Let u be a vertex in D so that there exists an unassigned vertex w , $w \in UT \setminus D$ so that $\text{dist}(u, w, G(U)) \leq br^*$.
 - ii. **For all** $z \in UT \setminus D$ so that $\text{dist}(u, z, G(U)) \leq br^*$, set $\psi(z) = u$ and declare that z is assigned.
4. **Phase 2:** Let u be a vertex that has been assigned to at least one vertex of $UT \setminus D$. Let T_u be the shortest path arborescence leading from $u \in D$ to $W_u = \{w \mid u = \psi(w)\}$ in the graph induced by U . Let $\mathcal{J}$ be the collection of all arborescences for all the different vertices u .
 /* Observe that we are using a non-disjoint collection of arborescences; The arborescences T_u are all computed over the same graph $G(U)$ and are not edge or vertex disjoint. */
 Use the busy schedule to broadcast over $\mathcal{J}$.

The following claim applies.

Claim. Phase 1 requires $O(\log n) \cdot br^*$ rounds.

We now consider some of the properties of the *jungle* $\mathcal{J}$

Claim. The maximum height of a tree in $\mathcal{J}$ is at most br^* .

In addition, by the properties of the $\sqrt{k}$ partition, the number of leaves in T_u is at most $\sqrt{k}$ (as otherwise more than $\sqrt{k}$ of the UT vertices are reachable from u via a path of length at most br^*).

As the arborescences are not vertex disjoint, the broadcasting task on one arborescence may conflict with the broadcasting task on other arborescences (since a single vertex is not allowed to send the message to two neighbors in the same round). It, therefore, follows that in the way Phase 2 is described now, the busy schedule may use too many rounds to inform the vertices of $\bigcup T_u$. The following change is used to form a vertex-disjoint collection of arborescences out of the jungle J .

Define a level on every vertex in every arborescence in $\mathcal{J}$. The root u of the arborescence T_u is at level 0 and the level of a non-root in T_u is one larger than the level of its parent. For simplicity, we say that every root u has a parent at level -1 . For every v appearing in several arborescences, let $par^*(v)$ be the parent of v in one of the arborescences, so that the level of $par^*(v)$ is minimum among the level of all the other parents of v in all other arborescences. Define a collection $\mathcal{F}$ of arborescences by taking the directed graph induced by the edges $(par^*(v), v)$ (the edges going from $par^*(v)$ into v). Note that some vertices of $U \setminus \mathcal{T}$ may become leaves (namely, have no children). Such vertices are discarded. Moreover, we discard every vertex that has no terminals in its sub-arborescence in $\mathcal{F}$.

Claim. The above definition changes the collection of arborescences $\mathcal{J}$ into a forest $\mathcal{F} = \{T'_w\}$, namely, into a collection of vertex disjoint arborescences.

Claim. (1) For every arborescence T' in $\mathcal{F}$, there exists a tree $\tau \in \mathcal{J}$ such that $root(\tau) = root(T')$. (2) The height of every arborescence in $\mathcal{F}$ is at most br^* . (3) The number of leaves in every T'_w in $\mathcal{F}$ is at most br^* .

Claim. There exists a procedure that accepts as input a $\sqrt{k}$ degree-height partition (U, I) and outputs a schedule with $O(\log n \cdot br^*) + 2\sqrt{k}$ rounds that informs UT . Furthermore, this procedure requires polynomial time.

4 The Multicast Algorithm and Its Analysis

In this section the pieces are put together and a schedule with $O(\log n \cdot br^*) + 2\sqrt{k}$ rounds is derived.

4.1 The Algorithm

We start with a formal description of Procedure *Multicast*.

Input: A graph $G = (V, E)$, a set $\mathcal{T}$ of terminals and a source $s \in V$.

Output: A schedule with $O(\log n \cdot br^*) + 2\sqrt{k}$ rounds.

Procedure *Multicast*

1. Invoke Procedure *Comp – Par* to derive $(I, U, Roots)$.
2. Use a shortest path arborescence T leading from s to $Roots$. Invoke the busy schedule to send the message from s to $Roots$.

3. For $u \in \text{Roots}$, let T_u be the arborescence as defined in Line 3b in Procedure *Comp – Par*. Every $u \in \text{Roots}$ broadcasts over T_u (in parallel) using the busy schedule.
4. Use the method described in Section 3.2 to inform a distance- br^* dominating set D of UT .
5. Use the forest $\mathcal{F}$ described in Section 3.3 to inform UT using the busy schedule over $\mathcal{F}$.

Combining Claim 2.1, Claim 3.3 and Claim 3.3, we get:

Theorem 2. *The directed telephone multicast problem admits an approximation ratio of $(O(\log n), O(\sqrt{k}))$.*

References

- [AS92] N. Alon and J. H. Spencer, *The Probabilistic Method*, Wiley, 1992.
- [BGNS98] A. Bar-noy, S. Guha, J. Naor and B. Schieber. Multicasting in Heterogeneous Networks, In *Proc. of 30th ACM Annual Symp. on Theory of Computing* 1998.
- [BK94] A. Bar-Noy and S. Kipnis, “Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems,” *Mathematical System Theory*, Vol. 27, pp. 431–452, 1994.
- [C52] H. Chernoff, A measure of asymptotic efficiency for tests of an hypothesis based on the sum of observations, *Annals of Mathematical Statistics*, 23:493–509, 1952.
- [EK02] M. Elkin and G. Kortsarz, A combinatorial logarithmic approximation algorithm for the directed telephone broadcast problem, In *Proc. of 34th ACM Annual Symp. on Theory of Computing*, , pp. 438–447, 2002.
- [EK03] M. Elkin and G. Kortsarz, A sublogarithmic approximation algorithm for the undirected telephone broadcast problem: a path out of a jungle. In *Proc. of 14th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 76–85, 2003.
- [F01] P. Fraigniaud Approximation Algorithms for Minimum-Time Broadcast under the Vertex-Disjoint Paths Mode, In *9th Annual European Symposium on Algorithms (ESA '01)*, LNCS Vol. 2161, pp. 440–451, 2001.
- [FR90] M. Furer and B. Raghavachari. An NC approximation algorithm for the minimum degree spanning tree problem. In *Proc. of the 28th Annual Allerton Conf. on Communication, Control and Computing*, pp. 274–281, 1990.
- [HHL88] S. Hedetniemi, S. Hedetniemi, and A. Liestman. A survey of broadcasting and gossiping in communication networks. *Networks*, 18: 319–349, 1988.
- [HRR+] H. B. Hunt, M. V. Marathe, R. Sundaram, R. Ravi, S. S. Ravi and D. J. Rosenkrantz. Bicriteria network design problems, *Journal of Algorithms*, Vol. 28, No. 1, 142–171 (1998).
- [K-79] L. G. Khachiyan. A Polynomial Algorithm in Linear Programming, *Doklady Akademii Nauk SSSR* 244, pp. 1093–1096, 1979.
- [K-84] N. Karmarkar, A new polynomial-time algorithm for linear programming, *Combinatorica*, vol. 4 (1984), pp. 373–396.
- [KP95] G. Kortsarz and D. Peleg. Approximation algorithms for minimum time broadcast, *SIAM journal on discrete methods*, vol. 8, pp. 401–427, 1995.

- [LST-90] L. K. Lenstra and D. Shmoys and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines", *Math. Programming*, 46, 259–271. (1990).
- [KR01] R. Krishnan and B. Raghavachari. The Directed Minimum-Degree Spanning Tree Problem. *FSTTCS 2001*, 232–243.
- [M93] M. Middendorf. Minimum Broadcast Time is NP-complete for 3-regular planar graphs and deadline 2. *Inf. Process. Lett.* 46 (1993) 281–287.
- [MR95] R. Motwani and P. Raghavan, Randomized Algorithms, *Cambridge University Press*, 1995.
- [PST95] S. A. Plotkin and D. B. Shmoys and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research* 20, 1995, 257–301.
- [R88] P. Raghavan, probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of computer and system sciences*, 37:130–143,1988.
- [R94] R. Ravi, Rapid rumor ramification: Approximating the minimum broadcast time. In *Proc. of the IEEE Symp. on Foundations of Computer Science (FOCS '94)*, pp. 202–213, 1994.
- [RS97] R. Raz, S. Safra. A Sub-Constant Error-Probability Low-Degree Test, and a Sub-Constant Error-Probability PCP Characterization of NP, in *Proc. of the 29th Symp. on Theory of Comp.*, pp. 475–484, 1997.
- [RT87] P. Raghavan and C. Thompson, Randomized Rounding, *Combinatorica*, vol. 7, pp. 365–374, 1987.
- [S00] C. Schindelhauer, On the Inapproximability of Broadcasting Time, In *The 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'00)*, 2000.

Mixin Modules and Computational Effects

Davide Ancona, Sonia Fagorzi, Eugenio Moggi, and Elena Zucca*

DISI, Univ. of Genova, v. Dodecaneso 35, 16146 Genova, Italy
{davide,fagorzi,moggi,zucca}@disi.unige.it

Abstract. We define a calculus for investigating the interactions between mixin modules and computational effects, by combining the purely functional mixin calculus *CMS* with a monadic metalanguage supporting the two separate notions of *simplification* (local rewrite rules) and *computation* (global evaluation able to modify the store). This distinction is important for smoothly integrating the *CMS* rules (which are all local) with the rules dealing with the imperative features.

In our calculus mixins can contain mutually recursive *computational* components which are explicitly computed by means of a new mixin operator whose semantics is defined in terms of a Haskell-like recursive monadic binding. Since we mainly focus on the operational aspects, we adopt a simple type system like that for Haskell, that does not detect dynamic errors related to bad recursive declarations involving effects. The calculus serves as a formal basis for defining the semantics of imperative programming languages supporting first class mixins while preserving the *CMS* equational reasoning.

1 Introduction

Mixin modules (or simply mixins) are modules supporting *parameterization*, *cross-module recursion* and *overriding* with *late binding*; these three features altogether make mixin module systems a valuable tool for promoting software reuse and incremental programming [AZ02]. As a consequence, there have been several proposals for extending existing languages with mixins; however, even though there already exist some prototype implementations of such extensions (see, e.g., [FF98a,FF98b,HL02]), there are still several problems to be solved in order to fully and smoothly integrate mixins with all the other features of a real language. For instance, in the presence of store manipulation primitives, expressions inside mixins can have side-effects, but this possibility raises some semantic issues: (1) because of side-effects, the *evaluation* order of components inside a mixin must be *deterministic*, while still retaining cross-module-recursion; (2) *when* computations inside a mixin must be evaluated and *how many times*?

Unfortunately, all formalizations defined so far [AZ99,AZ02,MT00,WV00] do not consider these issues, since they only model mixins in purely functional settings.

* Supported by MIUR project NAPOLI, EU project DART IST-2001-33477 and thematic network APPSEM II IST-2001-38957

In this paper we propose a monadic mixin calculus, called CMS_{do} , for studying the interaction between the notions of mixin and store. More precisely, this calculus should serve as a formal basis both for defining the semantics of imperative programming languages supporting mixins and for allowing equational reasoning.

Our approach consists in combining the purely functional mixin calculus CMS [AZ99,AZ02] with a monadic metalanguage [MF03] equipped with a Haskell-like recursive monadic binding [EL00,EL02] and supporting the two separate notions of *simplification* and *computation*, the former corresponding to local rewriting with no side-effects, the latter to global evaluation steps able to modify the store. This distinction is important for smoothly integrating the CMS rules (which are all local) with the rules dealing with the imperative features; furthermore, since simplification is a congruence, all CMS equations (except those related to selection) hold in CMS_{do} .

In CMS_{do} a mixin can contain, besides the usual CMS definitions, also *computational* definitions of the form $x \leftarrow e$, where e has monadic type. The (simplification) rules for the standard operators on mixins coincide with those given for CMS . However, before selecting components from a mixin, this must be transformed into a record. The transformation of a mixin (without deferred components) into a record is triggered by the `doall` primitive, and consists in (1) evaluating computational definitions $x_i \leftarrow e_i$ in the order they are declared; (2) binding the value returned by e_i to x_i immediately, to make it available to the subsequent computations e_j with $j > i$.

Mutual recursion has the following informal semantics: if $i \leq j$, then e_i can depend on the variable x_j , provided that the computation e_i can be successfully performed without knowing the value of e_j (which is bound to x_j only later). Formally, the semantics of `doall` is expressed in terms of a recursive monadic binding, similar to that defined in [EL00,EL02], and a standard recursive let-binding.

Since the emphasis of the paper is on the operational aspects, we adopt a simple type system like that for Haskell, that does not detect dynamic errors related to bad recursive declarations; for instance, `doall([; x ← set(y,1), y ← new(0)])` is a well-typed term which evaluates into a dynamic error. However, more refined type systems based on dependencies analysis [Bou02,HL02] could be considered for CMS_{do} in order to avoid this kind of dynamic errors.

The rest of the paper is organized as follows. In Section 2 we illustrate the main features of the original CMS calculus and introduce the new CMS_{do} calculus through some examples. In Section 3 we formally define the syntax of the calculus, the type system and the two relations of simplification and computation. We also prove standard technical results, including a bisimulation result (simplification does not affect computation steps) and the progress property for the combined relation. In Section 4 we discuss related work and in Section 5 we summarize the contribution of the paper and draw some further research directions.

2 An Overview of the Calculus

In this section we give an overview of the CMS_{do} calculus by means of some examples written in a more user-friendly syntax.

Like in CMS , a CMS_{do} *basic mixin module* consists of *defined* and *local* components, bound to an expression, and *deferred* components, declared but not yet defined.

Example 1. For instance,

```
M1 = mix import N2 as x,          (* deferred *)
      export N1 = e1[x,y],        (* defined *)
      local   y = e2[x,y]         (* local *)
end
```

denotes a mixin with one deferred, one defined and one local¹ component, where $e1[x,y]$ and $e2[x,y]$ denote two arbitrary expressions possibly containing the two free variables x and y . Deferred components are associated with both a component name (as $N2$) and a variable (as x); component names are used for external referencing of deferred and defined components but they are not expressions, while variables are used for accessing deferred and local components inside mixins (for further details on the separation between variables and component names see [Ler94], [HL94], [AZ02]). Local components are not visible from the outside and can be mutually recursive.

Besides this construct, CMS_{do} provides four operations on mixins: *sum*, *freeze*, *delete* (inherited from CMS) and *doall*.

Example 2. Two mixins can be combined by the *sum* operation, which performs the union of the deferred components (in the sense that components with the same name are shared), and the disjoint union of the defined and local components of the two mixins. However, while defined components must be disjoint because clashes are not allowed by the type system, the disjoint union of local components can be always performed by renaming variables.

```
M2 = mix import N1 as x,
      export N2 = e3[x,y],
      local   y = e4[x,y]
end
M3 = M1 + M2
```

Module $M3$ simplifies to

```
mix import N2 as x1, N1 as x2,
  export N1 = e1[x1,y1],
  local   y1 = e2[x1,y1],
  export N2 = e3[x2,y2],
  local   y2 = e4[x2,y2]
end
```

¹ Note that deferred, defined and local components can be declared in any order; in particular, definitions of defined and local components can be interleaved.

The sum operation supports cross-module recursion; in module M_3 , the definition of N_2 , which is needed by M_1 , is provided by M_2 , whereas the definition of N_1 , which is needed by M_2 , is provided by M_1 . However, in CMS_{do} component selection is permitted only if the module has no deferred components, therefore the defined components of M_3 cannot be selected even though the deferred components of M_3 (N_1 and N_2) are also among the defined ones.

Example 3. The *freeze* operation connects deferred and defined components having the same name inside a mixin; in other words, it is used for resolving “external names”, so that a deferred component becomes local.

For instance, in $(\text{mix import } N \text{ as } x, \text{ export } N = e_1[x,y], \text{ local } y = e_2[x,y]) ! N$ the deferred component N has been effectively bound to the corresponding defined component by freezing it, obtaining the following simplified expression:

```
mix local x = e1[x,y], export N = x, local y = e2[x,y] end
```

Example 4. The *delete* operation is used for hiding defined components:

```
(mix import N as x, export N = e1[x,y], local y = e2[x,y]) \ N
```

simplifies to

```
mix import N as x, local y = e2[x,y] end
```

So far the calculus is very similar to the pure functional calculus CMS defined in [AZ02]; its primitive operations can be used for expressing a variety of convenient constructs supporting cross-module recursion and overriding with late binding.

For instance, $M_6 = ((M_3 \setminus N_2) + \text{mix export } N_2 = e[] \text{ end}) ! N_1 ! N_2$ corresponds to declare a new mixin obtained from M_3 by overriding component N_2 ; since N_2 in M_3 is both deferred and defined, the definition of component N_2 in M_6 depends on the new definition of N_2 in M_6 rather than on that in M_3 (late binding). We refer to [AZ02] for more details on this.

In addition to the CMS operations and constructs presented above, CMS_{do} provides a new kind of mixin component called *computational*, a new mixin operation *doall* to deal with computational components, the usual primitives on the store, and the monadic constructs *mdo* (recursive *do*) and *ret* (embedding of values into computations).

Example 5. Let us consider the following mixin definition:

```
CM1 = mix local lc <= new(x-1), x = 1,
      export Inc = mdo v <= get(lc) in set(lc,v+1), Val <= get(lc)
      end
```

The local component *lc* and the defined component *Val* has been defined via *<=* (rather than *=*) and are called *computational*.

Evaluation of computational components like *lc* and *Val* can be performed only once by means of the *doall* operation (see below), provided that there are no deferred components (as in this case); furthermore, selection of the defined components of CM_1 is possible only after *lc* and *Val* have been evaluated.

Note that **Inc** is not computational, even though its associated expression contains effects, therefore the *doall* operation does not compute **Inc** (see below).

The computation **new(x-1)** returns a fresh location containing the expression **x-1**, **get(lc)** returns the expression stored at the location *l* denoted by **lc** and **set(lc,v+1)** updates the store by assigning **v+1** to *l* and returns *l*. Note that **new(e)** and **set(lc,e)** are “lazy”, in the sense that they do not evaluate the expression **e**.

Let us now consider the expression **doall(CM1)**; its evaluation returns a record containing only the defined components **Inc** and **Val**. As already explained, **Inc** is not evaluated, whereas **Val** is computed as follows. Since we require the evaluation of computational components to respect the declaration order, the expression associated with **lc** is computed before that defining **Val**; once the value of variable **lc** is computed it is made immediately available to the next computational component **Val**.

On the other hand, the component **Inc** (defined via **=**) is not computed, but its associated computation is treated as a value of monadic type that can be evaluated with the **mdo** construct. Therefore, if *l* is the location generated by the evaluation of component **lc**, then **doall(CM1)** evaluates to the record **r={Inc=mdo v<=get(l) in set(l,v+1), Val=0}**, where component **Inc** can be reevaluated several times, for instance, in the expression **mdo lc<=r.Inc in get(lc)** which increments the contents of *l* and evaluates to 1. Finally, note that the order of computational components matters, while that of non-computational components, like **x** and **Inc** in **CM1**, does not.

Example 6. Computational components can be mutually recursive like in the following mixin.

```
CM2 = mix export Loc1=l1, Loc2=l2,
      local l1<=new(l2), l2<=new(l1)
end
```

The expression **doall(CM2)** evaluates to the record **{Loc1=*l*₁, Loc2=*l*₂}** where *l*₁ and *l*₂ are two locations pointing two each other. This is possible because **new(e)** does not need to evaluate *e*.

On the other hand, evaluation of **doall(mix local x<=set(y,1), y<=new(0) end)** causes an error because of bad recursive declarations. In this case the error could be avoided by swapping **x** and **y**, but reordering computational components changes the semantics.

3 *CMS*_{do}: A Monadic Mixin Language

Before defining *CMS*_{do}, we introduce some notations and conventions.

- If *s*₁ and *s*₂ are two finite sequences, then *s*₁, *s*₂ denotes their concatenation.
- $f: A \xrightarrow{fin} B$ means that *f* is a partial function from *A* to *B* with a finite domain, written **dom(f)**. We write $\{a_i: b_i | i \in I\}$ for the partial function mapping for all *i* ∈ *I* *a*_{*i*} to *b*_{*i*} (where the *a*_{*i*} must be different, i.e. *a*_{*i*} = *a*_{*j*} implies *i* = *j*). We use the following operations on partial functions:

- $\emptyset$ is the everywhere undefined partial function;
- f and g are *compatible* when $f(x) = g(x)$ when $x \in \text{dom}(f) \cap \text{dom}(g)$.
- f_1, f_2 denotes the union of two compatible partial functions;
- $f\{a: b\}$ denotes the update of f in a ;
- $f \setminus a$ is the partial function g such that $g(x) \triangleq \begin{cases} f(x) & \text{if } x \neq a \\ \text{undefined} & \text{otherwise} \end{cases}$
- $\xrightarrow{*}$ denotes the reflexive and transitive closure of a binary relation $\longrightarrow$.
- If E is a set of terms, then $\text{FV}(e)$ is the set of free variables of e ; $\mathbf{E}_0$ is the set of $e \in \mathbf{E}$ s.t. $\text{FV}(e) = \emptyset$; $e\{\rho\}$, with ρ a finite partial function from a set of variables Var to E , denotes the parallel substitution of all variables $x \in \text{dom}(\rho)$ with $\rho(x)$ in e (modulo α -conversion).

The syntax of CMS_{do} definition is parametric in an infinite set Name of *component names* X (for records and mixins), an infinite set Var of variables x , and an infinite set $\mathbf{L}$ of *locations* l .

Terms e , recursive monadic bindings Θ and mixin bindings Δ are given by

$$\begin{array}{l}
 e \in \mathbf{E} ::= x \mid \{o\} \mid e.X \mid \text{let}(\rho; e) \mid \text{ret}(e) \mid \text{mdo}(\Theta; e) \mid \text{doall}(e) \\
 \quad \mid l \mid \text{new}(e) \mid \text{get}(e) \mid \text{set}(e_1, e_2) \mid e_1 + e_2 \mid e!X \mid e \setminus X \\
 \quad \mid [\iota; \Delta] \quad \text{with } \iota \text{ injective and } \text{dom}(\iota) \cap \text{DV}(\Delta) = \emptyset \\
 \Theta ::= \emptyset \mid \Theta, x \Leftarrow e \quad \text{with } x \notin \text{DV}(\Theta) \\
 \Delta ::= \emptyset \mid \Delta, D \quad \text{with } \text{DV}(\Delta) \cap \text{DV}(D) = \text{DN}(\Delta) \cap \text{DN}(D) = \emptyset \\
 D ::= X \triangleleft e \mid x \triangleleft e \quad \text{with } \triangleleft \text{ either } = \text{ or } \Leftarrow
 \end{array}$$

where $o: \text{Name} \xrightarrow{\text{fin}} \mathbf{E}$, $\rho: \text{Var} \xrightarrow{\text{fin}} \mathbf{E}$ and $\iota: \text{Var} \xrightarrow{\text{fin}} \text{Name}$. Some productions have side-conditions, the auxiliary functions DV and DN return the set of variables and component names defined in a sequence Δ of definitions, respectively. For lack of space, the straightforward definitions of DV , DN and FV have been omitted (see the long version of this paper²). The terms include:

- records $\{o\}$, where o is a partial function (since the order of record components is irrelevant), and selection $e.X$ of a record component;
- recursive bindings $\text{let}(\rho; e)$ and recursive monadic bindings $\text{mdo}(\Theta; e)$ of [EL00];
- the operations on references for allocation $\text{new}(e)$, dereferencing $\text{get}(e)$ and assignment $\text{set}(e_1, e_2)$;
- basic mixins $[\iota; \Delta]$ with deferred components ι , and the operations of sum $e_1 + e_2$, freezing $e!X$ and deletion $e \setminus X$ of a component (see [AZ02]).

The basic difference between a record $\{o\}$ and a mixin $[\emptyset; \Delta]$ without deferred components is that Δ may have local (recursive) definitions and computational components. The operation $\text{doall}([\emptyset; \Delta])$ denotes a computation which forces evaluation of all computational components in Δ (eliminates local definitions), and returns a record. Since computations may have side-effects, the order of the bindings in Δ (and Θ) matters.

² <http://www.disi.unige.it/person/AnconaD/Conferences>

Types are defined by $\tau \in \mathbb{T} ::= \dots \mid M\tau \mid \text{ref } \tau \mid \{II\} \mid [II; II']$ where

$II: \text{Name} \xrightarrow{\text{fin}} \mathbb{T}$. The set of types includes computational types $M\tau$, reference types, record types $\{II\}$ and mixin types $[II; II']$. Table 1 gives the typing rules for deriving judgments of the form $\Gamma \vdash_{\Sigma} e: \tau$, which mean “ e is a well-typed term of type τ in Γ and Σ ”, where $\Gamma: \text{Var} \xrightarrow{\text{fin}} \mathbb{T}$ is a type assignment, and $\Sigma: \text{L} \xrightarrow{\text{fin}} \mathbb{T}$ is a signature for locations. The type system enjoys the usual properties of weakening (w.r.t. Γ and Σ) and substitution.

3.1 Simplification

We define a confluent relation on terms (and other syntactic categories), called *simplification*, which induces a congruence on terms. There is no need to define a deterministic simplification strategy, since computational effects (in our case they amount to store changes) are *insensitive* to further simplification (see Theorem 1). Simplification $e_1 \longrightarrow e_2$ is the compatible relation on $\mathbb{E}$ induced by the rewrite rules in Table 2.

In mixin sum (S), deferred components can be shared whereas for the other components disjoint union is performed (recall example 2 in Section 2). Note that, except for $\text{DN}(\Delta_1) \cap \text{DN}(\Delta_2)$, all other conditions can be satisfied by an appropriate α -conversion. The last condition avoids capture of free variables.

In (F), like in example 3, the deferred component X can be frozen only if X is also defined; then, the deferred component $x: X$ is deleted and the local component $x \triangleleft e$ is inserted, which means either $x \Leftarrow e$ if X is defined by $X \Leftarrow e$, or $x = e$ if X is defined by $X = e$. Furthermore $X \triangleleft e$ is transformed into $X = x$ since if X is computational, then e must be evaluated only once³.

In (D), the defined component is simply removed, as in example 4.

Rule (A) expresses `doall` in terms of `mdo`: first, all computational components are evaluated according to the order given in the mixin (recall example 5), then a record value is returned containing both the non computational (o_1) and the computational defined components (o_2) of the mixin; substitution of the non computational local components (ρ) is needed in order to avoid variables to escape from their scope (the `let` construct is used because local variables can be mutually recursive). Finally, note that each computational defined component $X \Leftarrow e$ is transformed into $X = x_X$, with x_X freshly chosen variable, because e must be evaluated only once.

Simplification enjoys the Church Rosser and Subject Reduction properties.

Proposition 1 (CR for $\longrightarrow$). *The relation $\longrightarrow$ is confluent.*

Proof. The simplification rules are left-linear and non-overlapping. $\square$

Proposition 2 (SR for $\longrightarrow$). *If $\Gamma \vdash_{\Sigma} e: \tau$ and $e \longrightarrow e'$, then $\Gamma \vdash_{\Sigma} e': \tau$.*

Proof. By case analysis on the simplification rules. $\square$

³ For simplicity, this transformation is always applied, even though is really needed only when X is computational.

Table 1. Type system

(var)	$\frac{}{\Gamma \vdash_{\Sigma} x: \tau}$	$\Gamma(x) = \tau$	(ret)	$\frac{\Gamma \vdash_{\Sigma} e: \tau}{\Gamma \vdash_{\Sigma} \text{ret}(e): M\tau}$	
(mdo)	$\frac{\{ \Gamma, \Gamma_{\Theta} \vdash_{\Sigma} e: M\tau \mid (x \Leftarrow e) \in \Theta \wedge \tau = \Gamma_{\Theta}(x) \} \quad \Gamma, \Gamma_{\Theta} \vdash_{\Sigma} e': M\tau'}{\Gamma \vdash_{\Sigma} \text{mdo}(\Theta; e'): M\tau'}$	$\text{dom}(\Gamma_{\Theta}) = \text{DV}(\Theta)$			
(let)	$\frac{\{ \Gamma, \Gamma_{\rho} \vdash_{\Sigma} e: \tau \mid e = \rho(x) \wedge \tau = \Gamma_{\rho}(x) \} \quad \Gamma, \Gamma_{\rho} \vdash_{\Sigma} e': \tau}{\Gamma \vdash_{\Sigma} \text{let}(\rho; e'): \tau}$	$\text{dom}(\Gamma_{\rho}) = \text{dom}(\rho)$			
(l)	$\frac{}{\Gamma \vdash_{\Sigma} l: \text{ref}\tau}$	$\Sigma(l) = \tau$	(new)	$\frac{\Gamma \vdash_{\Sigma} e: \tau}{\Gamma \vdash_{\Sigma} \text{new}(e): M(\text{ref}\tau)}$	
(get)	$\frac{\Gamma \vdash_{\Sigma} e: \text{ref}\tau}{\Gamma \vdash_{\Sigma} \text{get}(e): M\tau}$		(set)	$\frac{\Gamma \vdash_{\Sigma} e_2: \tau \quad \Gamma \vdash_{\Sigma} e_1: \text{ref}\tau}{\Gamma \vdash_{\Sigma} \text{set}(e_1, e_2): M(\text{ref}\tau)}$	
(record)	$\frac{\{ \Gamma \vdash_{\Sigma} e: \tau \mid e = o(X) \wedge \tau = \Pi(X) \}}{\Gamma \vdash_{\Sigma} \{o\}: \{\Pi\}}$	$\text{dom}(\Pi) = \text{dom}(o)$			
(select)	$\frac{\Gamma \vdash_{\Sigma} e: \{\Pi\}}{\Gamma \vdash_{\Sigma} e.X: \tau}$	$\tau = \Pi(X)$	(doall)	$\frac{\Gamma \vdash_{\Sigma} e: [\emptyset; \Pi]}{\Gamma \vdash_{\Sigma} \text{doall}(e): M\{\Pi\}}$	
(mixin)	$\frac{\{ \Gamma, \Gamma_1, \Gamma_2 \vdash_{\Sigma} e: \tau \mid (X = e) \in \Delta \wedge \tau = \Pi'(X) \} \quad \{ \Gamma, \Gamma_1, \Gamma_2 \vdash_{\Sigma} e: M\tau \mid (X \Leftarrow e) \in \Delta \wedge \tau = \Pi'(X) \} \quad \{ \Gamma, \Gamma_1, \Gamma_2 \vdash_{\Sigma} e: \tau \mid (x = e) \in \Delta \wedge \tau = \Gamma_2(x) \} \quad \{ \Gamma, \Gamma_1, \Gamma_2 \vdash_{\Sigma} e: M\tau \mid (x \Leftarrow e) \in \Delta \wedge \tau = \Gamma_2(x) \}}{\Gamma \vdash_{\Sigma} [\iota; \Delta]: [\Pi; \Pi']}$	$\text{img}(\iota) = \text{dom}(\Pi)$ $\Gamma_1 \triangleq \Pi \circ \iota$ $\text{DN}(\Delta) = \text{dom}(\Pi')$ $\text{DV}(\Delta) = \text{dom}(\Gamma_2)$			
(sum)	$\frac{\Gamma \vdash_{\Sigma} e_1: [\Pi_1; \Pi'_1] \quad \Gamma \vdash_{\Sigma} e_2: [\Pi_2; \Pi'_2]}{\Gamma \vdash_{\Sigma} e_1 + e_2: [\Pi_1, \Pi_2; \Pi'_1, \Pi'_2]}$	Π_1 compatible with Π_2 $\text{dom}(\Pi'_1) \cap \text{dom}(\Pi'_2) = \emptyset$			
(freeze)	$\frac{\Gamma \vdash_{\Sigma} e: [\Pi; \Pi']}{\Gamma \vdash_{\Sigma} e!X: [\Pi \setminus X; \Pi']}$	$\Pi(X) = \Pi'(X)$			
(delete)	$\frac{\Gamma \vdash_{\Sigma} e: [\Pi; \Pi']}{\Gamma \vdash_{\Sigma} e \setminus X: [\Pi; \Pi' \setminus X]}$	$X \in \text{dom}(\Pi')$			

3.2 Computation

We now define *configurations* $Id \in \text{Conf}$, that represent snapshots of the execution of a program, and the computation relation $\mapsto$ (see Table 3), that describes how program execution evolves. Over these configurations we give an operational semantics that ensures the correct sequencing of computational ef-

Table 2. Simplification rules

-
- (R) $\{o\}.X \longrightarrow e$ provided $e = o(X)$
(L) $\text{let}(\rho; e) \longrightarrow e\{x: \text{let}(\rho; \rho(x)) | x \in \text{dom}(\rho)\}$
(S) $[\iota_1; \Delta_1] + [\iota_2; \Delta_2] \longrightarrow [\iota_1, \iota_2; \Delta_1, \Delta_2]$ provided $[\iota_1, \iota_2; \Delta_1, \Delta_2]$ is well-formed, i.e.
 - $\text{DN}(\Delta_1) \cap \text{DN}(\Delta_2) = \text{DV}(\Delta_1) \cap \text{DV}(\Delta_2) = \text{dom}(\iota_1, \iota_2) \cap \text{DV}(\Delta_1, \Delta_2) = \emptyset$
 - ι_1, ι_2 is an injection (therefore ι_1 is compatible with ι_2)
 - $\text{FV}(\Delta_1) \cap (\text{dom}(\iota_2) \cup \text{DV}(\Delta_2)) = \text{FV}(\Delta_2) \cap (\text{dom}(\iota_1) \cup \text{DV}(\Delta_1)) = \emptyset$
(F) $[\iota, x: X; \Delta, X \triangleleft e, \Delta']!X \longrightarrow [\iota; \Delta, x \triangleleft e, X = x, \Delta']$
(D) $[\iota; \Delta, X \triangleleft e, \Delta'] \setminus X \longrightarrow [\iota; \Delta, \Delta']$
(A) $\text{doall}([\emptyset; \Delta]) \longrightarrow \text{mdo}(|\Delta|; \text{ret}\{o_1, o_2\})\{x: \text{let}(\rho; x) | x \in \text{dom}(\rho)\}$ where
 - $\rho = \{x: e | (x = e) \in \Delta\}$
 - $o_1 = \{X: e | (X = e) \in \Delta\}$, $o_2 = \{X: x_X | X \triangleleft e \in \Delta\}$ with x_X freshly chosen
 - $|\Delta|$ is defined by induction on Δ as follows:
 - * $|\emptyset| = \emptyset$
 - * $|(\Delta, X = e)| = |(\Delta, x = e)| = |\Delta|$
 - * $|(\Delta, X \triangleleft e)| = |\Delta|, x_X \triangleleft e$
 - * $|(\Delta, x \triangleleft e)| = |\Delta|, x \triangleleft e$

fects, by adopting some well-established technique for specifying the operational semantics of programming languages (see [WF94]).

- Stores $\boxed{\mu \in \mathbf{S} \triangleq \mathbf{L} \xrightarrow{\text{fin}} \mathbf{E}}$ map locations to their content.
- Evaluation Contexts $\boxed{E \in \mathbf{EC}::= \square \mid E[\text{mdo}(x \triangleleft \square, \Theta; e)]}$ for terms of computational type.
- A configuration $\boxed{(\mu, e, E) \in \mathbf{Conf} \triangleq \mathbf{S} \times \mathbf{E} \times \mathbf{EC}}$ is a snapshot of the execution of a program: μ is the current store, e is the program fragment under consideration and E is the evaluation context for e .
- Bad terms b are terms that are stuck because they depend on a variable

$$\boxed{b \in \mathbf{BE}::= x \mid b.X \mid b + e \mid e + b \mid b!X \mid b \setminus X \mid \text{doall}(b) \mid \text{get}(b) \mid \text{set}(b, e)}$$

- Computational Redexes r are terms that enable computation (with no need for simplification); when r is a bad term, we raise a run-time error.

$$\boxed{r \in \mathbf{R}::= \text{mdo}(\Theta; e) \mid \text{ret}(e) \mid \text{new}(e) \mid \text{get}(l) \mid \text{set}(l, e) \mid b}$$

Definition 1. The sets $\text{CV}(E)$ and $\text{FV}(E)$ of captured and free variables are

- $\text{CV}(\square) \triangleq \text{FV}(\square) \triangleq \emptyset$
- $\text{CV}(E[\text{mdo}(x \triangleleft \square, \Theta; e)]) \triangleq \text{CV}(E) \cup \{x\} \cup \text{DV}(\Theta)$ and
 $\text{FV}(E[\text{mdo}(x \triangleleft \square, \Theta; e)]) \triangleq \text{FV}(E) \cup (\text{FV}(\Theta, e) \setminus \text{CV}(E[\text{mdo}(x \triangleleft \square, \Theta; e)]))$

Table 3. Computation Relation

Completion step

 $(\text{done}) \ (\mu, \text{ret}(e), \square) \mapsto \text{done}$

Recursive monadic binding steps

 $(\text{M.0}) \ (\mu, \text{mdo}(\emptyset; e), E) \mapsto (\mu, e, E)$
 $(\text{M.1}) \ (\mu, \text{mdo}(x_1 \leftarrow e_1, \Theta; e), E) \mapsto (\mu, e_1, E[\text{mdo}(x_1 \leftarrow \square, \Theta; e)])$
 with the variables in $\text{DV}(x_1 \leftarrow e_1, \Theta)$ renamed to avoid clashes with $\text{CV}(E)$
 $(\text{M.2}) \ (\mu, \text{ret}(e_1), E[\text{mdo}(x_1 \leftarrow \square; e)]) \mapsto (\mu\{\rho\}, e\{\rho\}, E) \text{ where } \rho \triangleq \{x_1: \text{let}(x_1: e_1; x_1)\}$
 $(\text{M.3}) \ (\mu, \text{ret}(e_1), E[\text{mdo}(x_1 \leftarrow \square, x_2 \leftarrow e_2, \Theta; e)]) \mapsto$
 $(\mu\{\rho\}, e_2\{\rho\}, E[\text{mdo}(x_2 \leftarrow \square, \Theta; e)\{\rho\}]) \text{ where } \rho \triangleq \{x_1: \text{let}(x_1: e_1; x_1)\}$

Imperative steps

 $(\text{I.1}) \ (\mu, \text{new}(e), E) \mapsto (\mu\{l: e\}, \text{ret}(l), E) \text{ where } l \notin \text{dom}(\mu)$
 $(\text{I.2}) \ (\mu, \text{get}(l), E) \mapsto (\mu, \text{ret}(e), E) \text{ provided } e = \mu(l)$
 $(\text{I.3}) \ (\mu, \text{set}(l, e), E) \mapsto (\mu\{l: e\}, \text{ret}(l), E) \text{ provided } l \in \text{dom}(\mu)$

Error step caused by a bad term

 $(\text{err}) \ (\mu, b, E) \mapsto \text{err}$

Rules for monadic binding deserve some explanations. Rule (M.0) deals with the special case of empty binding; rule (M.1) starts the computation when the binding is not empty: the first expression of the binding is evaluated and renaming is needed in order to avoid clashes due to nested monadic bindings; rule (M.2) completes the computation of the binding variables: when the last variable has been computed, it can be substituted with its “value” (the `let` construct is used because of mutual recursion) in both the store and the body of `mdo` which now can be evaluated; finally, (M.3) is used for continuing the computation by considering the next binding variable and is similar to (M.2).

The confluent simplification relation $\longrightarrow$ on terms extends in the obvious way to a confluent relation (still denoted $\longrightarrow$) on stores, evaluation contexts, computational redexes and configurations.

A *complete program* corresponds to a closed term $e \in E_0$ (with no occurrences of locations l), and its evaluation starts from the *initial configuration* $(\emptyset, e, \square)$. The following properties ensure that only closed configurations are reachable (by $\longrightarrow$ and $\mapsto$ steps) from the initial one.

Lemma 1.

1. If $(\mu, e, E) \longrightarrow (\mu', e', E')$, then $\text{dom}(\mu) = \text{dom}(\mu')$, $\text{CV}(E) = \text{CV}(E')$, $\text{FV}(\mu') \subseteq \text{FV}(\mu)$, $\text{FV}(e') \subseteq \text{FV}(e)$ and $\text{FV}(E') \subseteq \text{FV}(E)$.
2. If $(\mu, e, E) \mapsto (\mu', e', E')$ and $\text{FV}(e, \mu) \subseteq \text{CV}(E)$ and $\text{FV}(E) = \emptyset$, then $\text{FV}(e', \mu') \subseteq \text{CV}(E')$, $\text{FV}(E') = \emptyset$ and $\text{dom}(\mu) \subseteq \text{dom}(\mu')$.

Bad terms and computational redexes are closed w.r.t. simplification.

Lemma 2. If $b \longrightarrow e$, then $e \in \text{BE}$. If $r \longrightarrow e$, then $e \in \text{R}$.

When the program fragment under consideration is a computational redex, it is irrelevant whether simplification is done before or after a step of computation.

Theorem 1 (Bisimulation). *If $(\mu_1, r_1, E_1) \xrightarrow{*} (\mu_2, r_2, E_2)$, then*

1. $(\mu_1, r_1, E_1) \mapsto Id_1$ implies $\exists Id_2$ s.t. $(\mu_2, r_2, E_2) \mapsto Id_2$ and $Id_1 \xrightarrow{*} Id_2$
2. $(\mu_2, r_2, E_2) \mapsto Id_2$ implies $\exists Id_1$ s.t. $(\mu_1, r_1, E_1) \mapsto Id_1$ and $Id_1 \xrightarrow{*} Id_2$

where Id_1 and Id_2 range over $\text{Conf} \cup \{\text{done}, \text{err}\}$.

Proof. See [MF03]. □

3.3 Type Safety

We go through the proof of type safety for CMS_{do} . The result is standard, but we make some adjustments to the Subject Reduction and Progress properties for $\Longrightarrow \triangleq \longrightarrow \cup \mapsto$, in order to stress the different role of simplification $\longrightarrow$ and computation $\mapsto$. First of all, we define well-formedness for evaluation contexts $\Gamma, \square: M\tau \vdash_{\Sigma} E: M\tau'$ (in Table 4) and configurations $\Gamma \vdash_{\Sigma} (\mu, e, E)$.

Table 4. Well-formed evaluation contexts

$$\begin{array}{c}
 (\square) \frac{}{\emptyset, \square: M\tau \vdash_{\Sigma} \square: M\tau} \\
 \\
 \text{(mdo)} \frac{
 \begin{array}{l}
 \{ \Gamma, x_1: \tau_1, \Gamma_{\Theta} \vdash_{\Sigma} e': M\tau' \mid (x' \leftarrow e') \in \Theta \wedge \tau' = \Gamma_{\Theta}(x') \} \\
 \Gamma, x_1: \tau_1, \Gamma_{\Theta} \vdash_{\Sigma} e: M\tau_2 \\
 \Gamma, \square: M\tau_2 \vdash_{\Sigma} E: M\tau
 \end{array}
 }{\Gamma, x_1: \tau_1, \Gamma_{\Theta}, \square: M\tau_1 \vdash_{\Sigma} E[\text{mdo}(x_1 \leftarrow \square, \Theta; e)]: M\tau} \text{ dom}(\Gamma_{\Theta}) = \text{DV}(\Theta)
 \end{array}$$

Definition 2 (Well-formed configurations). $\Gamma \vdash_{\Sigma} (\mu, e, E) \triangleleft\!\!\!\rightleftharpoons$

- $\text{dom}(\Sigma) = \text{dom}(\mu)$ and $\text{dom}(\Gamma) = \text{CV}(E)$;
- $\mu(l) = e_l$ and $\Sigma(l) = \tau_l$ imply $\Gamma \vdash_{\Sigma} e_l: \tau_l$;
- exists τ such that $\Gamma \vdash_{\Sigma} e: M\tau$ derivable;
- exists τ' such that $\Gamma, \square: M\tau \vdash_{\Sigma} E: M\tau'$ derivable (see Table 4).

The formation rules of Table 4 for deriving $\Gamma, \square: M\tau \vdash_{\Sigma} E: M\tau'$ ensure that

- Γ assigns a type to all captured variables of E , indeed $\text{dom}(\Gamma) = \text{CV}(E)$;
- E has no free variables and cannot capture a variable x twice.

Proposition 3 (SR).

1. If $\Gamma \vdash_{\Sigma} (\mu, e, E)$ and $(\mu, e, E) \longrightarrow (\mu', e', E')$, then $\Gamma \vdash_{\Sigma} (\mu', e', E')$.
2. If $\Gamma \vdash_{\Sigma} (\mu, e, E)$ and $(\mu, e, E) \mapsto (\mu', e', E')$, then
there exist $\Sigma' \supseteq \Sigma$ and Γ' compatible with Γ such that $\Gamma' \vdash_{\Sigma'} (\mu', e', E')$.

Theorem 2 (Progress). *If $\Gamma \vdash_{\Sigma} (\mu, e, E)$, then one of the following cases holds*

1. $e \in \mathbf{R}$ and $(\mu, e, E) \mapsto \cdot$, or
2. $e \notin \mathbf{R}$ and $e \longrightarrow$

Proof. See the long version of this paper available on the web. □

4 Related Work

The notion of mixin module was firstly introduced in Bracha’s PhD thesis [Bra92] as a generalization of the notion of mixin class (see for instance [BC90]). The semantics of the mixin language in [Bra92] is based on the early work on denotational semantics of inheritance [Coo89,Red88] and is defined by a translation into an untyped λ -calculus equipped with a fixpoint operator and a rather rich set of record operators. Furthermore, imperative features are only marginally considered by implicitly using the technique developed in [Hen93] for extending the semantics of inheritance given in [Coo89,Red88] to object-oriented languages with state.

After this pioneer work, some proposals for extending existing languages with a system of mixin modules were considered: [DS96] and [FF98a,FF98b] go in this direction; however, imperative features are not considered and recursion problems are solved by separating initialization from component definition.

The first calculi based on the notion of mixin modules appeared in [AZ99, AZ02] and then in [WV00,MT00], but all of them are defined in a purely functional setting. More recently, [HL02] has considered a *CMS*-like calculus, called *CMS_v*, with a refined type system in order to avoid bad recursion in a call-by-value setting. A separate compilation schema for *CMS_v* has been also investigated by means of a translation down to a call-by-value λ -calculus λ_B extended with a non-standard **let rec** construct, inspired by the calculus defined in [Bou02].

Like *CMS_{do}*, both λ_B and the calculus of Boudol serve as semantic basis for programming languages supporting mixins and introduce non-standard constructs for recursion which can produce terms having an undefined semantics. However, λ_B does not have imperative features, whereas the calculus in [Bou02] does not allow recursion in the presence of side-effects. For instance, in *CMS_{do}* the term $\text{mdo}(x \leftarrow \text{new}(x); \text{ret}(x))$ has a well-defined semantics, whereas the corresponding translated term $\text{let rec } x = \text{ref } x \text{ in } x$ in Boudol’s calculus is not well-typed; indeed, the evaluation of this term gets stuck. Another advantage of our approach is that the separation of concerns made possible by the monadic metalanguage allows us to retain the equational reasoning of *CMS*.

On the other hand, the more refined type systems adopted in [HL02,Bou02] are able to statically detect all bad recursive declarations.

As already mentioned, the definition of the *mdo* construct in *CMS_{do}* is inspired by the work on the semantics of recursive monadic bindings in Haskell [EL00,ELM01,ELM02,EL02]. Our semantics is partly related to that in [ELM01], however the notion of heap in our calculus has been made implicit (thanks to

the `let rec` construct), since we are interested in a more abstract approach; and furthermore, the recursive `do` in [EL02] does not perform an incremental binding as happens in our semantics, but rather all values are bound to the corresponding variables only after all computations in the recursive monadic binding have been evaluated.

5 Conclusion and Future Work

We have defined CMS_{do} , a monadic mixin calculus in which mixin modules can contain components of three kinds: defined (bound to an expression), deferred (declared but not yet defined) and computational (bound to a computation which must be performed before actually using the module for component selection). Mixin modules can be combined by the sum, freeze and restrict operators of CMS ; moreover, a `doall` operator triggers all the computations in a mixin module.

We have provided a simple type system for the language, a simplification relation defined by local rewrite rules with no side-effects (satisfying the CR and SR properties), and a computation relation which models global evaluation able to modify the store (satisfying the SR property). Moreover, we have stated a bisimulation result (simplification does not affect computation steps) and the progress property for the combined relation; however, errors due to bad recursive declarations are only dynamically detected, since here we have preferred to keep a simple type system.

We envisage at least two possibilities which deserve investigation in the direction of defining more refined type systems. First, the dynamic errors due to bad recursive declarations mentioned above could be detected by introducing a type system similar to that in [HL02,Bou02] keeping explicit trace of *dependencies* between the evaluation of two computational components. On a different side, a type system distinguishing between modules possibly containing some computational components (or variables) and those with no computational components (and variables), would allow selection on CMS mixins, so that CMS could be more directly embedded into CMS_{do} .

For what concerns applications, CMS_{do} can be considered a powerful kernel calculus allowing to express, on one side, a variety of different operators for combination of software modules (including linking, parameterized modules as ML functors, overriding in the sense of object-oriented languages, see [AZ02] for details), on the other side different choices in the evaluation of computations. In particular, we mention at least two relevant scenarios of application: the modeling of object-oriented features, including the difference between computations which must be performed before instantiating a class, as field initializers, and computations which are evaluated each time they are selected, as methods; and the possibility of expressing different policies for dynamic linking and verification.

References

- [AZ99] D. Ancona and E. Zucca. A primitive calculus for module systems. In G. Nadathur, editor, *Principles and Practice of Declarative Programming, 1999*, number 1702 in Lecture Notes in Computer Science, pages 62–79. Springer Verlag, 1999.
- [AZ02] D. Ancona and E. Zucca. A calculus of module systems. *Journal of Functional Programming*, 12(2):91–132, March 2002.
- [BC90] G. Bracha and W. Cook. Mixin-based inheritance. In *Proc. of the Joint ACM Conf. on Object-Oriented Programming, Systems, Languages and Applications and the European Conference on Object-Oriented Programming*, October 1990.
- [Bou02] G. Boudol. The recursive record semantics of objects revisited. To appear in *Journal of Functional Programming*, 2002.
- [Bra92] G. Bracha. *The Programming Language JIGSAW: Mixins, Modularity and Multiple Inheritance*. PhD thesis, Department of Comp. Sci., Univ. of Utah, 1992.
- [Coo89] W.R. Cook. *A Denotational Semantics of Inheritance*. PhD thesis, Dept. of Computer Science, Brown University, 1989.
- [DS96] D. Duggan and C. Sourelis. Mixin modules. In *Intl. Conf. on Functional Programming*, Philadelphia, May 1996. ACM Press.
- [EL00] L. Erkök and J. Launchbury. Recursive monadic bindings. In *Intl. Conf. on Functional Programming 2000*, pages 174–185, 2000.
- [EL02] L. Erkök and J. Launchbury. A recursive do for Haskell. In *Haskell Workshop’02*, pages 29–37, 2002.
- [ELM01] L. Erkök, J. Launchbury, and A. Moran. Semantics of *fixIO*. In *FICS’01*, 2001.
- [ELM02] L. Erkök, J. Launchbury, and A. Moran. Semantics of value recursion for monadic input/output. *Journal of Theoretical Informatics and Applications*, 36(2):155–180, 2002.
- [FF98a] R.B. Findler and M. Flatt. Modular object-oriented programming with units and mixins. In *Intl. Conf. on Functional Programming 1998*, September 1998.
- [FF98b] M. Flatt and M. Felleisen. Units: Cool modules for HOT languages. In *PLDI’98 - ACM Conf. on Programming Language Design and Implementation*, pages 236–248, 1998.
- [Hen93] A. V. Hense. Denotational semantics of an object-oriented programming language with explicit wrappers. *Formal Aspects of Computing*, 5(3):181–207, 1993.
- [HL94] R. Harper and M. Lillibridge. A type-theoretic approach to higher-order modules with sharing. In *Conference record of POPL ’94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 123–137, 1994.
- [HL02] T. Hirschowitz and X. Leroy. Mixin modules in a call-by-value setting. In D. Le Métayer, editor, *ESOP 2002 - Programming Languages and Systems*, number 2305 in Lecture Notes in Computer Science, pages 6–20. Springer Verlag, 2002.
- [Ler94] X. Leroy. Manifest types, modules and separate compilation. In *Proc. 21st ACM Symp. on Principles of Programming Languages*, pages 109–122. ACM Press, 1994.

- [MF03] E. Moggi and S. Fagorzi. A Monadic Multi-stage Metalanguage. In A.D. Gordon, editor, *Foundations of Software Science and Computational Structures - FOSSACS 2003*, volume 2620 of *LNCS*, pages 358–374. Springer Verlag, 2003.
- [MT00] E. Machkasova and F.A. Turbak. A calculus for link-time compilation. In G. Smolka, editor, *ESOP 2000 - Programming Languages and Systems*, number 1782 in *Lecture Notes in Computer Science*, pages 260–274, Berlin, 2000. Springer Verlag.
- [Red88] U. S. Reddy. Objects as closures: Abstract semantics of object-oriented languages. In *Proc. ACM Conf. on Lisp and Functional Programming*, pages 289–297, 1988.
- [WF94] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, 1994.
- [WV00] J.B. Wells and R. Vestergaard. Equational reasoning for linking with first-class primitive modules. In G. Smolka, editor, *ESOP 2000 - Programming Languages and Systems*, number 1782 in *Lecture Notes in Computer Science*, pages 412–428, Berlin, 2000. Springer Verlag.

Decision Problems for Language Equations with Boolean Operations

Alexander Okhotin

School of Computing, Queen's University, Kingston, Ontario, Canada K7L3N6
okhotin@cs.queensu.ca

Abstract. The paper studies resolved systems of language equations that allow the use of all Boolean operations in addition to concatenation. Existence and uniqueness of solutions are shown to be their nontrivial properties, these properties are given characterizations by first order formulae, and the position of the corresponding decision problems in the arithmetical hierarchy is determined. The class of languages defined by components of unique solutions of such systems is shown to coincide with the class of recursive languages.

Keywords: language equations, Boolean operations, recursive sets.

1 Introduction

The theory of language equations that correspond to context-free grammars is a well established area of formal language theory [1]. These equations contain semiring operations of sum (interpreted as set-theoretic union) and product (concatenation of languages), and the basic properties of systems of such equations can be derived from more general algebraic results [3].

These general methods from semiring theory are restricted to algebraic structures with two operations, and therefore in order to consider language equations augmented with additional operations one has to investigate new methods. Numerous types of language equations are discussed in the recently appeared book [4]; among those sufficiently studied are systems over various sets of Boolean operations where some restrictions are imposed on the use of concatenation. Systems of language equations with unrestricted concatenation, union and intersection (but without complement) were considered in [6], where they were proved equivalent to *conjunctive grammars* [5], which are context-free grammars extended with an explicit intersection operation. This result was obtained using the same least fixed point techniques as used in the context-free case (following [1] rather than [3]), which actually do not rely on the properties of semirings and require only the monotonicity of all operations with respect to set inclusion.

The next obvious class of language equations to consider is a further extension of the equations of [6], where complement, a nonmonotonic operation, is also allowed. However, studying these equations using the existing methods reveals certain complications. Such systems may have no solutions (e.g., $X = \neg X$),

unique solution ($X = \neg(XaX)$ with solution $(aa)^*$) or multiple solutions ($X = X$); in the latter case the solutions can be pairwise incomparable ($X = \neg Y, Y = Y$). It is not immediately evident how one can distinguish between these types of systems. Moreover, as the existence of the least solution is not guaranteed, it is unclear how such systems can be used to define languages.

These issues are essential for any study of such language equations, and the present paper does settle them. In Section 2 the systems of language equations studied in this paper are defined; following the practice set in [5,6], the Boolean operations used in equations are viewed as logical connectives (conjunction, disjunction and negation), and then these connectives are interpreted as set-theoretic operations with languages. Section 3 introduces the apparatus of solutions modulo a language, which is used in all subsequent results. Sections 4 and 5 investigate existence and uniqueness of solutions respectively, characterize these properties by first-order formulae and determine their positions in the arithmetical hierarchy. Some basic results on systems with unique solution are obtained in Section 6.

2 Formulae and Equations

Definition 1 (Language formula). *Let Σ be a finite nonempty alphabet and let $X = (X_1, \dots, X_n)$ ($n \geq 1$) be a vector of language variables. Language formulae over the alphabet Σ in variables X are defined inductively as follows:*

- the empty string ϵ is a formula;
- any symbol from Σ is a formula;
- any variable from X is a formula;
- if φ and ψ are formulae, then $(\varphi\psi)$, $(\varphi \& \psi)$, $(\varphi \vee \psi)$ and $(\neg\varphi)$ are formulae.

As in logic formulae, we shall omit the parentheses whenever possible, using the following default precedence of operators: the concatenation has the highest precedence and is followed by the logical connectives arranged in their usual order $\neg$, $\&$ and $\vee$. If needed, this default precedence will be overridden with parentheses. For instance, $XY \vee \neg aX \& aY$ means the same as $(X \cdot Y) \vee ((\neg(a \cdot X)) \& (a \cdot Y))$.

We have defined the syntax of formulae; let us now define their semantics by interpreting the connectives as operations on languages, thus associating a language function with every formula:

Definition 2 (Value of a formula). *Let φ be a formula over an alphabet Σ in variables $X = (X_1, \dots, X_n)$. Let $L = (L_1, \dots, L_n)$ be a vector of languages over Σ . The value of the formula φ on the vector of languages L , denoted as $\varphi(L)$, is defined inductively on the structure of φ :*

- $\epsilon(L) = \{\epsilon\}$,
- $a(L) = \{a\}$ for every $a \in \Sigma$,
- $X_i(L) = L_i$ for every i ($1 \leq i \leq n$),
- $\psi\xi(L) = \psi(L) \cdot \xi(L)$,

- $(\psi \vee \xi)(L) = \psi(L) \cup \xi(L)$,
- $(\psi \& \xi)(L) = \psi(L) \cap \xi(L)$ and
- $(\neg\psi)(L) = \Sigma^* \setminus \psi(L)$.

The value of a vector of formulae $\varphi = (\varphi_1, \dots, \varphi_l)$ on a vector of languages $L = (L_1, \dots, L_n)$ is the vector of languages $\varphi(L) = (\varphi_1(L), \dots, \varphi_l(L))$.

Note that all the mentioned binary logical operations, as well as concatenation, are associative, and therefore there is no need to disambiguate formulae like XYZ or $X \vee Y \vee Z$ with extra parentheses.

Definition 3 (System of equations). Let Σ be an alphabet. Let $n \geq 1$. Let $X = (X_1, \dots, X_n)$ be a set of language variables. Let $\varphi = (\varphi_1, \dots, \varphi_n)$ be a vector of formulae in variables X over the alphabet Σ . Then

$$\begin{cases} X_1 = \varphi_1(X_1, \dots, X_n) \\ \vdots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{cases} \quad (1)$$

is called a resolved system of equations over Σ in variables X . (1) can also be denoted in the vector form as $X = \varphi(X)$.

Definition 4 (Solution of a resolved system). A vector of languages $L = (L_1, \dots, L_n)$ is said to be a solution of a system (1), if for every i ($1 \leq i \leq n$) it holds that $L_i = \varphi_i(L_1, \dots, L_n)$. In the vector form, this is denoted as $L = \varphi(L)$.

Let us give a sample system of language equations with Boolean operations:

Example 1. The following system of equations over the alphabet $\Sigma = \{a, b\}$

$$\begin{aligned} X_1 &= \neg X_2 X_3 \& \neg X_3 X_2 \& X_4 & X_3 &= (a \vee b) X_3 (a \vee b) \vee b \\ X_2 &= (a \vee b) X_2 (a \vee b) \vee a & X_4 &= (aa \vee ab \vee ba \vee bb) X_4 \vee \epsilon \end{aligned} \quad (2)$$

has unique solution $(\{ww \mid w \in \{a, b\}^*\}, \{xay \mid x, y \in \{a, b\}^*, |x| = |y|\}, \{xyb \mid x, y \in \{a, b\}^*, |x| = |y|\}, \{u \mid u \in \{a, b\}^{2n}, n \geq 0\})$.

Following the practice of the theory of context-free languages, we can consider the first variable of the system to be the start variable. Then the system of equations (2) could be said to denote the language $\{ww \mid w \in \{a, b\}^*\}$.

This semantics of the first component of the unique solution will be used in this paper as the only interpretation of language equations. There is one construct expressible by such equations that is worth particular mention: it turns out that it is possible to simulate any inclusions using resolved equations. In order to require that $\varphi(L) \subseteq \psi(L)$ for every solution L of some system, it suffices to add an auxiliary variable Y and an equation

$$Y = \neg Y \& \varphi \& \neg \psi, \quad (3)$$

which is a contradiction unless the mentioned inclusion holds.

3 Solutions Modulo a Language

The notion of equality of languages modulo a third language will be used as one of the main tools in course of this paper.

Definition 5. *Let us call two languages $L_1, L_2 \subseteq \Sigma^*$ equal modulo a third language $M \subseteq \Sigma^*$ (denoted $L_1 = L_2 \pmod{M}$), if $L_1 \cap M = L_2 \cap M$. This relation can be extended to vectors of languages by saying that $L' = (L'_1, \dots, L'_n)$ equals $L'' = (L''_1, \dots, L''_n)$ modulo M if $L'_i = L''_i \pmod{M}$ for all i .*

Every two languages are equal modulo $\emptyset$. Equality modulo Σ^* means equality in the ordinary sense. Obviously, equality modulo M implies equality modulo every subset of M . It is also easy to prove that for every fixed M equality modulo M is an equivalence relation.

Definition 6. *For every string $w \in \Sigma^*$, define substrings(w) = $\{y \mid w = xyz \text{ for some } x, z \in \Sigma^*\}$. For every language $L \subseteq \Sigma^*$, define substrings(L) = $\bigcup_{w \in L} \text{substrings}(w)$. Similarly, define proper substrings(w) = $\text{substrings}(w) \setminus \{w\}$ and proper substrings(L) = $\bigcup_{w \in L} \text{proper substrings}(w)$.*

A language L is said to be closed under substring, if $\text{substrings}(L) = L$, i.e. all substrings of every string from L are also in L .

The languages $\emptyset$, $\{\epsilon\}$ and Σ^* are simplest examples of languages closed under substring. The study of the properties of a given language modulo every language closed under substring can sometimes be used to cross the borderline between finite and infinite and determine some properties of the whole given language. Here is one trivial result of this kind:

Proposition 1. *If two languages (vectors of languages) L', L'' are equal modulo every finite language M closed under substring, then $L' = L''$.*

In equivalent form, if two languages (vectors of languages) are not equal, then they are not equal modulo some finite language closed under substring.

Indeed, if $L' \neq L''$, then the symmetric difference $L' \Delta L''$ contains some string w , and therefore L' and L'' are not equal modulo $\text{substrings}(w)$.

In the following we shall obtain several nontrivial characterizations of languages by their properties modulo finite languages. For now, let us obtain an important basic result stating that every language formula, as defined in Section 2, preserves equality modulo every fixed language closed under substring.

Lemma 1. *Let $\varphi(X_1, \dots, X_n)$ be a formula over an alphabet Σ . Let $M \subseteq \Sigma^*$ be an arbitrary language closed under substring. Then, if two vectors of languages, $(L'_1, \dots, L'_n)$ and $(L''_1, \dots, L''_n)$, are equal modulo M , this implies that $\varphi(L'_1, \dots, L'_n)$ and $\varphi(L''_1, \dots, L''_n)$ are also equal modulo M .*

Proof. The proof is a straightforward induction on the structure of φ :

- $\epsilon(L') = \{\epsilon\} = \epsilon(L'')$ and thus $\epsilon(L') = \epsilon(L'') \pmod{M}$.
- $a(L') = \{a\} = a(L'')$, and therefore $a(L') = a(L'') \pmod{M}$.
- $X_i(L') = L'_i$ and $X_i(L'') = L''_i$. Since $L'_i = L''_i \pmod{M}$ by assumption, it holds that $X_i(L') = X_i(L'') \pmod{M}$.

- Let w be a string from M . If $w \in \psi\xi(L') = \psi(L') \cdot \xi(L')$, then there exists a factorization $w = uv$, such that $u \in \psi(L')$ and $v \in \xi(L')$. $u, v \in M$ by the closure of M under substring. Now, by the induction hypothesis, $u \in \psi(L')$ if and only if $u \in \psi(L'')$, and $v \in \xi(L')$ if and only if $v \in \xi(L'')$. Therefore, $uv \in \psi(L'') \cdot \xi(L'') = \psi\xi(L'')$.
- If $w \in M$ is in $(\psi \vee \xi)(L') = \psi(L') \cup \xi(L')$, then w is in $\psi(L')$ or in $\xi(L')$ or in both. By induction hypothesis, $w \in \psi(L')$ iff $w \in \psi(L'')$, and $w \in \xi(L')$ iff $w \in \xi(L'')$, which means that w must be in one of $\psi(L'')$, $\xi(L'')$, which is equivalent to $w \in (\psi \vee \xi)(L'')$.
- The cases of $\varphi = (\psi \& \xi)$ and $\varphi = (\neg\psi)$ are proved similarly. $\square$

Definition 7. Let $X = \varphi(X)$ be a system of equations and let M be a language closed under substring. A vector $L = (L_1, \dots, L_n)$ is said to be a solution of the system $X = \varphi(X)$ modulo M if $\varphi_i(L) = L_i \pmod{M}$ for every i .

Equality of solutions modulo some M shall always be considered in the sense of equality modulo M ; this notion of equality will be used whenever solution modulo M will be said to be unique.

Proposition 2 (On nested moduli). A solution of a system $X = \varphi(X)$ modulo some language M closed under substring is a solution of the same system modulo every subset of M closed under substring. In particular, every solution in the usual sense (i.e., modulo Σ^*) is a solution modulo every language closed under substring.

4 Existence of Solution

As already noted in the Introduction, a system of language equations with negation does not necessarily have a solution: a single equation $X_1 = \neg X_1$ is the simplest example of such a system. In this section we develop a necessary and sufficient condition of existence of solutions based upon solutions modulo finite languages.

To begin with, let us prove some technical results on the relationship between solutions modulo finite languages and solutions in the ordinary sense (which may be regarded in this context as solutions modulo the set of all strings). The first of these results is quite obvious:

Lemma 2 (Finite refutation of a non-solution). If $L = (L_1, \dots, L_n)$ is not a solution of a system $X = \varphi(X)$, then there exists a finite language M closed under substring, such that L is not a solution of the system modulo M .

Equivalently, if a vector of languages L is a solution of a system $X = \varphi(X)$ modulo every finite language M closed under substring, then L is a solution of the system.

Proof. If L is not a solution of a system $X = \varphi(X)$, then $L \neq \varphi(L)$. By Proposition 1, there exists a modulus M closed under substring, such that $L \neq \varphi(L) \pmod{M}$, which means that L is not a solution modulo M . $\square$

Lemma 3 (Extension of a solution modulo a finite language). *Let $X = \varphi(X)$ be a system, let M be a finite language closed under substring, let $L_M = (L_1, \dots, L_n)$ be a solution of the system modulo M .*

If for every finite language $M' \supseteq M$ closed under substring the system has a solution modulo M' , which coincides with L_M modulo M , then the system has a solution, which also coincides with L_M modulo M .

Before proceeding to the proof, let us note that the statement of Lemma 3 is fairly general; for instance, if we fix M to be the empty set, then the lemma will state that any system of equations that has a solution modulo every finite language closed under substring necessarily has a solution.

Proof. We shall say that a solution L_M modulo M is *refuted modulo $M' \supseteq M$* (where both M and M' are finite languages closed under substring) if no solution modulo M' coincides with L_M modulo M . A solution L_M modulo M is said to be *refutable*, if it is refuted modulo some $M' \supseteq M$ (finite, closed under substring), and *unrefutable* otherwise.

Now we can reformulate the statement of the lemma as follows: if a system has an unrefutable solution L_M modulo finite M closed under substring, then L_M can be extended to a solution L of the whole system, such that $L = L_M \pmod{M}$.

Consider an arbitrary ascending sequence of nested finite moduli (each closed under substring)

$$M = M_0 \subset M_1 \subset M_2 \subset \dots \subset M_k \subset \dots \quad (4)$$

that converges to Σ^* in the sense that $\bigcup_{k=0}^{\infty} M_k = \Sigma^*$. Let us show that there exists a sequence of vectors of finite languages

$$L^{(0)} \preceq L^{(1)} \preceq L^{(2)} \preceq \dots \preceq L^{(k)} \preceq \dots, \quad (5)$$

monotonically increasing with respect to componentwise set inclusion “ $\preceq$ ”, such that each $L^{(k)}$ is an unrefutable solution modulo the corresponding M_k . The proof is not constructive; we inductively show the existence of consecutive terms of this sequence.

Basis. $L^{(0)} = L_M$ is an unrefutable solution modulo M by the assumption.

Induction Step. Let $L^{(k)}$ be an unrefutable solution modulo M_k , and let

$$L^{[1]}, L^{[2]}, \dots, L^{[m]} \quad (6)$$

be all solutions modulo M_{k+1} that coincide with $L^{(k)}$ modulo M_k . Let us prove that at least one of these solutions modulo M_{k+1} must be unrefutable. Suppose the contrary, i.e., that each $L^{[i]}$ is refuted modulo some language $M^{[i]} \supseteq M_{k+1}$. Then all (6) are refuted modulo the language

$$M^{[1..m]} = \bigcup_{i=1}^m M^{[i]} \supseteq M_{k+1} \quad (7)$$

Since $L^{(k)}$ is an unrefutable solution modulo M_k , it is not refuted modulo $M^{[1..m]}$, and thus there exists a solution $L' = (L'_1, \dots, L'_n)$ modulo $M^{[1..m]}$, which coincides with $L^{(k)}$ modulo M_k . By Proposition 2, $(L'_1 \cap M_{k+1}, \dots, L'_n \cap M_{k+1})$ is a solution modulo M_{k+1} , and it still coincides with $L^{(k)}$ modulo M_k . By the construction of the collection (6), it must be among $\{L^{(i)}\}_{i=1}^m$ and thus be refuted modulo $M^{[1..m]}$. However, the solution L' modulo $M^{[1..m]}$ witnesses the opposite.

The contradiction obtained proves that one of the solutions (6) modulo M_{k+1} must be unrefutable. Let $L^{[i]}$ be this unrefutable solution modulo M_{k+1} and define $L^{(k+1)} = L^{[i]}$.

Having obtained the increasing sequence (5), consider its limit

$$L = \left(\bigcup_{k=0}^{\infty} L_1^{(k)}, \dots, \bigcup_{k=0}^{\infty} L_n^{(k)} \right) \quad (8)$$

Clearly, $L = L^{(k)} \pmod{M_k}$ for every k , and therefore L is a solution modulo every M_k . Let us show that L is a solution modulo every fixed finite language M' closed under substring. Since the sequence $\{M_k\}_{k=0}^{\infty}$ is ascending and $\bigcup_{k=0}^{\infty} M_k = \Sigma^*$, there exists k , such that $M' \subseteq M_k$. Because L is a solution modulo M_k , it is a solution modulo M' by Proposition 2.

Therefore, L is a solution of the whole system by Lemma 2. $\square$

Now we can use these technical results to obtain the following characterization of the systems of equations that have solutions:

Theorem 1 (Criterion of solution existence). *A system has a solution if and only if it has a solution modulo every finite language closed under substring.*

Proof. $\Rightarrow$ If $L = (L_1, \dots, L_n)$ is a solution, then it is a solution modulo every finite language M closed under substring by Proposition 2.

$\Leftarrow$ It suffices to apply Lemma 3 with $M = \emptyset$. $\square$

The condition given by Theorem 1 is actually a first order formula with one universal quantifier over a countably infinite set. Hence, the set of systems that have at least one solution is co-recursively enumerable [2]. It turns out that the problem is hard for this class as well (which implies its undecidability).

Theorem 2. *The set of systems that have solutions is co-RE-complete.*

Proof. Membership in co-RE. The complement of the problem can be accepted by a nondeterministic Turing machine that guesses a finite modulus and then accepts if the given system has no solutions modulo this language, and rejects otherwise. The correctness is given by Theorem 1.

Co-RE-hardness. Reduction from the complement of Post Correspondence Problem. Given an alphabet Σ and an instance $\{(u_i, v_i)\}_{i=1}^k$ (where $u_i, v_i \in \Sigma^*$) of PCP, consider the alphabet $\Sigma \cup \{b_1, \dots, b_k\}$, where b_i are assumed to be not in Σ . Construct the system

$$\begin{aligned} X_1 &= \neg X_1 \& X_2 \& X_3 \\ X_2 &= b_1 X_2 u_1 \vee \dots \vee b_k X_2 u_k \vee b_1 u_1 \vee \dots \vee b_k u_k \\ X_3 &= b_1 X_3 v_1 \vee \dots \vee b_k X_3 v_k \vee b_1 v_1 \vee \dots \vee b_k v_k \end{aligned} \quad (9)$$

Let us show that the system (9) has solutions if and only if the instance of PCP is a no-instance. The equations for X_2 and X_3 uniquely determine two languages, L_2 and L_3 ; each of them is a linear context-free language. Every solution of the system (9) must be of the form (L, L_2, L_3) for some $L \subseteq \Sigma^*$.

If the instance of PCP is a yes-instance, then the language $L_2 \cap L_3$ is nonempty, i.e., there exists a string $w \in L_2 \cap L_3$. Suppose there exists a solution (L, L_2, L_3) of the system (9). Then, by the first equation of the system, $w \in L$ if and only if $w \notin L$, which is a contradiction.

If Post correspondence problem does not have solutions, then $L_2 \cap L_3 = \emptyset$ and the triple $(\emptyset, L_2, L_3)$ is the unique solution of the system (9). $\square$

5 Uniqueness of Solution

In Section 4 we have proved that a system has solutions if and only if it has solutions modulo every language closed under substring. However, it turns out that the same property does not hold in respect to the uniqueness of solution, and a system can have multiple solutions modulo every finite language, but still unique solution. Let us consider an example of such a system.

Example 2. Let $\Sigma = \{a\}$ and consider the system

$$\begin{aligned} X_1 &= X_1 \\ X_2 &= \neg X_2 \& a X_1 \end{aligned}$$

Every finite nonempty $M \subset a^*$ closed under substring is of the form $\{\epsilon, a, aa, \dots, a^l\}$ for some $l \geq 0$. The system has two exactly two solutions modulo every such $M - (\emptyset, \emptyset)$ and $(\{a^l\}, \emptyset)$. However, the whole system has unique solution $(\emptyset, \emptyset)$.

In this example, in order to check the membership of a string of length l in the components of the unique solution, one has to consider the strings of length $l + 1$. This illustrates the property of systems of language equations with unique solution that the membership of *longer* strings in the solution may in fact determine the membership of *shorter* strings, which is a quite unexpected context-dependency. It is not hard to show that the range of this context-dependency can be unlimited, i.e., the membership of shorter strings may depend on the membership of strings that are arbitrarily longer.

There exists the following necessary and sufficient condition of solution uniqueness similar to Theorem 1:

Theorem 3 (Criterion of solution uniqueness). *A system has unique solution if and only if for every finite language M closed under substring there exists a finite language $M' \supseteq M$ closed under substring, such that there exists at least one solution of the system modulo M' , and all the solutions modulo M' are equal modulo M .*

Proof. $\Rightarrow$ Let a system $X = \varphi(X)$ have unique solution $L = (L_1, \dots, L_n)$, and suppose that there exists a finite modulus M closed under substring, such that

for every finite modulus $M' \supseteq M$ closed under substring there exists a solution modulo M' , which is different from L modulo M (another possibility that there could be no solutions modulo M' is ruled out by Theorem 1).

This means that there exists a solution L' modulo M that is not refuted on any finite modulus $M' \supseteq M$. By Lemma 3, this L' can be extended to a solution of the whole system, which equals L' modulo M and thus differs from L already modulo M . Therefore, the system has multiple solutions. The contradiction obtained proves the necessity claim.

⊕ Let a system $X = \varphi(X)$ be such that for every finite modulus M closed under substring there exists a finite modulus $M' \supseteq M$ closed under substring, such that all solutions of the system modulo M' are equal modulo M .

Suppose that the system has at least two distinct solutions, $L' = (L'_1, \dots, L'_n)$ and $L'' = (L''_1, \dots, L''_n)$. $L' \neq L''$ means that $L' \neq L'' \pmod{M}$ for some finite modulus M closed under substring. By assumption, for this particular M there exists a finite modulus $M' \supseteq M$ closed under substring, such that all solutions modulo M' are equal modulo M . However, by Proposition 2, L' and L'' are solutions of the system modulo M' , and therefore must coincide modulo M , which yields a contradiction. $\square$

The necessary and sufficient condition of solution uniqueness given by Theorem 3 gives a characterization of the set of systems that have unique solution using a first-order formula with one universal quantifier and one existential quantifier over a denumerable set. This yields the following result:

Theorem 4. *The set of systems that have exactly one solution is Π_2 -complete.*

Proof. Membership in Π_2 . Following Theorem 3, let us denote uniqueness of solution with a first-order formula

$$\phi(w) = \forall x \exists y R(x, y, w), \quad (10)$$

where R is a recursive predicate that evaluates to true on a triple (x, y, w) if and only if

- i. w is a syntactically valid description of an alphabet Σ and of a system of language equations over Σ ,
- ii. x and y describe two finite languages $M_x \subseteq M_y \subset \Sigma^*$, each closed under substring,
- iii. the system denoted by w has solutions modulo the language denoted by y , and all of these solutions coincide modulo the language denoted by x .

The correctness of this representation is given by Theorem 3, while first-order formulae of the form (10) are known to characterize the class Π_2 [2].

Π_2 -hardness. Reduction from Turing machine universality problem, which is stated as “Given a Turing machine T over an alphabet Σ , determine whether $L(T) = \Sigma^*$ ” and is known to be complete for Π_2 [7].

Let T be an arbitrary given Turing machine that has finite input alphabet Σ , finite work alphabet $V \supset \Sigma$ and finite set of states Q . Consider the language

of all accepting computations of T , where the computation on input $w \in \Sigma^*$ is encoded as a string over the alphabet $\Sigma \cup V \cup Q \cup \{\#\}$ of the form

$$w\#ID(T, w, 0)\#ID(T, w, 1)\#\dots\#ID(T, w, l), \quad (11)$$

where every $ID(T, w, i)$ denotes in some form the instantaneous description of T at the i -th step of computation on w , and the machine accepts after exactly l steps. It is a well-known result that for some quite natural encodings of instantaneous descriptions this language is an intersection of two context-free languages, and therefore can be denoted as the first component of a unique solution of a system of language equations that contains disjunction and conjunction.

For any Turing machine T as above, let $X = \varphi(X)$ be such a system of language equations. Construct the following system:

$$Y = Y \quad (12a)$$

$$Z_1 = \epsilon \vee \bigvee_{a \in \Sigma \cup V \cup Q \cup \{\#\}} aZ_1 \quad (12b)$$

$$Z_2 = \epsilon \vee \bigvee_{a \in \Sigma} aZ_2 \quad (12c)$$

$$T_1 = \neg T_1 \& X_1 \& \neg Y \# Z_1 \quad (12d)$$

$$T_2 = \neg T_2 \& Y \& \neg Z_2 \quad (12e)$$

$$\left. \begin{array}{l} X_1 = \varphi_1(X_1, \dots, X_n) \\ \vdots \\ X_n = \varphi_n(X_1, \dots, X_n) \end{array} \right\} \begin{array}{l} \text{The system for the language of all} \\ \text{accepting computations of } T. \end{array} \quad (12f)$$

The equations for T_1 and T_2 of this resolved system implement the following inclusions using the method of (3):

$$X_1 \subseteq Y\#(\Sigma \cup V \cup Q \cup \{\#\})^* \quad (13a)$$

$$Y \subseteq \Sigma^* \quad (13b)$$

The inclusion (13a) states that every string accepted by T should be in the Y component of every solution of the system. The inclusion (13b) restricts the variable Y to languages over the input alphabet of the Turing machine. So, the set of solutions of the system (12) is

$$\{(L', (\Sigma \cup V \cup Q \cup \{\#\})^*, \Sigma^*, \emptyset, \emptyset, L_1, \dots, L_n) \mid \underbrace{L(T) \subseteq L' \subseteq \Sigma^*}_{(14^*)}\}, \quad (14)$$

where $(L_1, \dots, L_n)$ is the unique solution of the system $X = \varphi(X)$. The solution of (12) is easily seen to be unique if and only if the bounds (14*) are tight, i.e., $L(T) = \Sigma^*$.

This completes our reduction from Turing machine universality problem. Since this problem is Π_2 -complete, we obtain Π_2 -hardness of the solution uniqueness problem for systems of language equations, which, together with its membership in Π_2 established above, allows to conclude that it is Π_2 -complete. $\square$

Although the hardness part of Theorem 4 shows that there is no decision procedure to determine whether an arbitrary given system has unique solution, if a certain system is somehow known to have unique solution, then it is possible to compute this solution modulo any given finite language M closed under substring using the characterization given in Theorem 3.

6 Systems with Unique Solution and Their Properties

Let us consider systems of language equations with unique solutions as a tool for defining formal languages and determine the class of languages they can define.

For systems with disjunction only this is the class of context-free languages. For systems with disjunction and conjunction [6] this is the family of languages generated by conjunctive grammars, which is situated in the middle between context-free and context-sensitive languages. Let us prove the following characterization for systems that allow the complete set of logical connectives:

Theorem 5. *The class of languages defined by systems of language equations with Boolean operations that have unique solution, as the first component of this solution, is exactly the class of recursive sets.*

Proof. First of all, let us show that the first component of the solution of any system $X = \varphi(X)$ that has unique solution is always a recursive language. This is given by the following decision procedure that determines the membership of strings in this first component:

Given $w \in \Sigma^*$, let $M = \text{substrings}(w)$.

For all finite moduli $M' \supseteq M$ closed under substring:

If all solutions of $X = \varphi(X)$ modulo M' coincide modulo M

Let $L = (L_1, \dots, L_n)$ be the common part modulo M of solutions modulo M' .

Accept if $w \in L_1$, reject if $w \notin L_1$.

The loop *for all finite moduli* considers all finite languages closed under substring (they are countably many) in any order. Since $X = \varphi(X)$ has unique solution, then, by Theorem 3, the modulus sought in the *if* statement will eventually be found and therefore this algorithm always terminates.

Now let us demonstrate that an arbitrary recursive set $L \subseteq \Sigma^*$ can be denoted by a system of language equations with unique solution, in the sense that the first component of this unique solution is L . Let $T = (\Sigma, V, Q, q_0, \delta, F)$ be a Turing machine that halts on any input and accepts the language L . Let $X = \varphi(X)$ and $Y = \psi(Y)$ be systems of language equations over the alphabet $\Sigma \cup V \cup Q \cup \{\#\}$, such that the first component of the unique solution of $X = \varphi(X)$ is the language of all *accepting* computations of T , while the first component of the unique solution of $Y = \psi(Y)$ is the language of all *rejecting* computations of T (this system is constructed similarly to $X = \varphi(X)$). Construct the following system of equations:

$$Z = Z \quad (15a)$$

$$Z' = Z' \quad (15b)$$

$$U = \epsilon \vee \bigvee_{a \in \Sigma \cup V \cup Q \cup \{\#\}} aU \quad (15c)$$

$$V = \epsilon \vee \bigvee_{a \in \Sigma} aV \quad (15d)$$

$$T_1 = \neg T_1 \& X_1 \& \neg Z \# U \quad (15e)$$

$$T_2 = \neg T_2 \& Y_1 \& \neg Z' \# U \quad (15f)$$

$$T_3 = \neg T_3 \& Z \& \neg V \quad (15g)$$

$$T_4 = \neg T_4 \& Z' \& \neg V \quad (15h)$$

$$T_5 = \neg T_5 \& Z \& Z' \quad (15i)$$

$$\left. \begin{array}{l} X_1 = \varphi_1(X_1, \dots, X_m) \\ \vdots \\ X_m = \varphi_m(X_1, \dots, X_m) \end{array} \right\} \begin{array}{l} \text{The system for the language of all} \\ \text{accepting computations of } T. \end{array} \quad (15j)$$

$$\left. \begin{array}{l} Y_1 = \psi_1(Y_1, \dots, Y_n) \\ \vdots \\ Y_n = \psi_n(Y_1, \dots, Y_n) \end{array} \right\} \begin{array}{l} \text{The system for the language of all} \\ \text{rejecting computations of } T. \end{array} \quad (15k)$$

The equations (15c) and (15d) have unique solutions $(\Sigma \cup V \cup Q \cup \{\#\})^*$ and Σ^* respectively. Then the equations for T_i ($1 \leq i \leq 5$) implement the following inclusions using the method of (3):

$$X_1 \subseteq Z \# (\Sigma \cup V \cup Q \cup \{\#\})^* \quad (16a)$$

$$Y_1 \subseteq Z' \# (\Sigma \cup V \cup Q \cup \{\#\})^* \quad (16b)$$

$$Z \subseteq \Sigma^* \quad (16c)$$

$$Z' \subseteq \Sigma^* \quad (16d)$$

$$Z \cap Z' = \emptyset \quad (16e)$$

The last three inclusions state that Z and Z' should evaluate to disjoint subsets of Σ^* . (16a) means that every string accepted by T is in Z , while every string rejected by T is in Z' by (16b). If any string rejected by T would be in Z , then, as it is in Z' , it would be in $Z \cap Z'$, which would contradict (16e). Therefore, Z must coincide with $L(T)$, and the unique solution of the system is

$$(L(T), \Sigma^* \setminus L(T), (\Sigma \cup V \cup Q \cup \{\#\})^*, \Sigma^*, \emptyset, \emptyset, \emptyset, \emptyset, L'_1, \dots, L'_m, L''_1, \dots, L''_n), \quad (17)$$

where the first component is the given arbitrary recursive language. $\square$

Let us discuss the representability result of Theorem 5 in more detail. The construction (15) essentially means that for every recursive set L over an alphabet Σ there exists and can be effectively constructed an augmented alphabet

$\Sigma \cup \Gamma$ and a system of language equations over the this augmented alphabet, such that L is the first component of its unique solution.

What happens if the alphabet Σ is fixed and no auxiliary terminal symbols are allowed? If Σ has cardinality 2 or more, then the construction (15) can be modified to encode the auxiliary symbols as bit strings over Σ . The detailed construction is not included here; the improved statement of Theorem 5 is that for every alphabet Σ , such that $|\Sigma| \geq 2$, the set of languages representable by systems of language equations over Σ equals the set of recursive sets over Σ . Every unary recursive language can thus be denoted by adding one auxiliary symbol.

On the other hand, if $|\Sigma| = 1$ and no auxiliary symbols are allowed, then there seems to be no obvious way to reproduce the construction (15). The exact definition power of such systems is left as an open problem.

7 Conclusion

From the pure theoretical point of view, the basic issues regarding systems of language equations with Boolean operations have been solved. The technical results of this paper could be used as a basis for the study of more complicated properties of such systems, while the new characterization of recursive sets given by these systems might be interesting in itself.

However, if one considers using such systems of equations as a practical tool for denoting languages, everything is ruined by the enormous expressive power given by the peculiar type of context-dependency caused by the requirement of solution uniqueness. One has to invent some stronger conditions to impose on systems in order to limit their expressive power and make the membership problem computationally feasible; uniqueness of solution modulo every finite language could be one such condition.

References

1. J. Autebert, J. Berstel and L. Boasson, "Context-Free Languages and Pushdown Automata", *Handbook of Formal Languages*, Vol. 1, 111–174, Springer-Verlag, Berlin, 1997.
2. N. Immerman, *Descriptive complexity*, Springer-Verlag, New York, 1998.
3. W. Kuich, "Semirings and Formal Power Series: Their Relevance to Formal Language and Automata", *Handbook of Formal Languages*, Vol. 1, 609–677, Springer-Verlag, Berlin, 1997.
4. E. L. Leiss, *Language equations*, Springer-Verlag, New York, 1999.
5. A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.
6. A. Okhotin, "Conjunctive grammars and systems of language equations", *Programming and Computer Software*, 28:5 (2002) 243–249.
7. Ch. H. Papadimitriou, *Computational complexity*, Addison-Wesley, 1994.

Generalized Rewrite Theories^{*}

Roberto Bruni^{1,2} and José Meseguer²

¹ Dipartimento di Informatica, Università di Pisa, Italia.

² CS Department, University of Illinois at Urbana-Champaign, USA.

bruni@di.unipi.it, meseguer@cs.uiuc.edu

Abstract. Since its introduction, more than a decade ago, rewriting logic has attracted the interest of both theorists and practitioners, who have contributed in showing its generality as a semantic and logical framework and also as a programming paradigm. The experimentation conducted in these years has suggested that some significant extensions to the original definition of the logic would be very useful in practice. In particular, the Maude system now supports subsorting and conditions in the equational logic for data, and also frozen arguments to block undesired nested rewritings; moreover, it allows equality and membership assertions in rule conditions. In this paper, we give a detailed presentation of the inference rules, model theory, and completeness of such generalized rewrite theories.

Introduction

This paper develops new semantic foundations for a generalized version of rewriting logic. Since its original formulation [10], a substantial body of research (see the more than 300 references listed in the special TCS issue [6], and the four WRLA Proceedings in the ENTCS series, Vols. 4, 15, 36, and 71) has shown that rewriting logic (RL) has good properties as a *semantic framework*, particularly for concurrent and distributed computation, and also as a *logical framework*, a meta-logic in which other logics can be naturally represented. Indeed, the computational and logical meanings of a rewrite $t \rightarrow t'$ are like two sides of the same coin. Computationally, $t \rightarrow t'$ means that the state component t can *evolve* to the component t' . Logically, $t \rightarrow t'$ means that from the formula t one can *deduce* the formula t' . RL has also been shown to have good properties as a *declarative programming paradigm*, as demonstrated by the mature implementations of the ELAN [12], CafeOBJ [3], and Maude [2] languages.

The close contact with many applications in all the above areas has served as a good stimulus for a *substantial increase in expressive power* of the rewriting logic formalism by generalization along several dimensions:

1. Since a rewrite theory is essentially a triple $\mathcal{R} = (\Sigma, E, R)$, with (Σ, E) an equational theory, and R a set of labeled rewrite rules that are applied

^{*} Research supported by the MIUR Project COFIN 2001013518 CoMETA, by the FET-GC Project IST-2001-32747 AGILE, and by ONR Grant N00014-02-1-0715. The first author is also supported by a CNR fellowship for research on Information Sciences and Technologies.

modulo the equations E , it follows that rewriting logic is *parameterized by the choice of an underlying equational logic*; therefore, generalizations towards more expressive equational logics yield more expressive versions of rewriting logic.

2. Another dimension along which expressiveness can be increased is by allowing *more general conditions* in conditional rewrite rules.
3. Yet another dimension has to do with *forbidding rewriting under certain operators or operator positions* (frozen operators and arguments). Although this could be regarded as a purely *operational aspect*, the need for it in many applications suggests supporting it directly at the semantic level of rewrite theories.

In this paper we generalize rewrite theories along these three dimensions. Along dimension 1, we select *membership equational logic* (MEL) [11] as the underlying equational logic. This is a very expressive many-kinded Horn logic whose atomic formulas are equations $t = t'$ and memberships $t : s$. It contains as special cases the order-sorted, many-sorted, and unsorted versions of equational logic. Along dimension 2, assuming an underlying MEL theory (Σ, E) , we allow for conditional rewrite rules of the form,

$$(\forall X) r : t \rightarrow t' \text{ if } \bigwedge_{i \in I} p_i = q_i \wedge \bigwedge_{j \in J} w_j : s_j \wedge \bigwedge_{l \in L} t_l \rightarrow t'_l$$

where r is the rule label, all terms are Σ -terms, and the rule can be made conditional to other equations, memberships, and rewrites being satisfied. Finally, along dimension 3, we allow declaring certain operator arguments as *frozen*, thus blocking rewriting under them. This leads us to define a *generalized rewrite theory* (GRT) as a four tuple, $\mathcal{R} = (\Sigma, E, \phi, R)$, where (Σ, E) is a membership equational theory, R is a set of labeled conditional rewrite rules of the general form above, and ϕ is a function assigning to each operator $f : k_1 \dots k_n \rightarrow k$ in Σ the subset $\phi(f) \subseteq \{1, \dots, n\}$ of its frozen arguments.

As already mentioned, such a notion of generalized rewrite theory has been arrived at through a long and extensive contact with many applications. In fact, practice has gone somewhat ahead of theory: all the above generalizations have already been implemented in the latest alpha versions of Maude 2.0. The importance of generalizing rewrite theories along dimension 1 has to do with the greater expressiveness allowed by having sorts, subsorts, subsort overloaded operators, and partial functions; all this is further explained in Section 1.2. We can illustrate the importance of generalizing along dimensions 2 and 3 with an example showing that, in essence, this brings RL and *structural operational semantics* (whose strong relationship had already been emphasized in [5,7,8]) closer than ever before. Consider for example a reactive process calculus with a nondeterministic choice operator $+$ specified by SOS rules of the form,

$$\frac{P \rightarrow P'}{P + Q \rightarrow P'} \text{ left choice} \qquad \frac{Q \rightarrow Q'}{P + Q \rightarrow Q'} \text{ right choice}$$

The corresponding rewrite theory $\mathcal{R}$ will then have two conditional rules, like

$$\text{left choice} : P + Q \rightarrow P' \quad \text{if } P \rightarrow P' \qquad \text{right choice} : P + Q \rightarrow Q' \quad \text{if } Q \rightarrow Q'$$

Furthermore, both arguments of $+$ should be *frozen*, i.e., $\phi(+) = \{1, 2\}$. If we add to this process calculus a sequential composition $P; Q$, the fact that Q should not be able to evolve until P has finished its task can be straightforwardly modeled by declaring the second argument of $- ; -$ as frozen, plus the rule $\checkmark; Q \rightarrow Q$ (where $\checkmark$ is the “correct termination” process), which throws away the operator $- ; -$, unfreezing its second argument. Hence, (un)frozen arguments can naturally model *reactive contexts*, i.e., the distinguished set of environments where reactions can take place. Note that frozen arguments are for rewrite theories the analogous of the *strategy annotations* used for equational theories in OBJ, CafeOBJ, and Maude to improve efficiency and/or to guarantee the termination of computations, replacing unrestricted equational rewriting by so-called *context-sensitive rewriting* [4]. Thus, in Maude, rewriting with both equations E and rules R can be made context-sensitive. The usefulness of having frozen attributes in rewrite theories has emerged gradually. Stehr, Meseguer, and Ölveczky first proposed *frozen kinds* [13]. The generalization of this to a subset $\Omega \subseteq \Sigma$ of *frozen operators* emerged in a series of email exchanges between Stefani and the second author. The subsequent generalization of freezing operator arguments selectively brings us to the just mentioned two levels (for equations and for rules) of context-sensitive rewriting.

Given the above notion of GRT, the paper addresses the following questions:

- What are rewriting logic’s *rules of deduction* for generalized rewrite theories?
- What are the *models* of a rewrite theory? Are there initial and free models?
- Is rewriting logic *complete* with respect to its model theory, so that a rewrite is provable from a rewrite theory $\mathcal{R}$ if and only if it is satisfied by all models of $\mathcal{R}$?

The answers given (all in the affirmative) are in fact nontrivial *generalizations* of the original inference rules, model theory, initial and free models, and completeness theorem for rewriting logic over unsorted equational logic, as developed in [10]. In summary, therefore, this paper develops new *semantic foundations* for a generalized version of rewriting logic, along several dimensions that have been found to substantially increase its expressiveness in concrete applications. At the programming language level, this paper does also provide the needed mathematical semantics for Maude 2.0.

Synopsis. In § 1.1 we recap from [10] the original presentation of RL, and in § 1.2 we overview membership equational logic. § 2 and § 3 present the original contributions of the paper, introducing generalized rewrite theories, their proof theory, their model theory, and the completeness results. Note that the algebras of reachability and decorated sequents are expressed as membership equational theories themselves (a framework not available when [10] was published). Conclusions are drawn in the last section.

1 Background

1.1 Conditional Rewriting Logic

Though in the rewriting community it is folklore that rewrite theories are parametric w.r.t. the underlying equational logic of data specification, the details have been fully spelled out only for unsorted equational logic, and rules of the form (1) below.

Since only unsorted theories were treated in [10], here, but not in the rest of the paper where ordered sorts are used, an (equational) *signature* is a family of sets of *function symbols* (also *operators*) $\Sigma = \{\Sigma_n\}_{n \in \mathbb{N}}$ indexed by arities n , and a *theory* is a pair (Σ, E) where $E = \{(\forall X_i) t_i = t'_i\}_{1 \leq i \leq m}$ is a set of (universally quantified) Σ -equations, with $t_i, t'_i \in \mathbb{T}_\Sigma(X_i)$ two Σ -terms with variables in X_i . We let $t =_E t'$ denote the congruence modulo E of two terms t, t' and let $[t]_E$ or just $[t]$ denote the E -equivalence class of t modulo E . We shall denote by $t[u_1/x_1, \dots, u_n/x_n]$ (abbreviated $t[\vec{u}/\vec{x}]$) the term obtained from t by simultaneously replacing the occurrences of x_i by u_i for $1 \leq i \leq n$.

Definition 1.1 (Conditional rewrite theory). A (labeled) conditional rewrite theory $\mathcal{R}$ is a tuple $\mathcal{R} = (\Sigma, E, R)$, where (Σ, E) is an unsorted equational theory and R is a set of (labeled) conditional rewrite rules having the form below, with $t, t', t_i, t'_i \in \mathbb{T}_\Sigma(X)$.

$$(\forall X) r : t \rightarrow t' \quad \text{if} \quad t_1 \rightarrow t'_1 \wedge \dots \wedge t_\ell \rightarrow t'_\ell. \quad (1)$$

The theory (Σ, E) defines the static data structure for the states of the system (e.g., a free monoid for strings, or a free commutative monoid for multisets), while R defines the dynamics (e.g., productions in phrase-structure grammars or transitions in Petri nets).

Given a rewrite theory $\mathcal{R}$, its *rewriting logic* is a sequent calculus whose sentences have the form $(\forall X) t \rightarrow t'$ (with the dual, logico-computational meaning explained in the Introduction). We say that $\mathcal{R}$ *entails* a sequent $(\forall X) t \rightarrow t'$, and write $\mathcal{R} \vdash (\forall X) t \rightarrow t'$, if $(\forall X) t \rightarrow t'$ can be obtained by means of the inference rules in Figure 1. Roughly, **(Reflexivity)** introduces idle computations, **(Transitivity)** expresses the sequential composition of rewrites, **(Equality)** means that rewrites are applied modulo the equational theory E , **(Congruence)** says that rewrites can be nested inside larger contexts. The most complex rule is **(Nested Replacement)**, stating that given a rewrite rule $r \in R$ and two substitutions θ, θ' for its variables such that for each $x \in X$ we have $\theta(x) \rightarrow \theta'(x)$, then r can be concurrently applied to the rewrites of its arguments, once that the conditions of r can be satisfied in the initial state defined by θ . Since rewrites are applied modulo E , the sequents can be equivalently written $(\forall X) [t] \rightarrow [t']$.

From the model-theoretic viewpoint, the sequents can be decorated with *proof terms* in a suitable algebra that exactly captures concurrent computations. We remark that each rewrite theory $\mathcal{R}$ has initial and free models and that a completeness theorem reconciles the proof theory and the model theory, stating

$$\begin{array}{c}
\frac{t \in \mathbb{T}_\Sigma(X)}{(\forall X) t \rightarrow t} \text{ Reflexivity} \qquad \frac{(\forall X) t_1 \rightarrow t_2, \quad (\forall X) t_2 \rightarrow t_3}{(\forall X) t_1 \rightarrow t_3} \text{ Transitivity} \\
\\
\frac{E \vdash (\forall X) t = u, \quad (\forall X) u \rightarrow u', \quad E \vdash (\forall X) u' = t'}{(\forall X) t \rightarrow t'} \text{ Equality} \\
\\
\frac{f \in \Sigma_n, \quad (\forall X) t_i \rightarrow t'_i \text{ for } i \in [1, n]}{(\forall X) f(t_1, \dots, t_n) \rightarrow f(t'_1, \dots, t'_n)} \text{ Congruence} \\
\\
\frac{(\forall X) r: t \rightarrow t' \text{ if } \bigwedge_{1 \leq i \leq \ell} t_i \rightarrow t'_i \in R, \quad \theta, \theta': X \rightarrow \mathbb{T}_\Sigma(Y) \quad (\forall Y) \theta(t_i) \rightarrow \theta(t'_i) \text{ for } 1 \leq i \leq \ell, \quad (\forall Y) \theta(x) \rightarrow \theta'(x) \text{ for } x \in X}{(\forall Y) \theta(t) \rightarrow \theta'(t')} \text{ Nested Replacement}
\end{array}$$

Fig. 1. Deduction rules for conditional rewrite theories.

that a sequent is provable from $\mathcal{R}$ if and only if it is satisfied in all models of $\mathcal{R}$ (called $\mathcal{R}$ -systems).

Roughly, the algebra of sequents contains the terms $[t]$ in $\mathbb{T}_{\Sigma, E}$ for idle rewrites, with the operators and equations in (Σ, E) lifted to the level of sequents (e.g., if $\alpha_i: [t_i] \rightarrow [t'_i]$ for $i \in [1, n]$, then $f(\alpha_1, \dots, \alpha_n): [f(t_1, \dots, t_n)] \rightarrow [f(t'_1, \dots, t'_n)]$), plus the concatenation operator $_-$; $_-$ for composing $\alpha_1: [t_1] \rightarrow [t_2]$ and $\alpha_2: [t_2] \rightarrow [t_3]$ to $\alpha_1; \alpha_2: [t_1] \rightarrow [t_3]$ via **(Transitivity)**, and finally an additional operator r with arity $|X| + \ell$ for each rule $r \in R$ of the form (1). For example, if $\{\beta_i: [\theta(t_i)] \rightarrow [\theta(t'_i)]\}_{1 \leq i \leq \ell}$ and $\{\alpha_x: [\theta(x)] \rightarrow [\theta'(x)]\}_{x \in X}$ are used as premises in **(Nested Replacement)**, then the conclusion is decorated by $r(\vec{\alpha}, \vec{\beta})$. The axioms express: (i) that sequents form the arrows of a category with $_-$; $_-$ as composition and idle rewrites $[t]$ as identities; (ii) the functoriality of the (Σ, E) -structure, and (iii) the so-called *decomposition* and *exchange* laws, saying that the application of r to $[\theta(t)]$ is concurrent w.r.t. the rewrites of the arguments of t .

1.2 Membership Equational Logic

In many applications, unsorted signatures are not expressive enough to reflect in a natural way the features of the system to be modeled. The expressiveness can be increased by supporting sorts (e.g., **Bool**, **Nat**, **Int**) via *many-sorted* signatures and relating them via *order-sorted* signatures (e.g., **NzNat** $<$ **Nat** $<$ **Int**). Equations in E can be made more expressive by allowing *conditions* for their applications. Such conditions can be other equalities, or membership assertions. Conditional membership assertions are also useful. *Membership equational logic* (MEL) [11] possesses all the above features (generalizing order-sorted equational logic) and is supported by Maude [2].

A MEL *signature* is a triple (K, Σ, S) (just Σ in the following), with K a set of *kinds*, $\Sigma = \{\Sigma_{\delta,k}\}_{(\delta,k) \in K^* \times K}$ a many-kinded signature and $S = \{S_k\}_{k \in K}$ a K -kinded family of disjoint sets of sorts. The kind of a sort s is denoted by $[s]$. A MEL Σ -algebra A contains a set A_k for each kind $k \in K$, a function $A_f: A_{k_1} \times \cdots \times A_{k_n} \rightarrow A_k$ for each operator $f \in \Sigma_{k_1 \dots k_n, k}$ and a subset $A_s \subseteq A_k$ for each sort $s \in S_k$, with the meaning that the elements in sorts are well-defined, while elements without a sort are *errors*. We write $\mathbb{T}_{\Sigma,k}$ and $\mathbb{T}_{\Sigma}(X)_k$ to denote respectively the set of ground Σ -terms with kind k and of Σ -terms with kind k over variables in X , where $X = \{x_1 : k_1, \dots, x_n : k_n\}$ is a set of kinded variables.

Given a MEL signature Σ , *atomic formulae* have either the form $t = t'$ (Σ -equation) or $t : s$ (Σ -membership) with $t, t' \in \mathbb{T}_{\Sigma}(X)_k$ and $s \in S_k$; and Σ -sentences are conditional formulae of the form $(\forall X) \varphi$ if $\bigwedge_i p_i = q_i \wedge \bigwedge_j w_j : s_j$, where φ is either a Σ -equation or a Σ -membership and all the variables in φ , p_i , q_i , and w_j are in X . A MEL theory is a pair (Σ, E) with Σ a MEL signature and E a set of Σ -sentences. We refer to [11] for the detailed presentation of (Σ, E) -algebras, sound and complete deduction rules, initial and free algebras, and theory morphisms.

Order-sorted notation $s_1 < s_2$ can be used to abbreviate the conditional membership $(\forall x : k) x : s_2$ if $x : s_1$. Similarly, an operator declaration $f: s_1 \times \cdots \times s_n \rightarrow s$ corresponds to declaring f at the kind level and giving the membership axiom $(\forall x_1 : k_1, \dots, x_n : k_n) f(x_1, \dots, x_n) : s$ if $\bigwedge_{1 \leq i \leq n} x_i : s_i$. We write $(\forall x_1 : s_1, \dots, x_n : s_n) t = t'$ in place of $(\forall x_1 : k_1, \dots, x_n : k_n) t = t'$ if $\bigwedge_{1 \leq i \leq n} x_i : s_i$. Moreover, for a list of variables of the same sort s , we write $(\forall x_1, \dots, x_n : s)$, and let the sentence $(\forall X) t : k$ mean $t \in \mathbb{T}_{(\Sigma, E)}(X)_k$.

2 Generalized Rewrite Theories and Deduction

In this section we present the foundations of rewrite theories over MEL theories and where operators can have frozen arguments.

A *generalized operator* is a function symbol $f: k_1 \cdots k_n \rightarrow k$ together with a set $\phi(f) \subseteq \{1, \dots, n\}$ of frozen argument positions. We denote by $\nu(f)$ the set $\{1, \dots, n\} \setminus \phi(f)$ of *unfrozen* arguments, and say that f is *unfrozen* if $\phi(f) = \emptyset$.

Definition 2.1 (Generalized signatures). A generalized MEL signature (Σ, ϕ) is a MEL signature Σ whose function symbols are generalized operators. The function $\phi: \Sigma \rightarrow \wp_f(\mathbb{N})$ assigns to each $f \in \Sigma$ its set of frozen arguments ($\wp_f(\mathbb{N})$ denotes the set of finite sets of natural numbers and for any $f: k_1 \cdots k_n \rightarrow k$ in Σ we assume $\phi(f) \subseteq \{1, \dots, n\}$).

If the i th position of f is frozen, then in $f(t_1, \dots, t_n)$ any subterm of t_i is frozen. This can be made formal by considering the usual tree-like representation of terms (the same subterm can occur in many distinct positions that are not necessarily all frozen). Positions in a term are denoted by strings of natural numbers, indicating the sequences of branches we must follow from the root to reach that position. For example, the term $t = f(g(a, b, c), f(h(a, b), f(b, c)))$ has two occurrences of the constant c at positions 1.3 and 2.2.2, respectively. We let

t_π and $t(\pi)$ denote, respectively, the subterm of t occurring at position π , and its topmost operator. For λ the empty position, we let t_λ denote the whole term t . In the example above, we have $t_{2.1} = h(a, b)$ and $t(2.1) = h$.

Definition 2.2 (Frozen occurrences). *The occurrence t_π of the subterm of t at position π is frozen if there exist two positions π_1, π_2 and a natural number n such that $\pi = \pi_1.n.\pi_2$ and $n \in \phi(t(\pi_1))$. The occurrence t_π is called unfrozen if it is not frozen.*

In the example above, for $\phi(f) = \phi(g) = \emptyset$ and $\phi(h) = \{1\}$, we have that $t_{2.1.1} = a$ is frozen (because $t(2.1) = h$), while $t_{1.1} = a$ is unfrozen (because $t(\lambda) = f$ and $t(1) = g$).

Definition 2.3 (Frozen variables). *Given $t \in \mathbb{T}_\Sigma(X)$ we say that the variable $x \in X$ is frozen in t if there exists a frozen occurrence of x in t , otherwise it is called unfrozen.*

We let $\phi(t)$ and $\nu(t)$ denote, respectively, the set of frozen and unfrozen variables of t . Analogously, $\phi(t_1, \dots, t_n)$ (resp. $\nu(t_1, \dots, t_n)$) denotes the set of variables for which a frozen occurrence appears in at least one t_i (resp. that are unfrozen in all t_i).

By combining conditional rewrite theories with MEL specifications and frozen arguments, we obtain a rather general notion of rewrite theory.

Definition 2.4 (Generalized rewrite theory). *A generalized rewrite theory (GRT) is a tuple $\mathcal{R} = (\Sigma, E, \phi, R)$ consisting of: (i) a generalized MEL signature (Σ, ϕ) with say kinds $k \in K$, sorts $s \in S$, and $K^* \times K$ -indexed set of generalized operators $f \in \Sigma$ with frozen arguments according to ϕ ; (ii) a MEL theory (Σ, E) ; (iii) a set R of (universally quantified) labeled conditional rewrite rules r having the general form*

$$(\forall X) \ r: t \rightarrow t' \text{ if } \bigwedge_{i \in I} p_i = q_i \ \wedge \ \bigwedge_{j \in J} w_j : s_j \ \wedge \ \bigwedge_{l \in L} t_l \rightarrow t'_l \quad (2)$$

where, for appropriate kinds k and k_l in K , $t, t' \in \mathbb{T}_\Sigma(X)_k$ and $t_l, t'_l \in \mathbb{T}_\Sigma(X)_{k_l}$ for $l \in L$.

2.1 Inference in Generalized Rewriting Logic

Given a GRT $\mathcal{R} = (\Sigma, E, \phi, R)$, a *sequent* of $\mathcal{R}$ is a pair of (universally quantified) terms of the same kind t, t' , denoted $(\forall X) t \rightarrow t'$ with $X = \{x_1 : k_1, \dots, x_n : k_n\}$ a set of kinded variables and $t, t' \in \mathbb{T}_\Sigma(X)_k$ for some k . We say that $\mathcal{R}$ *entails* the sequent $(\forall X) t \rightarrow t'$, and write $\mathcal{R} \vdash (\forall X) t \rightarrow t'$, if the sequent $(\forall X) t \rightarrow t'$ can be obtained by means of the inference rules in Figure 2, which are briefly described below.

(Reflexivity), **(Transitivity)**, and **(Equality)** are the usual rules for idle rewrites, concatenation of rewrites, and rewriting modulo the MEL theory E . **(Congruence)** allows rewriting the arguments of a generalized operator, but

$$\begin{array}{c}
\frac{t \in \mathbb{T}_\Sigma(X)_k}{(\forall X) t \rightarrow t} \text{ Reflexivity} \qquad \frac{(\forall X) t_1 \rightarrow t_2, \quad (\forall X) t_2 \rightarrow t_3}{(\forall X) t_1 \rightarrow t_3} \text{ Transitivity} \\
\\
\frac{E \vdash (\forall X) t = u, \quad (\forall X) u \rightarrow u', \quad E \vdash (\forall X) u' = t'}{(\forall X) t \rightarrow t'} \text{ Equality} \\
\\
\frac{\begin{array}{l} f \in \Sigma_{k_1 \dots k_n, k}, \quad t_i, t'_i \in \mathbb{T}_\Sigma(X)_{k_i} \text{ for } i \in [1, n] \\ t'_i = t_i \text{ for } i \in \phi(f), \quad (\forall X) t_j \rightarrow t'_j \text{ for } j \in \nu(f) \end{array}}{(\forall X) f(t_1, \dots, t_n) \rightarrow f(t'_1, \dots, t'_n)} \text{ Congruence} \\
\\
\frac{\begin{array}{l} (\forall X) r: t \rightarrow t' \text{ if } \bigwedge_{i \in I} p_i = q_i \wedge \bigwedge_{j \in J} w_j : s_j \wedge \bigwedge_{l \in L} t_l \rightarrow t'_l \in R \\ \theta, \theta': X \rightarrow \mathbb{T}_\Sigma(Y), \quad \theta(x) = \theta'(x) \text{ for } x \in \phi(t, t') \\ E \vdash (\forall Y) \theta(p_i) = \theta(q_i) \text{ for } i \in I, \quad E \vdash (\forall Y) \theta(w_j) : s_j \text{ for } j \in J \\ (\forall Y) \theta(t_l) \rightarrow \theta(t'_l) \text{ for } l \in L, \quad (\forall Y) \theta(x) \rightarrow \theta'(x) \text{ for } x \in \nu(t, t') \end{array}}{(\forall Y) \theta(t) \rightarrow \theta'(t')} \text{ Nested Replacement}
\end{array}$$

Fig. 2. Deduction rules for generalized rewrite theories.

we add the condition that frozen arguments must stay idle (note that $t'_i = t_i$ is syntactic equality). Any unfrozen argument can still be rewritten, as expressed by the premise $(\forall X) t_j \rightarrow t'_j$ for $j \in \nu(f)$.

(Nested Replacement) takes into account the application of a rewrite rule in its most general form (2). It specifies that for any rewrite rule $r \in R$ and for any (kind-preserving) substitution θ such that the condition of r is satisfied when θ is applied to all terms p_i, q_i, w_j, t_l, t'_l involved, then it is possible to apply the rewrite r to $\theta(t)$. Moreover, if θ' is a second (kind-preserving) substitution for the variables in X such that θ and θ' coincide on all frozen variables $x \in \phi(t, t')$ (second line of premises), while the rewrites $(\forall Y) \theta(x) \rightarrow \theta'(x)$ are provable for the unfrozen variables $x \in \nu(t, t')$ (last premise), then such nested rewrites can be applied concurrently with r .

Of course, any unsorted rewrite theory can be regarded as a GRT where: (i) Σ has a unique kind and no sorts; (ii) all the operators are total and unfrozen (i.e., $\phi(f) = \emptyset$ for any $f \in \Sigma$); (iii) conditions in rewrite rules contain neither equalities nor membership predicates. In this case, deduction via rules for conditional rewrite theories (Figure 1) coincides with deduction via rules for generalized rewrite theories (Figure 2).

Theorem 2.1. *Let $\mathcal{R}$ be a conditional rewrite theory, and let $\hat{\mathcal{R}}$ denote its corresponding GRT. Then:* $\mathcal{R} \vdash (\forall X) t \rightarrow t' \Leftrightarrow \hat{\mathcal{R}} \vdash (\forall X) t \rightarrow t'.$

3 Models of Generalized Rewrite Theories

In this section, exploiting MEL, we define the reachability and concurrent model theories of GRTs and state completeness results.

3.1 Reachability Models

Reachability models focus just on *what* terms/states can be reached from a certain state t via sequences of rewrites, ignoring *how* the rewrites can lead to them.

Definition 3.1 (Reachability relation). *Given a GRT $\mathcal{R} = (\Sigma, E, \phi, R)$, its reachability relation $\rightarrow_{\mathcal{R}}$, is defined proof-theoretically, for each kind k in Σ and each $[t], [t'] \in \mathbb{T}_{\Sigma, E}(X)_k$, by the equivalence: $[t] \rightarrow_{\mathcal{R}} [t'] \Leftrightarrow \mathcal{R} \vdash (\forall X) t \longrightarrow t'$.*

The above definition is sound because we have the following easy lemma.

Lemma 3.1. *Let $\mathcal{R} = (\Sigma, E, \phi, R)$ be a GRT, and $t \in \mathbb{T}_{\Sigma}(X)_k$. If $\mathcal{R} \vdash (\forall X) t \longrightarrow t'$, then $t' \in \mathbb{T}_{\Sigma}(X)_k$. Moreover, for any $t, u, u', t' \in \mathbb{T}_{\Sigma}(X)_k$ such that $u \in [t]_E$, $u' \in [t']_E$ and $\mathcal{R} \vdash (\forall X) u \longrightarrow u'$, then $\mathcal{R} \vdash (\forall X) t \longrightarrow t'$.*

The reachability relation admits a model-theoretic presentation in terms of the free models of a suitable MEL theory. We give the details below as a “warm up” for the model-theoretic concurrent semantics given in the next section. The idea is that $\rightarrow_{\mathcal{R}}$ can be defined as the family of relations, indexed by the kinds k , given by interpreting the sorts Ar_k in the free model of the following MEL theory $Reach(\mathcal{R})$.

Definition 3.2 (The theory $Reach(\mathcal{R})$). *The membership equational theory $Reach(\mathcal{R})$ contains the signature and sentences in (Σ, E) together with the following extensions:*

1. *For each kind k in Σ we add:*
 - a) *a new kind $[Pair_k]$ (for k -indexed binary relations on terms of kind k) with four sorts Ar_k^0 , Ar_k^1 , Ar_k and $Pair_k$ and subsort inclusions: $Ar_k^0, Ar_k^1 < Ar_k < Pair_k$;*
 - b) *the operators $(\_ \rightarrow \_) : k\ k \longrightarrow Pair_k$ (pair constructor), $s, t : Pair_k \longrightarrow k$ (source and target projections), and $(\_ ; \_) : [Pair_k] [Pair_k] \longrightarrow [Pair_k]$ (concatenation);*
 - c) *the (conditional) equations and memberships*

$$\begin{aligned}
 &(\forall x, y : k) \ s(x \rightarrow y) = x \\
 &(\forall x, y : k) \ t(x \rightarrow y) = y \\
 &(\forall z : Pair_k) \ (s(z) \rightarrow t(z)) = z \\
 &(\forall x : k) \ (x \rightarrow x) : Ar_k^0 \\
 &(\forall x, y, z : k) \ (x \rightarrow z) : Ar_k \quad \text{if} \quad (x \rightarrow y) : Ar_k \wedge (y \rightarrow z) : Ar_k \\
 &(\forall x, y, z : k) \ (x \rightarrow y); (y \rightarrow z) = (x \rightarrow z).
 \end{aligned}$$

2. Each $f : k_1 \dots k_n \rightarrow k$ in Σ with $\nu(f) \neq \emptyset$ is lifted to $f : [Pair_{k_1}] \dots [Pair_{k_n}] \rightarrow [Pair_k]$, and for each $i \in \nu(f)$ we declare $f : Ar_{k_1}^0 \dots Ar_{k_i}^1 \dots Ar_{k_n}^0 \rightarrow Ar_k^1$; we then give, for each $i \in \nu(f)$, the equation below, where $X_i = \{x_1 : k_1, \dots, x_n : k_n, y_i : k_i\}$

$$(\forall X_i) f((x_1 \rightarrow x_1), \dots, (x_i \rightarrow y_i), \dots, (x_n \rightarrow x_n)) = f(x_1, \dots, x_n) \rightarrow f(x_1, \dots, y_i, \dots, x_n).$$

3. For each rule $(\forall X) r : t \rightarrow t'$ if $\bigwedge_{i \in I} p_i = q_i \wedge \bigwedge_{j \in J} w_j : s_j \wedge \bigwedge_{l \in L} t_l \rightarrow t'_l$ in R , with, say t, t' of kind k , and t_l, t'_l of kind k_l , we give the conditional membership,

$$(\forall X) (t \rightarrow t') : Ar_k^1 \text{ if } \bigwedge_{i \in I} p_i = q_i \wedge \bigwedge_{j \in J} w_j : s_j \wedge \bigwedge_{l \in L} t_l \rightarrow t'_l : Ar_{k_l}.$$

The sorts Ar_k^0 and Ar_k^1 contain respectively idle rewrites and one-step rewrites of k -kinded terms, while the sort Ar_k contains k -rewrites of arbitrary length. The **(Congruence)** rule is modeled so that exactly one unfrozen argument can be rewritten in one-step (see item 2 in Definition 3.2), and **(Nested Replacement)** is restricted so that no nested rewrites can take place concurrently (item 3). Nevertheless, these two restrictions on how the inference rules are modeled do not alter the reachability relation Ar_k , because one-step rewrites can be composed in any admissible interleaved fashion (see the fifth axiom at point 1.(c)). Note that the concatenation operator $_ ; _$ is not really necessary, but its introduction facilitates the proof of Theorem 3.2.

The theory $Reach(\mathcal{R})$ provides an algebraic model for the reachability relation. For ground terms, such a model is given by the interpretation of the sorts Ar_k in the initial model $\mathbb{T}_{Reach(\mathcal{R})}$. For terms with variables in X , the reachability model is the free algebra $\mathbb{T}_{Reach(\mathcal{R})}(X)$. This can be summarized by the following theorem:

Theorem 3.1. *For $\mathcal{R} = (\Sigma, E, \phi, R)$ a GRT and $t, t' \in T_\Sigma(X)_k$ we have the equivalences:*

$$\begin{aligned} \mathcal{R} \vdash (\forall X) t \rightarrow t' &\Leftrightarrow Reach(\mathcal{R}) \vdash (\forall X) (t \rightarrow t') : Ar_k \\ &\Leftrightarrow Reach(\mathcal{R}) \models (\forall X) (t \rightarrow t') : Ar_k \\ &\Leftrightarrow [(t \rightarrow t')] \in \mathbb{T}_{Reach(\mathcal{R})}(X)_{Ar_k}. \end{aligned}$$

3.2 Concurrent Models

In general, many proofs concluding that $\mathcal{R} \vdash (\forall X) t \rightarrow t'$ are possible. However: (1) some of the proofs can be computationally equivalent, because they represent different interleaved sequences for the same concurrent computation, but (2) not all those proofs are necessarily equivalent, as they may, e.g., differ in the underlying set of applied rewrite rules, or in the different causal connections between the applications of the same rules. In this section, we show how to extend the notion of decorated sequents to GRTs, so as to define an algebraic model of *true concurrency* for $\mathcal{R}$.

As usual, decorated sequents are first defined by attaching a *proof term* (i.e., an expression built from variables, operators in Σ , and labels in R) to each sequent, and then by quotienting out proof terms modulo suitable functoriality, decomposition, and exchange laws. We can present $\mathcal{R}$'s algebra of sequents as the initial (or free) algebra of a suitable MEL theory $\text{Proof}(\mathcal{R})$. With respect to the classical presentation via decorated deduction rules, the MEL specification allows a standard algebraic definition of initial and loose semantics. Moreover, here we can naturally support many-sorted, order-sorted, and MEL data theories instead of just unsorted equational theories as in [10].

The construction of $\text{Proof}(\mathcal{R})$ is analogous to that of $\text{Reach}(\mathcal{R})$. The kind $[Pair_k]$ of $\text{Reach}(\mathcal{R})$ is replaced here by a kind $[Rw_k]$, whose elements include the proofs of concurrent computations. The initial and final states are still defined by means of the source (**s**) and target (**t**) operators. Moreover, since the proof of an idle rewrite $[t] \rightarrow [t]$ is $[t]$ itself, we can exploit subsorting to make k a sort of kind $[Rw_k]$. The sorts Rw_k^1 and Rw_k are the analogues of Ar_k^1 and Ar_k . The sort Ar_k^1 was introduced in $\text{Reach}(\mathcal{R})$ to deal with the “restricted” form of **(Congruence)** and **(Nested Replacement)**. Having decorations at hand, we can restore the full expressiveness of the two inference rules, but the sort Rw_k^1 is still useful in axiomatizing proof-decorated sequents: we define the **(Equality)** rule on Rw_k^1 , lifting the equational theory E to one-step rewrites, and then exploit functoriality and transitivity to extend E to rewrites of arbitrary length in Rw_k .

Definition 3.3 (The theory $\text{Proof}(\mathcal{R})$). *The membership equational theory $\text{Proof}(\mathcal{R})$ contains the signature and sentences of (Σ, E) together with the following extensions:*

1. Each kind k in Σ becomes a sort k in $\text{Proof}(\mathcal{R})$, with $s < k$ for any $s \in S_k$ in Σ .
2. For each kind k in Σ we add:
 - a) a new kind $[Rw_k]$ (for k -indexed decorated rewrites on Σ -terms of kind k) with sorts all sorts in k and the (new) sorts k , Rw_k^1 and Rw_k , with: $k \leq Rw_k^1 < Rw_k$;
 - b) the (overloaded) operators $(- ; -) : [Rw_k] [Rw_k] \longrightarrow [Rw_k]$ and $s, t : Rw_k \longrightarrow k$;
 - c) the (conditional) equations and memberships

$$\begin{aligned}
 &(\forall x : k) \, s(x) = x \\
 &(\forall x : k) \, t(x) = x \\
 &(\forall x, y : Rw_k) \, x; y : Rw_k \quad \text{if} \quad t(x) = s(y) \\
 &(\forall x, y : Rw_k) \, s(x; y) = s(x) \quad \text{if} \quad t(x) = s(y) \\
 &(\forall x : Rw_k, y : Rw_k) \, t(x; y) = t(y) \quad \text{if} \quad t(x) = s(y) \\
 &(\forall x : k, y : Rw_k) \, x; y = y \quad \text{if} \quad x = s(y) \\
 &(\forall x : Rw_k, y : k) \, x; y = x \quad \text{if} \quad t(x) = y \\
 &(\forall x, y, z : Rw_k) \, x; (y; z) = (x; y); z \quad \text{if} \quad t(x) = s(y) \wedge t(y) = s(z).
 \end{aligned}$$

3. We lift each operator $f : k_1 \dots k_n \longrightarrow k$ in Σ to $f : [Rw_{k_1}] \dots [Rw_{k_n}] \longrightarrow [Rw_k]$, and for $\nu(f) = \{i_1, \dots, i_m\}$ we overload f by $f : k_1 \dots Rw_{k_{i_1}} \dots Rw_{k_{i_m}} \dots k_n \longrightarrow Rw_k$ and $f : k_1 \dots Rw_{k_{i_j}}^1 \dots k_n \longrightarrow Rw_k^1$ for $j = 1, \dots, m$, with equations

$$\begin{aligned} (\forall X) \mathbf{s}(f(x_1, \dots, x_n)) &= f(\mathbf{s}(x_1), \dots, \mathbf{s}(x_n)) \\ (\forall X) \mathbf{t}(f(x_1, \dots, x_n)) &= f(\mathbf{t}(x_1), \dots, \mathbf{t}(x_n)), \end{aligned}$$

where $X = \{x_1 : k_1, \dots, x_{i_1} : Rw_{k_{i_1}}, \dots, x_{i_m} : Rw_{k_{i_m}}, \dots, x_n : k_n\}$ and the equation

$$(\forall Y) f(x_1, \dots, (x_{i_1}; y_{i_1}), \dots, (x_{i_m}; y_{i_m}), \dots, x_n) = f(x_1, \dots, x_n; f(x_1, \dots, y_{i_1}, \dots, y_{i_m}, \dots, x_n) \text{ if } \bigwedge_{1 \leq j \leq m} \mathbf{t}(x_{i_j}) = \mathbf{s}(y_{i_j}),$$

where $Y = \{x_1 : k_1, \dots, x_{i_1}, y_{i_1} : Rw_{k_{i_1}}, \dots, x_{i_m}, y_{i_m} : Rw_{k_{i_m}}, \dots, x_n : k_n\}$.

4. For each equation $(\forall x_1 : k_1, \dots, x_n : k_n) t = t'$ if $\bigwedge_{i \in I} p_i = q_i \wedge \bigwedge_{j \in J} w_j : s_j$ in E , we let $X = \{x_1 : Rw_{k_1}, \dots, x_n : Rw_{k_n}\}$ and add the conditional equation

$$(\forall X) t = t' \text{ if } \bigwedge_{i \in I} p_i = q_i \wedge \bigwedge_{j \in J} \mathbf{s}(w_j) : s_j \wedge \bigwedge_{j \in J} \mathbf{t}(w_j) : s_j \wedge \bigwedge_{x_h \in \phi(t, t')} x_h : k_h \wedge \bigwedge_{x_h \in \nu(t, t')} x_h : Rw_{k_h}^1.$$

5. For each rule $(\forall X) r : t \rightarrow t'$ if $\bigwedge_{i \in I} p_i = q_i \wedge \bigwedge_{j \in J} w_j : s_j \wedge \bigwedge_{l \in L} t_l \rightarrow t'_l$ in R , with, say, $X = \{x_1 : k_1, \dots, x_n : k_n\}$, t, t' of kind k , and t_l, t'_l of kind k'_l with $L = \{1, \dots, \ell\}$, we add the operator

$$r : [Rw_{k_1}] \dots [Rw_{k_n}] [Rw_{k'_1}] \dots [Rw_{k'_\ell}] \rightarrow [Rw_k]$$

with

- a) the conditional membership for characterizing basic one-step rewrites:

$$(\forall x_1 : k_1, \dots, x_n : k_n, y_1 : Rw_{k'_1}, \dots, y_\ell : Rw_{k'_\ell}) r(\vec{x}, \vec{y}) : Rw_k^1 \text{ if } \Delta$$

where $\Delta = (\bigwedge_{i \in I} p_i = q_i \wedge \bigwedge_{j \in J} w_j : s_j \wedge \bigwedge_{l \in L} \mathbf{s}(y_l) = t_l \wedge \bigwedge_{l \in L} \mathbf{t}(y_l) = t'_l)$ checks that the conditions for the application of the rule r are satisfied;

- b) the conditional equations and memberships

$$\begin{aligned} (\forall Y) r(\vec{z}, \vec{y}) : Rw_k &\text{ if } \Delta \wedge \Psi \\ (\forall Y) \mathbf{s}(r(\vec{z}, \vec{y})) &= t \text{ if } \Delta \wedge \Psi \\ (\forall Y) \mathbf{t}(r(\vec{z}, \vec{y})) &= t'[\mathbf{t}(\vec{z})/\vec{x}] \text{ if } \Delta \wedge \Psi \end{aligned}$$

where $Y = \{x_1 : k_1, \dots, x_n : k_n, z_1 : Rw_{k_1}, \dots, z_n : Rw_{k_n}, y_1 : Rw_{k'_1}, \dots, y_\ell : Rw_{k'_\ell}\}$, Δ is as before, and $\Psi = (\bigwedge_{x_h \in \phi(t, t')} z_h = x_h \wedge \bigwedge_{x_h \in \nu(t, t')} \mathbf{s}(z_h) = x_h)$;

- c) the decomposition law

$$(\forall Z) r(\vec{z}, \vec{y}) = r(\vec{x}, \vec{y}); t'[\vec{z}/\vec{x}] \text{ if } \Delta \wedge \Psi$$

where $Z = \{x_1 : k_1, \dots, x_n : k_n, z_1 : Rw_{k_1}, \dots, z_n : Rw_{k_n}, y_1 : Rw_{k'_1}, \dots, y_\ell : Rw_{k'_\ell}\}$, while Δ and Ψ are as before;

d) the exchange law

$$(\forall W) \ r(\vec{x}, \vec{y}); t'[\vec{z}/\vec{x}] = t[\vec{z}/\vec{x}]; r(\mathbf{t}(\vec{z}), \vec{y}') \quad \text{if} \quad \Delta \wedge \Psi \wedge \Delta' \wedge \Phi$$

where $W = \{x_1 : k_1, \dots, x_n : k_n, z_1 : Rw_{k_1}^1, \dots, z_n : Rw_{k_n}^1, y_1 : Rw_{k_1'}^1, \dots, y_\ell : Rw_{k_\ell'}^1, y'_1 : Rw_{k_1'}^1, \dots, y'_\ell : Rw_{k_\ell'}^1\}$, Δ and Ψ are as before, $\Delta' = (\bigwedge_{i \in I} p_i[\mathbf{t}(\vec{z})/\vec{x}] = q_i[\mathbf{t}(\vec{z})/\vec{x}] \wedge \bigwedge_{j \in J} w_j[\mathbf{t}(\vec{z})/\vec{x}] : s_j \wedge \bigwedge_{l \in L} s(y'_l) = t_l[\mathbf{t}(\vec{z})/\vec{x}] \wedge \bigwedge_{l \in L} \mathbf{t}(y'_l) = t'_l[\mathbf{t}(\vec{z})/\vec{x}])$ checks that the conditions for the application of the rule r are satisfied after applying the rewrites $\vec{z}$ to the arguments of t , and $\Phi = (\bigwedge_{l \in L} y_l; t'_l[\mathbf{t}(\vec{z})/\vec{x}] = t_l[\mathbf{t}(\vec{z})/\vec{x}]; y'_l)$ states the correspondence between the “side” rewrites $\vec{y}$ and $\vec{y}'$ (via $\vec{z}$).

We briefly comment on the definition of $\text{Proof}(\mathcal{R})$. The operators defined at point 2.(b) are the obvious source/target projections and sequential composition of rewrites, with the axioms stating that, for each k , the rewrites in Rw_k are the arrows of a category with objects in k . The operators f in Σ are lifted to functors over rewrites in 3, while the equations in E are extended to rewrites in 4. It is worth noting that: (i) when $f \in \Sigma$ is lifted, only unfrozen positions can have rewrites as arguments, and therefore the functoriality is stated w.r.t. unfrozen positions only; (ii) the axioms in E are extended to one-step rewrites only (in unfrozen positions), hence, they hold for sequences of rewrites if and only if they can be proved to hold for each rewrite step. Point 5.(a) defines the basic one-step rewrites, i.e., where no rewrite occurs in the arguments $\vec{x}$. Point 5.(b) accounts for nested rewrites $\vec{z}$ below r , provided that the side-conditions of r are satisfied by the initial state; in particular note that the expression $r(\vec{z}, \vec{y})$ is always equivalent to $r(\vec{x}, \vec{y}); t'[\vec{z}/\vec{x}]$ (see decomposition law), where first r is applied at the top of the term and then the arguments are rewritten according to $\vec{z}$ under t' . Finally, the exchange law states that, under suitable hypotheses, the arguments $\vec{x}$ can be equivalently rewritten first, and the rewrite rule r applied later. Note that, as in the equations extending E , the exchange law is stated for one-step nested rewrites only. Nevertheless, it can be used in conjunction with the decomposition law to prove the exchange law for arbitrary long sequences of rewrites (provided that it can be applied step-by-step).

An important property for $\text{Proof}(\mathcal{R})$ is the preservation of the underlying state theory (Σ, E) . Otherwise, the additional axioms in $\text{Proof}(\mathcal{R})$ might collapse terms that are different in (Σ, E) . In this regard, the fact of adding the sorts Rw_k^1 and Rw_k on top of k is a potential source of term collapses. However, we can prove that, for any GRT $\mathcal{R}$, the theory $\text{Proof}(\mathcal{R})$ is a conservative extension of the underlying theory (Σ, E) .

Proposition 3.1. *Let $\mathcal{R} = (\Sigma, E, \phi, R)$ be a GRT, and let $t, t' \in T_\Sigma(X)_k$, and $s \in S_k$ for some kind k . Then, for any formula φ of the form $t : k$ or $t : s$ or $t = t'$ we have that:* $E \vdash (\forall X) \varphi \Leftrightarrow \text{Proof}(\mathcal{R}) \vdash (\forall X) \varphi$.

The main result is that $\text{Proof}(\mathcal{R})$ is complete w.r.t. the inference rules in Figure 2.

Theorem 3.2 (Completeness I). *For any GRT $\mathcal{R} = (\Sigma, E, \phi, R)$ and any $t, t' \in T_\Sigma(X)_k$, we have: $\mathcal{R} \vdash (\forall X) t \rightarrow t' \Leftrightarrow \exists \alpha. \text{Proof}(\mathcal{R}) \vdash (\forall X) \alpha : Rw_k \wedge s(\alpha) = t \wedge \mathbf{t}(\alpha) = t'$.*

The relevance of the MEL theory $Proof(\mathcal{R})$ is far beyond the essence of reachability, as it precisely characterizes the class of computational models of $\mathcal{R}$.

Definition 3.4 (Concurrent models of $\mathcal{R}$). *Let $\mathcal{R}$ be a GRT. A concurrent model of $\mathcal{R}$ is a $Proof(\mathcal{R})$ -algebra.*

Since $Proof(\mathcal{R})$ is an ordinary MEL theory, it admits initial and free models [11]. Hence, the completeness result can be consolidated by stating the equivalence between formulae provable in $Proof(\mathcal{R})$ using MEL deduction rules, formulae holding for all concurrent models of $\mathcal{R}$ and formulae valid in the initial and free concurrent models.

Theorem 3.3 (Completeness II). *For $\mathcal{R}$ a GRT and for any MEL sentence φ over $Proof(\mathcal{R})$ (and thus, for φ any of the formulae $\alpha : Rw_k, s(\alpha) = t, t(\alpha) = t'$), we have: $Proof(\mathcal{R}) \vdash (\forall X) \varphi \Leftrightarrow Proof(\mathcal{R}) \models (\forall X) \varphi \Leftrightarrow \mathbb{T}_{Proof(\mathcal{R})}(X) \models (\forall X) \varphi$.*

Theorems 3.1, 3.2 and 3.3 can be combined together to state a stronger completeness result for $Proof(\mathcal{R})$, showing the equivalence between deduction at the level of GRTs, their (initial and free) reachability models, and their (initial and free) concurrent models.

By Theorem 2.1 we have that the specialized versions of all our results for GRT over unsorted equational theories without frozen arguments and without equality / membership conditions in rewrite rules coincide with the classical ones. In particular, if $\mathcal{R}$ is an ordinary rewrite theory, any $\mathcal{R}$ -system is a concurrent model of the corresponding GRT $\hat{\mathcal{R}}$, because there is a forgetful functor $\mathbf{M}_{\mathcal{R}}$ from the category of $Proof(\hat{\mathcal{R}})$ -algebras to the category of $\mathcal{R}$ -systems. Indeed, the functor $\mathbf{M}_{\mathcal{R}}$ preserves initial and free models.

Conclusion

We have defined *generalized rewrite theories* to substantially extend the expressiveness of rewriting logic in many applications. We have given rules of deduction for these theories, defined their models as MEL algebras, and shown that initial and free models exist (for both reachability and true concurrency models). We have also shown that this generalized rewriting logic is complete with respect to its model theory, and that our results generalize the original results for unsorted rewrite theories in [10]. Future work will make more explicit the 2-categorical nature of our model theory, and will develop the semantics of *generalized rewrite theory morphisms*, extending the ideas in [9].

When evaluating the trade-offs between the complexity of the presentation and the expressiveness of the proposed rewrite theories, we have preferred to give the precise foundational semantics for the most general form of rewrite theories used in practice. Although the result suggests that MEL is expressive enough to embed GRTs just as MEL theories plus some syntactic sugar, we argue that the intrinsic separation of concerns in GRTs (i.e., equational vs operational reasoning) is fundamental in most applications.

The theory *Proof*($\mathcal{R}$) has an obvious reading as the GRT counterpart of the classic Curry-Howard isomorphism. Along this line of research there is a flourishing literature that focuses on the full integration of type theory with rewriting logic. We just mention the joint work of Stehr with the second author on the formalization of Pure Type Systems in RL [14], and the work of Cirstea, Kirchner and Liquori on the ρ -calculus [1].

Acknowledgment. We thank Mark-Oliver Stehr, Peter Ölveczky, and Jean-Bernard Stefani for helping us along the way to frozen operators, and all the members of the Maude team for invaluable insights towards more general notions of rewrite theory. We warmly thank Miguel Palomino, Narciso Martí-Oliet and the anonymous reviewers for many helpful comments.

References

1. H. Cirstea, C. Kirchner, and L. Liquori. The Rho Cube. *Proc. FoSSaCS'01, LNCS* 2030, pp. 168–183. Springer, 2001.
2. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: Specification and programming in rewriting logic. *TCS* 285:187–243, 2002.
3. R. Diaconescu and K. Futatsugi. *CafeOBJ Report: The language, proof techniques, and methodologies for object-oriented algebraic specification, AMAST Series in Computing* volume 6. World Scientific, 1998.
4. S. Lucas. Termination of rewriting with strategy annotations. *Proc. LPAR'01, Lect. Notes in Artificial Intelligence* 2250, pp. 669–684. Springer, 2001.
5. N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. *Handbook of Philosophical Logic* volume 9, pp. 1–87. Kluwer, second edition, 2002.
6. N. Martí-Oliet and J. Meseguer. Rewriting logic: roadmap and bibliography. *Theoret. Comput. Sci.* 285(2):121–154, 2002.
7. N. Martí-Oliet, K. Sen, and P. Thati. An executable specification of asynchronous pi-calculus semantics and may testing in Maude 2.0. *Proc. WRLA'02, ENTCS* 71. Elsevier, 2002.
8. N. Martí-Oliet and A. Verdejo. Implementing CCS in Maude 2. *Proc. WRLA'02, ENTCS* 71. Elsevier, 2002.
9. J. Meseguer. Rewriting as a unified model of concurrency. Technical Report SRI-CSL-90-02R, SRI International, Computer Science Laboratory, 1990.
10. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoret. Comput. Sci.*, 96:73–155, 1992.
11. J. Meseguer. Membership algebra as a logical framework for equational specification. *Proc. WADT'97, LNCS* 1376, pp. 18–61. Springer, 1998.
12. Protheo Team. The ELAN home page, 2001. www page <http://elan.loria.fr>.
13. M.-O. Stehr, J. Meseguer, and P. Ölveczky. Rewriting logic as a unifying framework for Petri nets. *Unifying Petri Nets, LNCS* 2128, pp. 250–303. Springer, 2001.
14. M.-O. Stehr and J. Meseguer. Pure Type Systems in Rewriting Logic. *Proc. LFM'99*. 1999.

Sophistication Revisited

Luís Antunes^{1*} and Lance Fortnow²

¹ DCC-FC & LIACC-University of Porto
R.Campo Alegre, 823, 4150-180 Porto, Portugal.

`lfa@ncc.up.pt`

² NEC Laboratories America

4 Independence way, Princeton, NJ 08540.

`fortnow@nec-labs.com`

Abstract. The Kolmogorov structure function divides the smallest program producing a string in two parts: the *useful information* present in the string, called *sophistication* if based on total functions, and the remaining *accidental information*. We revisit the notion of sophistication due to Koppel, formalize a connection between *sophistication* and a variation of *computational depth* (intuitively the useful or nonrandom information in a string), prove the existence of strings with maximum sophistication and show that they encode solutions of the halting problem, i.e., they are the deepest of all strings.

1 Introduction

Kolmogorov complexity measures the amount of information in a string x as the length of the shortest description of x . The Kolmogorov structure function divides the smallest program producing a string in two parts: the *useful information* present in the string and the remaining *accidental information*. Kolmogorov represented the useful information by a finite set of which the object is a typical element, so that the two-stage description of the finite set together with the index of the object in that set is as short as the shortest one-part description. Cover [Cov85] suggests that the Kolmogorov structure function is similar to “algorithmic sufficient statistics” and it is suggested that it is the algorithmic approach to the probabilistic notion of sufficient statistics. Later Gács *et. al.* [GTV01] established this relation, generalizing the Kolmogorov structure function approach to computable probability functions, the resulting theory is referred to as *Algorithmic Statistics*. Koppel [Kop88] used total functions to represent the useful information, and the resulting measure has been called *sophistication*. Recently, Vereshchagin and Vitányi [VV02], claimed the rightness of the Kolmogorov structure function, proving that ML, MDL, and related methods in model selection, always give a best possible model, in complexity-restricted model classes.

* Research done during an academic internship at NEC. This author is partially supported by funds granted to LIACC through the Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia and Programa POSI.

Koppel [Kop88] looked at the connection between sophistication and logical depth for infinite strings. Koppel's paper uses a different definition of logical depth imposing totality in the functions defining logical depth and defining the length of a time bound by the smallest program describing it. Bennett [Ben88] formally defined the *s-significant logical depth* of an object x as the time required by a standard universal Turing machine to generate x by a program that is no more than s bits longer than the shortest descriptions of x . Antunes *et. al.* [AFvM01] considered logical depth as one instantiation of a more general theme, computational depth, and propose several other variants. Intuitively, *computational depth* measures the amount of “nonrandom” or “useful” information in a string. Formalizing this intuitive notion is tricky. A computationally deep string x should take a lot of effort to construct from its short description. Incompressible strings are trivially constructible from their shortest description, and therefore computationally shallow.

In this paper we take a fresh look at sophistication showing that there are strings with high depth but low sophistication, there are strings with near maximum sophistication and strings of high sophistication encode the halting problem for smaller strings.

We define a notion of coarse sophistication, a robust variation of sophistication. We show this notion roughly equivalent to a variation of computational depth based on the busy beaver function.

2 Preliminaries

We use binary alphabet $\Sigma = \{0, 1\}$ for encoding strings. Our computation model will be Turing machines, and U denotes a fixed universal Turing machine. The function $\log$ denote $\log_2$, and we will use $|\cdot|$ for the length of a string and also to the cardinality of a set.

Kolmogorov Complexity

We give essential definitions and basic result in Kolmogorov complexity for our needs and refer the reader to [LV97] for more details.

Definition 1. A function $t : \mathbf{N} \rightarrow [0, \infty)$ is time-constructible if there exists a Turing machine that runs in time exactly $t(n)$ on every input of size n .

All explicit resource bounds we use in this paper are time-constructible.

Definition 2. Let U be a fixed universal Turing machine. Then for any string $x \in \{0, 1\}^*$, the Kolmogorov complexity of x is, $C(x) = \min_p \{|p| : U(p) = x\}$. For any time constructible t , the t -time-bounded Kolmogorov complexity of x is, $C^t(x) = \min\{|p| : U(p) = x \text{ in at most } t(|x|) \text{ steps}\}$.

We extend the definition of Kolmogorov complexity to finite sets in the following way: the Kolmogorov complexity of a set S (denoted $C(S)$) is the Kolmogorov

complexity of a list of its elements. As noted by Cover [Cov85], Kolmogorov proposed in 1973 at the Information Theory Symposium, Tallin, Estonia the following function:

Definition 3. *The Kolmogorov structure function $H_k(x|n)$ of $x \in \Sigma^n$ is defined by*

$$H_k(x|n) = \min\{\log |S| : x \in S \text{ and } C(S|n) \leq k\}.$$

Of special interest is the value

$$C^*(x|n) = \min\{k : H_k(x|n) + k = C(x|n)\}.$$

A program for x can be written in two stages:

1. Use p to print the indicator function for S .
2. The desired string is the i th sequence in a lexicographic ordering of the elements of this set.

This program has length $|p| + \log |S| + O(1)$, and $C^*(x|n)$ is the length of the shortest program p for which this two-stage description is as concise as the shortest one stage description. Note that x must be maximally random (a typical element) with respect to S otherwise the two stage description could be improved, contradicting the minimality of $C(x|n)$. Gács *et al.* [GTV01] generalize the model class from finite sets to probability distributions, where the models are computable probability density functions.

In 1982 at a seminar in the Moscow State University (see [V'y99]) Kolmogorov raised the question if “absolutely non-random” (or non-stochastic) objects exist.

Definition 4. *Let α and β be natural numbers. A string $x \in \Sigma^n$ is called (α, β) -stochastic if there exists a finite set S such that $x \in S$, $C(S) \leq \alpha$ and $C(x|S) \geq \log |S| - \beta$.*

The following theorem, proved by Shen [She83], is part of the answer for a corresponding problem posed by Kolmogorov about the existence of “absolutely non-random” strings.

Theorem 1. *1. There exists a constant c such that, for any n and any α and β with $\alpha \geq \log n + c$ and $\alpha + \beta \geq n + 4 \log n + c$, all the numbers from 0 to $2^n - 1$ are (α, β) -stochastic.*
2. There exists a constant c such that, for any n and any α and β with $2\alpha + \beta < n - 6 \log n - c$ not all the numbers from 0 to $2^n - 1$ are (α, β) -stochastic.

Gács *et al.* [GTV01] improved Shen’s result and proved that for every n there are objects of length n with complexity $C(x|n) \approx n$ such that every explicit algorithmic sufficient statistic for x has complexity about n .

2.1 Sophistication

Expressing the useful information as a recursive function Koppel [Kop88,KA91] [Kop91] introduced the concept of *sophistication* of an object based on process (monotonic) complexity defined by Schnorr [Sch73]. A function $f : \Sigma^* \rightarrow \Sigma^*$ is monotonic if $x \leq y$ (x is a prefix of y) implies that $f(x) \leq f(y)$ for all x and y . S_Σ is a sample space consisting of all finite and infinite sequences over Σ . If α is an infinite string then $\alpha_{1:n}$ represent the first n bits of α .

Definition 5. Let U be the reference monotone machine. The monotone complexity, is defined as,

$$Km(x) = \min_p \{|p| : U(p) = x\omega, \omega \in S_\Sigma\}.$$

Definition 6. The sophistication of an infinite string α is,

$$\begin{aligned} \text{soph}_c(\alpha) = \min_p \{ & |p| : p \text{ is total, for all } n \text{ exists } d_n \\ & \text{such that } |p| + |d_n| \leq Km(\alpha_{1:n}) + c \text{ and } d_{n-1} \leq d_n \} \end{aligned}$$

Computational Depth

The Kolmogorov complexity of a string x does not take into account the time necessary to produce the string from a description of length $C(x)$. Levin [Lev73], introduced a useful variant weighing program size and running time.

Definition 7. For any strings x, y , the Levin complexity of x given y is

$$Ct(x|y) = \min_p \{|p| + \log t : U(p, y) \text{ halts in at most } t \text{ steps and outputs } x\}.$$

After some attempts, Bennett [Ben88] formally defined the s -significant logical depth of a string x as the time required by a standard universal Turing machine to generate x by a program that is no more than s bits longer than the shortest descriptions of x . A string x is called logically deep if it takes a lot of time to generate it from any short description.

Definition 8. Let x be a string and s be a nonnegative integer. The logical depth of x at a significance level s is

$$\text{depth}_s(x) = \min_p \{t : U(p) \text{ halts in at most } t \text{ steps, outputs } x \text{ and } |p| < C(x) + s\}$$

Note that algorithmically random strings are shallow at any significance level. In particular, Chaitin's Ω is shallow. Deep strings are hard to find, however they can be constructed by diagonalization, see [Ben88]. Bennett has proved that a fast deterministic processes is unable to transform a shallow object into a deep one, and that fast probabilistic processes can do so only with small probability (*slow growth law*).

Antunes, Fortnow and van Melkebeek [AFvM01] consider logical depth as one instantiation of this more general theme, the authors propose several other variants and show many applications. Intuitively strings of high computational depth are low Kolmogorov complexity strings (and hence nonrandom), but a resource bounded machine cannot identify this fact. The following measure was introduced in [AFvM01].

Definition 9. *For any string x the basic computational depth of x is*

$$bcd(x) = Ct(x) - C(x) = \min_p \{ \log(t) + |p| - C(x) : U(p) \text{ outputs } x \text{ in } t \text{ steps} \}.$$

Basic computational depth incorporate the significance level in the formula, adding to (the logarithm of) the running time a penalty $|p| - C(x)$.

3 Some Results on Sophistication

Like depth, sophistication is a measure of the non-random information in a string. We start by recasting Definition 6 for finite strings.

Definition 10. *Let c be a constant, $x \in \Sigma^n$ and U the universal reference Turing machine*

$$soph_c(x) = \min\{|p| : p \text{ is total, exists } d [U(p, d) = x \text{ and } |p| + |d| \leq C(x) + c]\}.$$

Initial sequences of the halting set give a nice example of a language with high depth but low sophistication.

Example 1. Let ϕ be a universal partial recursive function. Define $\chi = \chi_1\chi_2\chi_3\ldots$ the characteristic sequence of the diagonal halting set $K_0 = \{i : \phi_i(i) < \infty\}$

$$\chi_i = \begin{cases} 1 & \text{if } i \in K_0 \\ 0 & \text{otherwise} \end{cases}$$

By the Barzdin's Lemma (see [LV97]) $\log n \leq C(\chi_{1:n}|n) \leq \log n + c$, so $soph_s(\chi_{1:n}) \leq \log n + c$, but $depth_s(\chi_{1:n})$ is very high ($\chi_{1:n}$ is very deep).

We can also use the characteristic sequence of the recursive set constructed (by diagonalization) in Li and Vitányi [LV97, Theorem 7.1.4] to show this gap between logical depth and sophistication.

Regarding the question raised by Kolmogorov about the existence of “absolutely non-random” (or absolutely non-stochastic) objects, we note that we can reformulate the question using sophistication, and ask if exists strings $x \in \Sigma^n$ such that the sophistication is close to n , i.e., highly sophisticated strings.

Theorem 2. *Let c be a constant, for some string $x \in \Sigma^n$,*

$$soph_c(x) > n - 2 \log n - 2c.$$

Proof. For all p such that $|p| \leq n - 2 \log n - 2c$ we define

$$r_p = \begin{cases} 0 & \text{if } \exists d : |d| < n - |p| - c \text{ such that } U(p, d) \text{ diverges} \\ \max_{d: |d| < n - |p| - c} & \text{running time of } U(p, d) \end{cases}$$

Let $S = \max_p r_p$. Given n and p that maximizes r_p we can compute S . Consider

$$V = \{x : \exists p, d \ |p| \leq n - 2 \log n - 2c, \ |d| \leq n - |p| - c \text{ s.t. } U^t(p, d) = x, \ t \leq S\}.$$

Let z be the least, in the lexicographic order, element in $\{0, 1\}^n$ such that $z \notin V$. Such z exists since $\forall x \in V, C(x) \leq |p| + |d| \leq |p| + n - |p| - c = n - c$ and by a simple counting argument there must exist at least $2^n(1 - \frac{1}{2^c})$ strings $z \in \{0, 1\}^n$ with $C(z) \geq n - c$. Since given n and p that maximizes r_p we can compute S , then by construction $C(z) \leq C(p) + 2 \log n \leq n - 2c$.

Assume that $\text{soph}_c(z)$ is small, i.e., $\text{soph}_c(z) \leq n - 2 \log n - 2c$; then, by definition

$$\exists p^*, d^* : p^* \text{ is total, } |p^*| \leq n - 2 \log n - 2c \text{ and } |p^*| + |d^*| \leq C(z) + c$$

but then we have that $|p^*| \leq n - 2 \log n - 2c$ and $|d^*| \leq C(z) + c - |p^*| \leq n - |p^*| - c$ so $U(p^*, d^*)$ runs in time $\leq S$, i.e., $z \in V$. But by construction $z \notin V$, so

$$\text{soph}_c(z) > n - 2 \log n - 2c.$$

◇

We can get a sharper result using conditional Kolmogorov complexity. Gács *et al.* [GTV01] proved a similar result independently.

Corollary 1. *For some string x of length n , $\text{soph}_c(x|n) \geq n - c$.*

4 Coarse Sophistication

Koppel's definition of sophistication, Definition 10, may not be stable. Small changes in c could cause large changes in $\text{soph}_c(x)$. In this section, we consider a new notion of *coarse sophistication* that incorporate the “constant” as a penalty in the formula to obtain a more robust measure.

Definition 11. *The coarse sophistication of a string $x \in \Sigma^n$, is defined as*

$$csoph(x) = \min\{2|p| + |d| - C(x) : U(p, d) = x \text{ and } p \text{ is total.}\}$$

Think of this definition as $|p|$ for sophistication plus a penalty $|p| + |d| - C(x)$ for how far away we are from the minimal program. The choice of using $|p| + |d| - C(x)$ instead of some other penalty function is admittedly arbitrary but it seems natural and does lead to some interesting consequences.

Some “sensibility” is lost as now $csoph(x)$ is upper bounded by $\frac{|x|}{2}$.

Theorem 3. *There is a constant c such that for all $x \in \Sigma^n$, $csoph(x) \leq \frac{n}{2} + c$.*

Proof. If $C(x) \leq \frac{n}{2}$ then by definition $csoph(x) \leq \frac{n}{2} + c$.

If $C(x) > \frac{n}{2}$ then considering the *print*(x) program we have $csoph(x) \leq \frac{n}{2} + c$. $\diamond$

There are strings for which this upper bound is tight.

Theorem 4. *For some string x of length n , $csoph(x) > \frac{n}{2} - 4 \log n$.*

Proof. For all p such that $|p| \leq n - 2 \log n$ we define

$$r_p = \begin{cases} 0 & \text{if exists } d : |d| < n - |p| \text{ such that } U(p, d) \text{ diverges} \\ \max_{d: |d| < n - |p|} & \text{running time of } U(p, d) \end{cases}$$

Let $S = \max r_p$. Given n and p that maximizes r_p we can compute S . Consider

$$V = \{x : \text{exists } p, d \mid |p| \leq \frac{n}{2} - 2 \log n, \ |d| \leq n - 2|p| - 2 \log n \text{ s.t. } U^t(p, d) = x, \ t \leq S\}.$$

Let z be the least, in the lexicographic order, element in $\{0, 1\}^n$ such that $z \notin V$. Such z exists since for all $x \in V$, $C(x) < |p| + |d| + 2 \log n \leq |p| + 2 \log n + n - 2|p| - 2 \log n = n - |p| < n$ and by a simple counting argument there exists random strings. By construction we know that

$$C(z) \leq C(p) + 2 \log n \leq \frac{n}{2}.$$

Assume that $csoph(z)$ is small, i.e., $csoph(z) \leq \frac{n}{2} - 4 \log n$; the program p^* which defines the sophistication is such that

$$|p^*| \leq \frac{n}{2} - 2 \log n$$

and there exists d^* such that

$$2|p^*| + |d^*| - C(z) \leq \frac{n}{2} - 2 \log n$$

but then we have that

$$|d^*| \leq \frac{n}{2} - 2 \log n + C(z) - 2|p^*| \leq n - 2 \log n - 2|p^*|$$

so $U(p^*, d^*)$ runs in time $\leq S$, i.e., $z \in V$. But by construction $z \notin V$, so

$$csoph(z) > \frac{n}{2} - 4 \log n.$$

$\diamond$

We investigate the computational power of coarse sophistication, relating coarse sophistication with the halting problem. We prove that with the knowledge of the sophistication of a string $x \in \Sigma^n$ and some extra $\log n$ bits we can solve the halting problem for all programs of length smaller than $\frac{csoph(x)}{2} - 2 \log n$. This confirms our position that sophistication truly is the ultimate limit of depth.

Theorem 5. *For all $x \in \Sigma^n$, given x and $O(\log n)$ bits we can solve the halting problem for all programs q such that $|q| < \frac{csoph(x)}{2} - 2 \log n$.*

Proof. With the given $O(\log n)$ bits find the minimum program p such that $U(p) = x$, and consider S its running time. Suppose that there is some q such that $|q| < \frac{csoph(x)}{2} - 2 \log n$ and $U(q)$ converges in time $v > S$. Consider the program w such that $U(w, p)$ first computes v and then simulates $U(p)$ for v steps, producing x . Now w is total and there is a constant c such that $|w| = |q| + c$, and

$$\begin{aligned} csoph(x) &\leq 2|w| + |p| - C(x) \\ &\leq 2|q| + 2c + |p| - C(x) \\ &\leq csoph(x) - 2 \log n + 2c + |p| - C(x) \\ &< csoph(x) - 2 \log n + 2c \end{aligned}$$

so such an input can not exist. ◇

5 Coarse Sophistication vs. Busy Beaver Computational Depth

The main drawback of basic computational depth is the fact that it is only suitable for strings whose program runs in time at most exponential in the length of the string. This motivate the definition of *busy beaver computational depth*, preserving the intuition of basic computational depth and capturing all possible running times. Besides, using properly the Busy Beaver function, we can scale down computational depth from running time to program length. Usually the busy beaver function, $BB(n)$, is defined to be the largest number which can be computed by an n -state Turing machine. Although many variations on the definition of busy beaver have been used, we recast the original definition (see Daley [Dal82]) as follows:

Definition 12. *The busy beaver function is defined $BB : \mathbf{N} \rightarrow \mathbf{N}$ as*

$$BB(n) = \max_{p: |p| \leq n} \{ \text{Running time of } U(p) \text{ when defined} \}$$

Definition 13. *The busy beaver computational depth of $x \in \Sigma^n$, is defined as*

$$depth_{bb}(x) = \min\{|p| - C(x) + k : U(p) = x \text{ in } t \text{ steps and } t \leq BB(k)\}$$

As in basic computational depth, $depth_{bb}$ also incorporate a significance level in the formula; $|p| - C(x)$ is a penalty measuring how far away we are from the minimal program. It is important to note that, instead of use the running time, $depth_{bb}$ uses the inverse Busy Beaver of the running time. However, as for coarse sophistication, some “sensibility” is lost, $depth_{bb}$ is nearly upper bounded by $\frac{n}{2}$.

Theorem 6. *There is a constant c such that for all $x \in \Sigma^n$, $\text{depth}_{bb}(x) \leq \frac{n}{2} + BB^{-1}(n) + c$.*

Proof. If $C(x) \leq \frac{n}{2}$, considering the minimum program producing x we have $\text{depth}_{bb}(x) \leq \frac{n}{2} + c$.

If $C(x) > \frac{n}{2}$, considering the $\text{print}(x)$ program we have $\text{depth}_{bb}(x) \leq \frac{n}{2} + BB^{-1}(n) + c$. $\diamond$

There are strings for which this upper bound is tight.

Theorem 7. *For some string x of length n , $\text{depth}_{bb}(x) > \frac{n}{2} - 2 \log n$*

Proof. Let V be the set of all $x \in \Sigma^n$ such that there exists p and k such that $|p| \leq \frac{n}{2} - 2 \log n$, $k \leq n - 2 \log n - |p|$, $U(p) = x$ in t steps and $t \leq BB(k)$.

Consider z the least, in the lexicographic order, element in Σ^n such that $z \notin V$. Such z exists since for all $x \in V$, $C(x) < |p| + 2 \log n \leq \frac{n}{2}$. We can approximate BB from below so V is r.e. and by construction we know that the size of V is smaller than $2^{|p| - 2 \log n}$ so $C(z) < \frac{n}{2}$.

Assume $\text{depth}_{bb}(z) \leq \frac{n}{2} - 2 \log n$, then by definition exists a p and k such that

$$k + |p| - C(z) \leq \frac{n}{2} - 2 \log n \text{ and } U(p) = x \text{ in } t \text{ steps and } t \leq BB(k),$$

i.e.,

$$k \leq n - 2 \log n - |p| \text{ and } U(p) = x \text{ in } t \text{ steps and } t \leq BB(k)$$

so $z \in V$. But by construction $z \notin V$, so $\text{depth}_{bb}(z) > \frac{n}{2} - 2 \log n$. $\diamond$

We now prove the equivalence between coarse sophistication and busy beaver computational depth.

Theorem 8. *For all $x \in \Sigma^n$, $|csoph(x) - \text{depth}_{bb}(x)| \leq O(\log n)$.*

Proof. We use p_s to denote the program associated with $csoph$ and p_d with depth_{bb} .

- We start proving that $\text{depth}_{bb}(x) \leq csoph(x) + O(\log n)$.

For all d such that $|d| \leq n$, let t be the maximum running time of $U(p_s, d)$, then using p_s and $\log n$ bits to describe d we get $t \leq BB(|p_s| + O(\log n))$.

So we get an upper bound on the depth of x

$$\begin{aligned} \text{depth}_{bb}(x) &\leq |p_s| + |d| - C(x) + |p_s| + O(\log n) \\ &\leq csoph(x) + O(\log n) \end{aligned}$$

- Now we prove that $csoph(x) \leq \text{depth}_{bb}(x) + O(\log n)$.

We denote the running time of a program p by $rt(p)$. Let q be the first program of length $k = BB^{-1}(rt(p_d))$ whose running time immediately follows the running time of p_d , i.e.

$$|q| = k, \quad rt(q) \geq rt(p_d) \text{ and for all } u : |u| = k, \quad rt(u) > rt(p_d) \Rightarrow rt(q) \leq rt(u).$$

Consider the set

$$A = \{v : |v| = |p_d|, \text{ } rt(v) < rt(q) \text{ and for all } u : |u| = k, \text{ } rt(u) > rt(v) \Rightarrow rt(u) > rt(q)\}.$$

Given $q, n, |p_d|$ and k , A is recursive, since $rt(q)$ is used as the time limit for the running time of all programs. By symmetry of information we have

$$C(v|q) \leq C(q|v) + C(v) - C(q) + O(\log n).$$

But, given v we can use its running time to get q , because q is the first program of length k whose running time immediately follows the running time of p_d , so $C(q|v) \leq O(\log n)$ and $C(v|q) \leq |p_d| - k + O(\log n)$.

As A is recursive we have $|A| \leq 2^{|p_d| - k + O(\log n)}$ and we can point p_d with its index i in A . Note that for every $p \in A$, $rt(p) < rt(q)$, i.e., it halts. Now consider the code of the machine that with input $\langle q, n, |p_d|, k \rangle$ and i constructs the set A and picks p_d that given as input to the universal Turing machine produces x . Then

$$\begin{aligned} csoph(x) &\leq 2|q| + O(\log n) + |i| - C(x) \\ &\leq 2k + O(\log n) + |p_d| - k - C(x) \\ &\leq k + |p_d| - C(x) + O(\log n) \\ &\leq depth_{bb}(x) + O(\log n) \end{aligned}$$

◇

Acknowledgment. The authors thank Paul Vitányi and the anonymous ICALP reviewers for their comments.

References

- [Ant02] L. Antunes. Useful Information. *PhD Thesis*, Computer Science Department, Oporto University, 2002.
- [AFvM01] Luis Antunes, Lance Fortnow, and Dieter van Melkebeek. Computational depth. In *Proceedings of the 16th IEEE Conference on Computational Complexity*, pages 266–273, 2001.
- [Ben88] C. H. Bennett. Logical depth and physical complexity. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 227–257. Oxford University Press, 1988.
- [Cov85] T. M. Cover. Kolmogorov Complexity, Data Compression, and Inference, pp. 23–33. In *The Impact of Processing Techniques on Communications*. J. K. Skwirzynski, Ed., Martinus Nijhoff Publishers, 1985.
- [Dal82] Robert P. Daley. Busy Beaver Sets: Characterizations and Applications. In *Information and Control*, 52(1982), 52–67.
- [GTV01] Peter Gács and John Tromp and Paul Vitányi. Algorithmic Statistics. In *IEEE Trans. Inform. Theory*, 47:6(2001), 2443–2463.

- [Kop88] M. Koppel. Structure, pp. 435–452. In *The Universal Turing Machine: A Half-Century Survey*. R. Herken, Ed., Oxford University Press, 1988.
- [Kop91] M. Koppel. Learning to Predict Non-Deterministically Generated Strings. *Machine Learning*, 7(1991), 85–99.
- [KA91] M. Koppel and H. Atlan. An Almost Machine-Independent Theory of Program-Length Complexity, Sophistication, and Induction. *Information Sciences*, 56(1991), 23–33.
- [Lev73] Leonid A. Levin. Universal Search Problems. *Problems Inform. Transmission*, 9(1973), 265–266.
- [Lev84] Leonid A. Levin. Randomness conservation inequalities: information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.
- [LV97] Ming Li and Paul M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2nd edition, 1997.
- [Sch73] C. P. Schnorr. Process Complexity and Effective Random Tests. *J. Comp. System Sci.*, 7(1973), 376–388.
- [She83] A. Kh. Shen. The Concept of (α, β) -Stochasticity in the Kolmogorov Sense, and its Properties. *Soviet Math. Dokl.*, 28(1983), 295–299.
- [VV02] N. Vereshchagin and P. Vitányi. Kolmogorov’s structure functions and an application to the foundations of model selection. *Proc. 47th IEEE Symp. Found. Comput. Sci.*, 2002.
- [Vit02] P. Vitányi. Meaningful information. *Proc. 13th International Symposium on Algorithms and Computation (ISAAC’02)*. Lecture Notes in Computer Science, Vol 2518, Springer-Verlag, Berlin, 2002, 588–599.
- [V’y99] V.V. V’yugin. Algorithmic complexity and stochastic properties of finite binary sequences. *The Computer Journal*, 42(4):294–317, 1999.

Scaled Dimension and Nonuniform Complexity

John M. Hitchcock^{1*}, Jack H. Lutz^{1**}, and Elvira Mayordomo^{2***}

¹ Department of Computer Science, Iowa State University

{jhitchco,lutz}@cs.iastate.edu

² Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza

elvira@posta.unizar.es

Abstract. Resource-bounded dimension is a complexity-theoretic extension of classical Hausdorff dimension introduced by Lutz (2000) in order to investigate the fractal structure of sets that have resource-bounded measure 0. For example, while it has long been known that the Boolean circuit-size complexity class $\text{SIZE}\left(\alpha \frac{2^n}{n}\right)$ has measure 0 in ESPACE for all $0 \leq \alpha \leq 1$, we now know that $\text{SIZE}\left(\alpha \frac{2^n}{n}\right)$ has dimension α in ESPACE for all $0 \leq \alpha \leq 1$. The present paper furthers this program by developing a natural hierarchy of “rescaled” resource-bounded dimensions. For each integer i and each set X of decision problems, we define the i^{th} dimension of X in suitable complexity classes. The 0^{th} -order dimension is precisely the dimension of Hausdorff (1919) and Lutz (2000). Higher and lower orders are useful for various sets X . For example, we prove the following for $0 \leq \alpha \leq 1$ and any polynomial $q(n) \geq n^2$.

1. The class $\text{SIZE}(2^{\alpha n})$ and the time- and space-bounded Kolmogorov complexity classes $\text{KT}^q(2^{\alpha n})$ and $\text{KS}^q(2^{\alpha n})$ have 1^{st} -order dimension α in ESPACE.
2. The classes $\text{SIZE}(2^{n^\alpha})$, $\text{KT}^q(2^{n^\alpha})$, and $\text{KS}^q(2^{n^\alpha})$ have 2^{nd} -order dimension α in ESPACE.
3. The classes $\text{KT}^q(2^n(1 - 2^{-\alpha n}))$ and $\text{KS}^q(2^n(1 - 2^{-\alpha n}))$ have -1^{st} -order dimension α in ESPACE.

1 Introduction

Many sets of interest in computational complexity have quantitative structures that are too fine to be elucidated by resource-bounded measure. For example, it has long been known that the Boolean circuit-size complexity class $\text{SIZE}\left(\frac{2^n}{n}\right)$ has measure 0 in ESPACE [13], so resource-bounded measure cannot make quantitative distinctions among subclasses of $\text{SIZE}\left(\frac{2^n}{n}\right)$.

* This research was supported in part by National Science Foundation Grant 9988483.

** This research was supported in part by National Science Foundation Grants 9610461 and 9988483.

*** This research was supported in part by Spanish Government MEC projects PB98-0937-C04-02 and TIC98-0973-C03-02. It was done while visiting Iowa State University.

In early 2000, Lutz [11] developed resource-bounded dimension in order to remedy this situation. Just as resource-bounded measure is a complexity-theoretic generalization of classical Lebesgue measure, resource-bounded dimension is a complexity-theoretic generalization of classical Hausdorff dimension. Moreover, just as classical Hausdorff dimension enables us to quantify the structures of many sets of Lebesgue measure 0, resource-bounded dimension enables us to quantify the structures of some sets that have measure 0 in complexity classes. For example, Lutz [11] showed that for every real number $\alpha \in [0, 1]$, the class $\text{SIZE}(\alpha \frac{2^n}{n})$ has dimension α in SPACE . He also showed that for every p -computable $\alpha \in [0, 1]$, the class of languages with limiting frequency α has dimension $\mathcal{H}(\alpha)$ in E , where $\mathcal{H}$ is the binary entropy function of Shannon information theory. (This is a complexity-theoretic extension of a classical result of Eggleston [3].) These preliminary results suggest new relationships between information and complexity and open the way for investigating the fractal structure of complexity classes. More recent work has already used resource-bounded dimension to illuminate a variety of topics in computational complexity [1, 2, 5, 7, 8].

However, there is a conspicuous obstacle to further progress along these lines. Many classes that occur naturally in computational complexity are parametrized in such a way as to remain out of reach of the resource-bounded dimension of [11]. For example, when discussing cryptographic security or derandomization, one is typically interested in circuit-size bounds of the form $2^{\alpha n}$ or 2^{n^α} , rather than the $\alpha \frac{2^n}{n}$ bound of the above-cited result. It is easy to see that for all $\alpha < 1$, $\text{SIZE}(2^{\alpha n})$ and $\text{SIZE}(2^{n^\alpha})$ have dimension 0 in SPACE , so the resource-bounded dimension of [11] cannot provide the sort of quantitative classification that is needed. Similarly, in their investigations of the information content of complete problems, Juedes and Lutz [9] established tight bounds on space-bounded Kolmogorov complexity of the forms 2^{n^ϵ} and $2^{n+1} - 2^{n^\epsilon}$; in the investigation of completeness in E one is typically interested in dense languages, which have census at least 2^{n^ϵ} ; etc. The difficulty here is that classes arising naturally in computational complexity are often scaled in a nonlinear way that is not compatible with the linear scaling implicit in classical Hausdorff dimension and the resource-bounded dimension of Lutz [11].

This sort of difficulty has already been encountered in the classical theory of Hausdorff dimension and dealt with by *rescaling* the dimension. The 1970 classic [15] by C.A. Rogers describes the resulting theory of *generalized dimension*, in which Hausdorff dimension may be rescaled by any element of a very large class of extended real-valued functions. Choosing the right such function for a particular set often yields more precise information about that set's dimension. For example, it is known that with probability 1 a Brownian sample path in the plane has Hausdorff dimension 2 (the dimension of the plane), but a more careful analysis using the generalized approach shows that “the dimension is, in a sense, logarithmically smaller than 2” [4].

In this paper we extend the resource-bounded dimension of [11] by introducing the notion of a *scale* according to which dimension may be measured. Our

scales are slightly less general than the functions used for generalized dimension and take two arguments instead of one, but every scale g defines for every set X of decision problems a g -scaled dimension $\dim^{(g)}(X) \in [0, 1]$. Thus, although the spirit of our approach is much like that of generalized dimension, scaled dimension typically yields quantitative results that are as precise as, but crisper than, the result quoted at the end of the preceding paragraph.

The choice of which scale to use for a particular application is very much like the choice of whether to plot data on a standard Cartesian graph or a log-log graph. In fact, a very restricted family of scales appears to be adequate for analyzing many problems in computational complexity. Specifically, we define a particular, natural hierarchy of scales, one for each integer, and use these to define the i^{th} dimension of arbitrary sets X in suitable complexity classes. The 0th-order dimension is precisely the dimension used by Hausdorff [6] and Lutz [11]. We propose that higher- and lower-order dimensions will be useful for many investigations in computational complexity. In support of this proposal we prove the following for $0 \leq \alpha \leq 1$ and any polynomial $q(n) \geq n^2$.

1. The class $\text{SIZE}(2^{\alpha n})$ and the time- and space-bounded Kolmogorov complexity classes $\text{KT}^q(2^{\alpha n})$ and $\text{KS}^q(2^{\alpha n})$ have 1st-order dimension α in ESPACE.
2. The classes $\text{SIZE}(2^{n^\alpha})$, $\text{KT}^q(2^{n^\alpha})$, and $\text{KS}^q(2^{n^\alpha})$ have 2nd-order dimension α in ESPACE.
3. The classes $\text{KT}^q(2^n(1 - 2^{-\alpha n}))$ and $\text{KS}^q(2^n(1 - 2^{-\alpha n}))$ have -1^{st} -order dimension α in ESPACE.

We emphasize that, for *all* $\alpha \in (0, 1)$, all these classes have measure 0 in ESPACE, the classes in 1 and 2 have 0th-order dimension 0 in ESPACE, and the class in 3 has 0th-order dimension 1 in ESPACE. Only when the dimension is appropriately rescaled does it respond informatively to variation of the parameter α .

We also prove more general results along these lines.

2 Preliminaries

A *decision problem* (a.k.a. *language*) is a set $A \subseteq \{0, 1\}^*$. We identify each language with its characteristic sequence $\llbracket s_0 \in A \rrbracket \llbracket s_1 \in A \rrbracket \llbracket s_2 \in A \rrbracket \cdots$, where $s_0, s_1, s_2, \dots$ is the standard enumeration of $\{0, 1\}^*$ and $\llbracket \phi \rrbracket = \text{if } \phi \text{ then } 1 \text{ else } 0$. We write $A[i..j]$ for the string consisting of the i -th through the j -th bits of (the characteristic sequence of) A . The Cantor space $\mathbf{C}$ is the set of all decision problems. If $w \in \{0, 1\}^*$ and $x \in \{0, 1\}^* \cup \mathbf{C}$, then $w \sqsubseteq x$ means that w is a prefix of x . The *cylinder* generated by a string $w \in \{0, 1\}^*$ is $\mathbf{C}_w = \{A \in \mathbf{C} \mid w \sqsubseteq A\}$. A *prefix set* is a language A such that no element of A is a prefix of any other element of A . If A is a language and $n \in \mathbb{N}$, then we write $A_{=n} = A \cap \{0, 1\}^n$, $A_{\leq n} = A \cap \{0, 1\}^{\leq n}$. All logarithms in this paper are base 2.

For each $i \in \mathbb{N}$ we define a class G_i of functions from $\mathbb{N}$ into $\mathbb{N}$ as follows.

$$\begin{aligned} G_0 &= \{f \mid (\exists k)(\forall^\infty n)f(n) \leq kn\} \\ G_{i+1} &= 2^{G_i(\log n)} = \{f \mid (\exists g \in G_i)(\forall^\infty n)f(n) \leq 2^{g(\log n)}\} \end{aligned}$$

We also define the functions $\hat{g}_i \in G_i$ by $\hat{g}_0(n) = 2n$, $\hat{g}_{i+1}(n) = 2^{\hat{g}_i(\log n)}$. We regard the functions in these classes as growth rates. In particular, G_0 contains the linearly bounded growth rates and G_1 contains the polynomially bounded growth rates. It is easy to show that each G_i is closed under composition, that each $f \in G_i$ is $o(\hat{g}_{i+1})$, and that each $\hat{g}_i$ is $o(2^n)$. Thus G_i contains superpolynomial growth rates for all $i > 1$, but all growth rates in the G_i -hierarchy are subexponential.

We use the following classes of functions.

$$\begin{aligned} \text{all} &= \{f \mid f : \{0, 1\}^* \rightarrow \{0, 1\}^*\} \\ \text{rec} &= \{f \in \text{all} \mid f \text{ is computable}\} \\ \text{p}_i &= \{f \in \text{all} \mid f \text{ is computable in } G_i \text{ time}\} \ (i \geq 1) \\ \text{p}_i\text{space} &= \{f \in \text{all} \mid f \text{ is computable in } G_i \text{ space}\} \ (i \geq 1) \end{aligned}$$

(The length of the output is included as part of the space used in computing f .) We write p for p_1 and pspace for p_1space . Throughout this paper, Δ and Δ' denote one of the classes all , rec , $\text{p}_i (i \geq 1)$, $\text{p}_i\text{space} (i \geq 1)$.

A *constructor* is a function $\delta : \{0, 1\}^* \rightarrow \{0, 1\}^*$ that satisfies $x \not\sqsubseteq \delta(x)$ for all x . The *result* of a constructor δ (i.e., the language *constructed* by δ) is the unique language $R(\delta)$ such that $\delta^n(\lambda) \sqsubseteq R(\delta)$ for all $n \in \mathbb{N}$. Intuitively, δ constructs $R(\delta)$ by starting with λ and then iteratively generating successively longer prefixes of $R(\delta)$. We write $R(\Delta)$ for the set of languages $R(\delta)$ such that δ is a constructor in Δ . The following facts are the reason for our interest in the above-defined classes of functions.

$$\begin{aligned} R(\text{all}) &= \mathbf{C}. \\ R(\text{rec}) &= \text{REC}. \\ \text{For } i \geq 1, R(\text{p}_i) &= \text{E}_i. \\ \text{For } i \geq 1, R(\text{p}_i\text{space}) &= \text{E}_i\text{SPACE}. \end{aligned}$$

If D is a discrete domain, then a function $f : D \rightarrow [0, \infty)$ is Δ -*computable* if there is a function $\hat{f} : \mathbb{N} \times D \rightarrow \mathbb{Q} \cap [0, \infty)$ such that $|\hat{f}(r, x) - f(x)| \leq 2^{-r}$ for all $r \in \mathbb{N}$ and $x \in D$ and $\hat{f} \in \Delta$ (with r coded in unary and the output coded in binary). We say that f is *exactly* Δ -*computable* if $f : D \rightarrow \mathbb{Q} \cap [0, \infty)$ and $f \in \Delta$.

3 Scaled Dimension

In this section we develop a theory of scaled dimensions in complexity classes. We then develop a particular, natural hierarchy of scaled dimensions that are suitable for complexity-theoretic applications such as those in section 4.

Definition. A *scale* is a continuous function $g : H \times [0, \infty) \rightarrow \mathbb{R}$ with the following properties.

1. $H = (a, \infty)$ for some $a \in \mathbb{R} \cup \{-\infty\}$.
2. $g(m, 1) = m$ for all $m \in H$.
3. $g(m, 0) = g(m', 0) \geq 0$ for all $m, m' \in H$.
4. For every sufficiently large $m \in H$, the function $s \mapsto g(m, s)$ is nonnegative and strictly increasing.
5. For all $s' > s \geq 0$, $\lim_{m \rightarrow \infty} [g(m, s') - g(m, s)] = \infty$.

Example 3.1. The function $g_0 : \mathbb{R} \times [0, \infty) \rightarrow \mathbb{R}$ defined by $g_0(m, s) = sm$ is the canonical example of a scale.

Example 3.2. The function $g_1 : (0, \infty) \times [0, \infty) \rightarrow \mathbb{R}$ defined by $g_1(m, s) = m^s$ is also a scale.

Definition. If $g : H \times [0, \infty) \rightarrow \mathbb{R}$ is a scale, then the *first rescaling* of g is the function $g^\# : H^\# \times [0, \infty) \rightarrow \mathbb{R}$ defined by

$$H^\# = \{2^m \mid m \in H\}$$

$$g^\#(m, s) = 2^{g(\log m, s)}.$$

Note that $g_0^\# = g_1$, where g_0 and g_1 are the scales of Examples 3.1 and 3.2.

If g is a scale, then for all $m \in H^\#$ and $s \in [0, \infty)$, $\log g^\#(m, s) = g(\log m, s)$, which means that a log-log graph of the function $m \mapsto g^\#(m, s)$ is precisely the ordinary graph of the function $m \mapsto g(m, s)$. This is the sense in which $g^\#$ is a rescaling of g .

Lemma 3.3. *If g is a scale, then $g^\#$ is a scale.*

Definition. If $g : H \times [0, \infty) \rightarrow \mathbb{R}$ is a scale, then the *reflection* of g is the function $g^R : H \times [0, \infty) \rightarrow \mathbb{R}$ defined by

$$g^R(m, s) = \begin{cases} m + g(m, 0) - g(m, 1 - s) & \text{if } 0 \leq s \leq 1 \\ g(m, s) & \text{if } s \geq 1. \end{cases}$$

Example 3.4. It is easy to verify that $g_0^R = g_0$ and that

$$g_1^R(m, s) = \begin{cases} m + 1 - m^{1-s} & \text{if } 0 \leq s \leq 1 \\ m^s & \text{if } s \geq 1. \end{cases}$$

for all $m > 0$ and $s \in \mathbb{R}$.

Lemma 3.5. *If g is a scale, then g^R is a scale.*

Notation. For each scale $g : H \times [0, \infty) \rightarrow \mathbb{R}$, we define the function $\Delta g : H \times [0, \infty) \rightarrow \mathbb{R}$ by

$$\Delta g(m, s) = g(m+1, s) - g(m, s).$$

Note that g is the usual finite difference operator, with the proviso that it is applied only to the first variable, m . For $l \in \mathbb{N}$, we also use the extended notation

$$\Delta^l g(m, s) = g(m+l, s) - g(m, s).$$

The following definition is central to scaled dimension.

Definition. Let $g : H \times [0, \infty) \rightarrow \mathbb{R}$ be a scale, and let $s \in [0, \infty)$.

1. A g -scaled s -supergale (briefly, an $s^{(g)}$ -supergale) is a function $d : \{0, 1\}^* \rightarrow [0, \infty)$ such that for all $w \in \{0, 1\}^*$ with $|w| \in H$,

$$d(w) \geq 2^{-\Delta g(|w|, s)} [d(w0) + d(w1)]. \quad (3.1)$$

2. A g -scaled s -gale (briefly, an $s^{(g)}$ -gale) is an $s^{(g)}$ -supergale that satisfies (3.1) with equality for all $w \in \{0, 1\}^*$ such that $|w| \in H$.
3. An s -supergale is an $s^{(g_0)}$ -supergale.
4. An s -gale is an $s^{(g_0)}$ -gale.
5. A *supermartingale* is a 1-supergale.
6. A *martingale* is a 1-gale.

Remark. 1. Martingales were introduced by Lévy [10] and named by Ville [20], who used them in early investigations of random sequences. Martingales were later used extensively by Schnorr [16,17,18,19] in his investigations of random sequences and by Lutz [13,14] in the development of resource-bounded measure. Gales were introduced by Lutz [11,12] in the development of resource-bounded and constructive dimension. Scaled gales are introduced here in order to formulate scaled dimension.

2. Although the martingale condition is usually stated in the form $d(w) = \frac{d(w0) + d(w1)}{2}$, this is a simplification of $d(w)\mu(w) = d(w0)\mu(w0) + d(w1)\mu(w1)$, where $\mu(x) = 2^{-|x|}$ is the measure (probability) of the cylinder $\mathbf{C}_x = \{A \in \mathbf{C} \mid x \subseteq A\}$. Similarly, the s -gale condition $d(w) = 2^{-s} [d(w0) + d(w1)]$ of [11, 12] is a simplification of $d(w)\mu(w)^s = d(w0)\mu(w0)^s + d(w1)\mu(w1)^s$, which is equivalent to

$$d(w) = 2^{-\Delta g_0(|w|, s)} [d(w0) + d(w1)]. \quad (3.2)$$

In defining $s^{(g)}$ -gales we have replaced the scale g_0 in (3.2) by an arbitrary scale g .

3. Condition (3.1) is only required to hold for strings w that are long enough for $g(|w|, s)$ to be defined. In fact, several of the scales $g(m, s)$ used in this paper are not defined for small m . For such a scale g , an $s^{(g)}$ -supergale must satisfy condition (3.1) for all but finitely many strings w , and this is sufficient for our development.

Definition. Let g be a scale, let $s \in [0, \infty)$, and let d be an $s^{(g)}$ -supergale.

1. We say that d *succeeds* on a language $A \in \mathbf{C}$ if $\limsup_{n \rightarrow \infty} d(A[0 \dots n-1]) = \infty$.
2. The *success set* of d is $S^\infty[d] = \{A \in \mathbf{C} \mid d \text{ succeeds on } A\}$.

We now use scaled gales to define scaled dimension.

Notation. Let g be a scale, and let $X \subseteq \mathbf{C}$.

1. $\mathcal{G}^{(g)}(X)$ is the set of all $s \in [0, \infty)$ such that there is an $s^{(g)}$ -gale d for which $X \subseteq S^\infty[d]$.
2. $\hat{\mathcal{G}}^{(g)}(X)$ is the set of all $s \in [0, \infty)$ such that there is an $s^{(g)}$ -supergale d for which $X \subseteq S^\infty[d]$.

Lemma 3.6. *If g is a scale, then for all $X \subseteq \mathbf{C}$, $\mathcal{G}^{(g)}(X) = \hat{\mathcal{G}}^{(g)}(X)$.*

Recall the scale g_0 of Example 3.1. It was proven by Lutz [11] that the following definition is equivalent to the classical definition of Hausdorff dimension in $\mathbf{C}$.

Definition. The *Hausdorff dimension* of a set $X \subseteq \mathbf{C}$ is $\dim_H(X) = \inf \mathcal{G}^{(g_0)}(X)$.

This suggests the following rescaling of Hausdorff dimension in Cantor space.

Definition. If g is a scale, then the *g -scaled dimension* of a set $X \subseteq \mathbf{C}$ is $\dim^{(g)}(X) = \inf \mathcal{G}^{(g)}(X)$.

By Lemma 3.6, this definition would not be altered if we used $\hat{\mathcal{G}}^{(g)}(X)$ in place of $\mathcal{G}^{(g)}(X)$.

We now use resource-bounded scaled gales to develop scaled dimension in complexity classes. In the following, the resource bound Δ may be any one of the classes all, rec, p, p₂, pspace, p₂space, etc., defined in section 2.

Notation. If g is a scale and $X \subseteq \mathbf{C}$, let $\mathcal{G}_\Delta^{(g)}(X)$ be the set of all $s \in [0, \infty)$ such that there is a Δ -computable $s^{(g)}$ -gale d for which $X \subseteq S^\infty[d]$.

Definition. Let g be a scale and $X \subseteq \mathbf{C}$.

1. The *g -scaled Δ -dimension* of X is $\dim_\Delta^{(g)}(X) = \inf \mathcal{G}_\Delta^{(g)}(X)$.
2. The *g -scaled dimension of X in $R(\Delta)$* is $\dim^{(g)}(X \mid R(\Delta)) = \dim_\Delta^{(g)}(X \cap R(\Delta))$.

Note that $\dim_\Delta^{(g)}(X)$ and $\dim^{(g)}(X \mid R(\Delta))$ are defined for every scale g and every set $X \subseteq \mathbf{C}$. Recalling the scale $g_0(m, s) = sm$, we write

$$\dim_\Delta(X) = \dim_\Delta^{(g_0)}(X),$$

$$\dim(X \mid R(\Delta)) = \dim^{(g_0)}(X \mid R(\Delta))$$

and note that these are exactly the resource-bounded dimensions defined by Lutz [11].

Observation 3.7. *Let g be a scale.*

1. For all $X \subseteq Y \subseteq \mathbf{C}$, $\dim_{\Delta}^{(g)}(X) \leq \dim_{\Delta}^{(g)}(Y)$ and $\dim^{(g)}(X \mid R(\Delta)) \leq \dim^{(g)}(Y \mid R(\Delta))$.
2. If Δ and Δ' are resource bounds such that $\Delta \subseteq \Delta'$, then for all $X \subseteq \mathbf{C}$, $\dim_{\Delta'}^{(g)}(X) \leq \dim_{\Delta}^{(g)}(X)$.
3. For all $X \subseteq \mathbf{C}$, $0 \leq \dim^{(g)}(X \mid R(\Delta)) \leq \dim_{\Delta}^{(g)}(X)$.
4. For all $X \subseteq \mathbf{C}$, $\dim^{(g)}(X \mid \mathbf{C}) = \dim_{\text{all}}^{(g)}(X) = \dim^{(g)}(X)$.

The following lemma relates resource-bounded scaled dimension to resource-bounded measure.

Lemma 3.8. *If g is a Δ -computable scale, then for all $X \subseteq \mathbf{C}$, $\dim_{\Delta}^{(g)}(X) < 1 \Rightarrow \mu_{\Delta}(X) = 0$ and $\dim^{(g)}(X \mid R(\Delta)) < 1 \Rightarrow \mu(X \mid R(\Delta)) = 0$.*

Finite subsets of $R(\Delta)$ have scaled dimension 0 in $R(\Delta)$ for Δ -computable scales. This can be extended to show that all “ Δ -countable” subsets of $R(\Delta)$ have scaled dimension 0 in $R(\Delta)$. This implies, for example, that for all pspace-computable scales g and all constants $c \in \mathbb{N}$,

$$\dim^{(g)}(\text{DSPACE}(2^{cn}) \mid \text{ESPACE}) = 0.$$

In contrast, even if $R(\Delta)$ is countable, $R(\Delta)$ does not have scaled dimension 0 in $R(\Delta)$. In fact we have the following.

Theorem 3.9. *If g is a Δ -computable scale, then*

$$\dim^{(g)}(R(\Delta) \mid R(\Delta)) = \dim_{\Delta}^{(g)}(R(\Delta)) = \dim_{\Delta}^{(g)}(\mathbf{C}) = 1$$

We now define a particular family of scales that will be useful for studying the fractal structures of classes that arise naturally in computational complexity.

Definition. 1. For each $i \in \mathbb{N}$, define a_i by the recurrence $a_0 = -\infty, a_{i+1} = 2^{a_i}$.

2. For each $i \in \mathbb{Z}$, define the i^{th} scale $g_i : (a_{|i|}, \infty) \times [0, \infty) \rightarrow \mathbb{R}$ by the following recursion.

- (a) $g_0(m, s) = sm$.
- (b) For $i \geq 0$, $g_{i+1} = g_i^{\#}$.
- (c) For $i < 0$, $g_i = g_{-i}^R$.

Note that each g_i is a scale by Lemmas 3.3 and 3.5. It is easy to see that each g_i is Δ -computable.

Definition. Let $i \in \mathbb{Z}$ and $X \subseteq \mathbf{C}$.

1. The i^{th} dimension of X is $\dim^{(i)}(X) = \dim^{(g_i)}(X)$.
2. The i^{th} Δ -dimension of X is $\dim_{\Delta}^{(i)}(X) = \dim_{\Delta}^{(g_i)}(X)$.
3. The i^{th} dimension of X in $R(\Delta)$ is $\dim^{(i)}(X \mid R(\Delta)) = \dim^{(g_i)}(X \mid R(\Delta))$.

In the spirit of the above definition, $s^{(g_i)}$ -gales are now called $s^{(i)}$ -gales, etc.

Intuitively, if $i < j$, then it is harder to succeed with an $s^{(j)}$ -gale than with an $s^{(i)}$ -gale, so $\dim^{(i)}(X) \leq \dim^{(j)}(X)$. We conclude this section by showing that even more is true.

Theorem 3.10. *Let $i \in \mathbb{Z}$ and $X \subseteq \mathbf{C}$. If $\dim_{\Delta}^{(i+1)}(X) < 1$, then $\dim_{\Delta}^{(i)}(X) = 0$.*

This theorem tells us that for every set $X \subseteq \mathbf{C}$, the sequence of dimensions $\dim_{\Delta}^{(i)}(X)$ for $i \in \mathbb{Z}$ satisfies exactly one of the following three conditions.

- (i) $\dim_{\Delta}^{(i)}(X) = 0$ for all $i \in \mathbb{Z}$.
- (ii) $\dim_{\Delta}^{(i)}(X) = 1$ for all $i \in \mathbb{Z}$.
- (iii) There exist $i^* \in \mathbb{Z}$ such that $\dim_{\Delta}^{(i)}(X) = 0$ for all $i < i^*$ and $\dim_{\Delta}^{(i)}(X) = 1$ for all $i > i^*$.

Intuitively, if condition (iii) holds and $0 < \dim_{\Delta}^{(i^*)}(X) < 1$, then i^* is the “best” order at which to measure the Δ -dimension of X because $\dim_{\Delta}^{(i^*)}(X)$ provides more quantitative information about X than is provided by $\dim_{\Delta}^{(i)}(X)$ for $i \neq i^*$. The following section provides some concrete examples of this phenomenon.

4 Nonuniform Complexity

In this section we examine the scaled dimension of several nonuniform complexity classes in the complexity class ESPACE.

The *circuit-size complexity* of a language $A \subseteq \{0, 1\}^*$ is the function $\text{CS}_A : \mathbb{N} \rightarrow \mathbb{N}$, where $\text{CS}_A(n)$ is the number of gates in the smallest n -input Boolean circuit that decides $A \cap \{0, 1\}^n$. For each function $f : \mathbb{N} \rightarrow \mathbb{N}$, we define the circuit-size complexity classes

$$\text{SIZE}(f) = \{A \in \mathbf{C} \mid (\forall^\infty n) \text{CS}_A(n) \leq f(n)\}$$

and

$$\text{SIZE}_{\text{i.o.}}(f) = \{A \in \mathbf{C} \mid (\exists^\infty n) \text{CS}_A(n) \leq f(n)\}.$$

Given a machine M , a resource-bound $t : \mathbb{N} \rightarrow \mathbb{N}$, a language $L \subseteq \{0, 1\}^*$, and a natural number n , the t -space-bounded Kolmogorov complexity of $L_{=n}$ relative to M is

$$\text{KS}_M^t(L_{=n}) = \min \left\{ |\pi| \mid M(\pi, n) = \chi_{L_{=n}} \text{ in } \leq t(2^n) \text{ space} \right\},$$

i.e., the length of the shortest program π such that M , on input (π, n) , outputs the characteristic string of $L_{=n}$ and halts without using more than $t(2^n)$ workspace. Similarly the t -time-bounded Kolmogorov complexity of $L_{=n}$ relative to M is

$$\text{KT}_M^t(L_{=n}) = \min \left\{ |\pi| \mid M(\pi, n) = \chi_{L_{=n}} \text{ in } \leq t(2^n) \text{ time} \right\},$$

Well-known simulation techniques show that there exists a machine U which is *optimal* in the sense that for each machine M there is a constant c such that for all t , L and n we have

$$\text{KS}_U^{ct+c}(L_{=n}) \leq \text{KS}_M^t(L_{=n}) + c$$

and

$$\text{KT}_U^{ct \log t + c}(L_{=n}) \leq \text{KT}_M^t(L_{=n}) + c.$$

As usual, we fix such a universal machine and omit it from the notation.

For each resource bound $t : \mathbb{N} \rightarrow \mathbb{N}$ and function $f : \mathbb{N} \rightarrow \mathbb{N}$ we define the following complexity classes.

$$\begin{aligned} \text{KS}^t(f) &= \{L \in \mathbf{C} \mid (\forall^\infty n) \text{KS}^t(L_{=n}) < f(n)\} \\ \text{KT}^t(f) &= \{L \in \mathbf{C} \mid (\forall^\infty n) \text{KT}^t(L_{=n}) < f(n)\} \\ \text{KS}_{\text{i.o.}}^t(f) &= \{L \in \mathbf{C} \mid (\exists^\infty n) \text{KS}^t(L_{=n}) < f(n)\} \\ \text{KT}_{\text{i.o.}}^t(f) &= \{L \in \mathbf{C} \mid (\exists^\infty n) \text{KT}^t(L_{=n}) < f(n)\} \end{aligned}$$

Our first lemma provides inclusion relationships between some SIZE and KS classes defined using the scales.

Lemma 4.1. *There exists a constant $c_0 \in \mathbb{N}$ such that for all $i > 0$, $\alpha \in [0, 1]$, and $\epsilon > 0$, $\text{SIZE}(g_i(2^n, \alpha)) \subseteq \text{KS}^{c_0 n + c_0}(g_i(2^n, \alpha + \epsilon))$.*

The next two lemmas present positive-order dimension lower bounds for some SIZE classes.

Lemma 4.2. *For all $i \geq 1$ and $\alpha \in (0, 1]$, for all sufficiently large n there are at least $2^{g_i(2^n, \alpha)}$ different sets $B \subseteq \{0, 1\}^n$ that are decided by Boolean circuits of fewer than $g_i(2^n, \alpha)$ gates.*

Lemma 4.3. *For every $i \geq 1$, for every real $\alpha \in [0, 1]$,*

$$\dim^{(i)}(\text{SIZE}(g_i(2^n, \alpha)) \mid \text{SPACE}) \geq \alpha.$$

We now give positive-order scaled dimension upper bounds for some KS classes defined using the scales.

Lemma 4.4. *For all $i \geq 0$, for any polynomial q , and any $\alpha \in [0, 1]$,*
 $\dim_{\text{space}}^{(i)}(\text{KS}^q(g_i(2^n, \alpha))) \leq \alpha.$

Now we are able to present exact scaled-dimension results for circuit-size complexity classes defined in terms of the positive scales. Note that in each case, we have obtained the “best” order at which to measure the dimension of the class.

Theorem 4.5. *Let $i \geq 1$ and $\alpha \in [0, 1]$. Then $\dim^{(i)}(\text{SIZE}(g_i(2^n, \alpha)) \mid \text{SPACE}) = \alpha$. In particular,*

$$\dim^{(1)}(\text{SIZE}(2^{\alpha n}) \mid \text{SPACE}) = \alpha$$

and

$$\dim^{(2)}(\text{SIZE}(2^{n^\alpha}) \mid \text{SPACE}) = \alpha.$$

Proof. By Lemma 4.1 we have $\text{SIZE}(g_i(2^n, \alpha)) \subseteq \text{KS}^{c_0 n + c_0}(2^n, \alpha + \epsilon)$ for all $\epsilon > 0$. The theorem then follows from Lemmas 4.3 and 4.4.

At this point, we could use Lemmas 4.1 and 4.3 to give scaled dimension lower bounds for some KS classes defined using the positive scales. Also, proving an analogue of Lemma 4.1 for KT complexity will yield scaled dimension lower bounds for similar KT classes. However, taking a direct approach to these lower bounds yields slightly stronger results for KT complexity. In the next lemma we do this, and we also obtain scaled dimension lower bounds for all orders (not just the positive ones) at the same time.

Lemma 4.6. *There exist constants $c_1, c_2 \in \mathbb{N}$ such that for all $i \in \mathbb{Z}$ and $\alpha \in [0, 1]$,*

$$\dim^{(i)}(\text{KT}^{c_1 n \log n + c_1}(g_i(2^n, \alpha)) | \text{SPACE}) \geq \alpha$$

and

$$\dim^{(i)}(\text{KS}^{c_2 n + c_2}(g_i(2^n, \alpha)) | \text{SPACE}) \geq \alpha.$$

Now we can state exact scaled dimensions results for some KS and KT classes in the 0^{th} - and positive-order scales.

Theorem 4.7. *Let $i \geq 0$, $\alpha \in [0, 1]$, and $t : \mathbb{N} \rightarrow \mathbb{N}$ be a polynomially-bounded function. Let c_1 and c_2 be as in Lemma 4.6. If $t(n) \geq c_1 n \log n + c_1$ almost everywhere, then $\dim^{(i)}(\text{KT}^t(g_i(2^n, \alpha)) | \text{SPACE}) = \alpha$, and if $t(n) \geq c_2 n + c_2$ almost everywhere, then $\dim^{(i)}(\text{KS}^t(g_i(2^n, \alpha)) | \text{SPACE}) = \alpha$. In particular, for any polynomial $q(n) \geq n^2$,*

$$\dim^{(0)}(\text{KT}^q(2^{\alpha n}) | \text{SPACE}) = \dim^{(0)}(\text{KS}^q(2^{\alpha n}) | \text{SPACE}) = \alpha,$$

and

$$\dim^{(1)}(\text{KT}^q(2^{n^\alpha}) | \text{SPACE}) = \dim^{(0)}(\text{KS}^q(2^{n^\alpha}) | \text{SPACE}) = \alpha.$$

Proof. This follows immediately from Lemmas 4.4 and 4.6.

Now we give an upper bound on the scaled dimension of some KS classes for the negative scales. In the negative orders, we are able to work with classes of the infinitely-often type.

Lemma 4.8. *Let $i \leq -1$, q be a polynomial, and $\alpha \in [0, 1]$. Then*

$$\dim_{\text{pspace}}^{(i)}(\text{KS}_{\text{i.o.}}^q(g_i(2^n, \alpha))) \leq \alpha.$$

Our final theorem is an exact scaled dimension result analogous to Theorem 4.7 for the negative scales. Here the dimension is invariant if we change the type of the class from almost-everywhere to infinitely-often.

Theorem 4.9. *Let $i \leq -1$, $\alpha \in [0, 1]$, and $t : \mathbb{N} \rightarrow \mathbb{N}$ be a polynomially-bounded function. Let c_1 and c_2 be as in Lemma 4.6. If $t(n) \geq c_1 n \log n + c_1$ almost everywhere, then*

$$\dim^{(i)}(\text{KT}^t(g_i(2^n, \alpha))|\text{SPACE}) = \dim^{(i)}(\text{KT}_{\text{i.o.}}^t(g_i(2^n, \alpha))|\text{SPACE}) = \alpha,$$

and if $t(n) \geq c_2 n + c_2$ almost everywhere,

$$\dim^{(i)}(\text{KS}^t(g_i(2^n, \alpha))|\text{SPACE}) = \dim^{(i)}(\text{KS}_{\text{i.o.}}^t(g_i(2^n, \alpha))|\text{SPACE}) = \alpha.$$

In particular, for any polynomial $q(n) \geq n^2$,

$$\begin{aligned} \dim^{(-1)}(\text{KT}^q(2^n(1 - 2^{-\alpha n}))|\text{SPACE}) \\ = \dim^{(-1)}(\text{KS}^q(2^n(1 - 2^{-\alpha n}))|\text{SPACE}) = \alpha. \end{aligned}$$

Proof. This follows from Lemmas 4.6 and 4.8.

References

1. K. Ambos-Spies, W. Merkle, J. Reimann, and F. Stephan. Hausdorff dimension in exponential time. In *Proceedings of the 16th IEEE Conference on Computational Complexity*, pages 210–217, 2001.
2. K. B. Athreya, J. M. Hitchcock, J. H. Lutz, and E. Mayordomo. Effective strong dimension, algorithmic information, and computational complexity. Technical Report cs.CC/0211025, Computing Research Repository, 2002.
3. H.G. Eggleston. The fractional dimension of a set defined by decimal properties. *Quarterly Journal of Mathematics*, Oxford Series 20:31–36, 1949.
4. K. Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. John Wiley & Sons, 1990.
5. L. Fortnow and J. H. Lutz. Prediction and dimension. In *Proceedings of the 15th Annual Conference on Computational Learning Theory*, pages 380–395, 2002.
6. F. Hausdorff. Dimension und äusseres Mass. *Mathematische Annalen*, 79:157–179, 1919.
7. J. M. Hitchcock. Fractal dimension and logarithmic loss unpredictability. *Theoretical Computer Science*. To appear.
8. J. M. Hitchcock. MAX3SAT is exponentially hard to approximate if NP has positive dimension. *Theoretical Computer Science*, 289(1):861–869, 2002.
9. D. W. Juedes and J. H. Lutz. Completeness and weak completeness under polynomial-size circuits. *Information and Computation*, 125:13–31, 1996.
10. P. Lévy. *Théorie de l'Addition des Variables Aleatoires*. Gauthier-Villars, 1937 (second edition 1954).
11. J. H. Lutz. Dimension in complexity classes. *SIAM Journal on Computing*. To appear. Available as Technical Report cs.CC/0203016, Computing Research Repository, 2002.
12. J. H. Lutz. The dimensions of individual strings and sequences. *Information and Computation*. To appear. Available as Technical Report cs.CC/0203017, Computing Research Repository, 2002.

13. J. H. Lutz. Almost everywhere high nonuniform complexity. *Journal of Computer and System Sciences*, 44:220–258, 1992.
14. J. H. Lutz. Resource-bounded measure. In *Proceedings of the 13th IEEE Conference on Computational Complexity*, pages 236–248, 1998.
15. C. A. Rogers. *Hausdorff Measures*. Cambridge University Press, 1998. Originally published in 1970.
16. C. P. Schnorr. Klassifikation der Zufallsgesetze nach Komplexität und Ordnung. *Z. Wahrscheinlichkeitstheorie verw. Geb.*, 16:1–21, 1970.
17. C. P. Schnorr. A unified approach to the definition of random sequences. *Mathematical Systems Theory*, 5:246–258, 1971.
18. C. P. Schnorr. Zufälligkeit und Wahrscheinlichkeit. *Lecture Notes in Mathematics*, 218, 1971.
19. C. P. Schnorr. Process complexity and effective random tests. *Journal of Computer and System Sciences*, 7:376–388, 1973.
20. J. Ville. *Étude Critique de la Notion de Collectif*. Gauthier–Villars, Paris, 1939.

Quantum Search on Bounded-Error Inputs

Peter Høyer^{1,*}, Michele Mosca², and Ronald de Wolf^{3,**}

¹ Dept. of Computer Science, Univ. of Calgary, Alberta, Canada.

`hoyer@cpsc.ucalgary.ca`

² Dept. of Combinatorics & Optimization, Univ. of Waterloo, Ontario, Canada.

`mmosca@cacr.math.uwaterloo.ca`

³ CWI. Kruislaan 413, 1098 SJ, Amsterdam, The Netherlands. `rdewolf@cwi.nl`

Abstract. Suppose we have n algorithms, quantum or classical, each computing some bit-value *with bounded error probability*. We describe a quantum algorithm that uses $O(\sqrt{n})$ repetitions of the base algorithms and with high probability finds the index of a 1-bit among these n bits (if there is such an index). This shows that it is not necessary to first significantly reduce the error probability in the base algorithms to $O(1/\text{poly}(n))$ (which would require $O(\sqrt{n} \log n)$ repetitions in total). Our technique is a recursive interleaving of amplitude amplification and error-reduction, and may be of more general interest. Essentially, it shows that quantum amplitude amplification can be made to work also with a *bounded-error* verifier. As a corollary we obtain optimal quantum upper bounds of $O(\sqrt{N})$ queries for all constant-depth AND-OR trees on N variables, improving upon earlier upper bounds of $O(\sqrt{N} \text{polylog}(N))$.

1 Introduction

One of the main successes of quantum computing is Grover's algorithm [10,7]. It can search an n -element space in $O(\sqrt{n})$ steps, which is quadratically faster than any classical algorithm. The algorithm assumes oracle access to the elements in the space, meaning that in unit time it can decide whether the i th element is a solution to its search problem or not. In some more realistic settings we can efficiently make such an oracle ourselves. For instance, if we want to decide satisfiability of an m -variable Boolean formula, the search space is the set of all $n = 2^m$ truth assignments, and we can efficiently decide whether a given assignment satisfies the formula. However, in these cases the decision is made without any error probability. In this paper we study the complexity of quantum search if we only have *bounded-error* access to the elements in the space.

More precisely, suppose that among n Boolean values $f_1, \dots, f_n$ we want to find a solution (if one exists), i.e., an index j such that $f_j = 1$. For each i we have at our disposal an algorithm F_i that computes the bit f_i with two-sided error: if f_i

* Supported in part by the Alberta Ingenuity Fund and the Pacific Institute for the Mathematical Sciences.

** This research was (partially) funded by projects QAIP (IST-1999-11234) and RESQ (IST-2001-37559) of the IST-FET programme of the EC.

is 1 then the algorithm outputs 1 with probability, say, at least 9/10, and if $f_i = 0$ then it outputs 0 with probability at least 9/10. Grover's algorithm is no longer applicable in this bounded-error setting, at least not directly, because the errors in each step will quickly add up to something uncontrollably large. Accordingly, we need to do something different to get a quantum search algorithm that works here. We will measure the complexity of our quantum search algorithms by the number of times they call the underlying algorithms F_i . Clearly, the $\Omega(\sqrt{n})$ lower bound for the standard error-less search problem, due to Bennett, Bernstein, Brassard, and Vazirani [4], also applies to our more general setting. Our aim is to give a matching upper bound.

An obvious but sub-optimal quantum search algorithm is the following. By repeating F_i $k = O(\log n)$ times and outputting the majority value of the k outcomes, we can compute f_i with error probability at most $1/100n$. If we then copy the answer to a safe place and reverse the computation to clean up (most of) the workspace, then we get something that is sufficiently “close” to perfect oracle access to the f_i bits to just treat it as such. Now we can apply Grover's algorithm on top of this, and because quantum computational errors add linearly [5], the overall difference with perfect oracle access will be negligibly small. This solves the bounded-error quantum search problem using $O(\sqrt{n} \log n)$ repetitions of the F_i 's, which is an $O(\log n)$ -factor worse than the lower bound. Below we will refer to this algorithm as “the simple search algorithm”.

A relatively straightforward improvement over the simple search algorithm is the following. Partition the search space into $n/\log^2 n$ blocks of size $\log^2 n$ each. Pick one such block at random. We can find a potential solution (an index j in the chosen block such that $f_j = 1$, if there is such a j) in complexity $O(\log n \log \log n)$ using the simple search algorithm, and then verify that it is indeed 1 with error probability at most $1/n$ using another $O(\log n)$ invocations of F_j . Applying Grover search on the space of all $n/\log^2 n$ blocks, we obtain an algorithm with complexity $O(\sqrt{n/\log^2 n}) \cdot O(\log n \log \log n + \log n) = O(\sqrt{n} \log \log n)$.

A further improvement comes from doing the splitting recursively: we can use the improved upper bound to do the computation of the “inner” blocks, instead of the simple search algorithm. Using $T(n)$ to denote the complexity on search space of size n , this gives us the recursion

$$T(n) \leq d \left(T(\log^2 n) \sqrt{\frac{n}{\log^2 n}} + \log n \right)$$

for some constant $d > 0$. This recursion resolves to complexity $O(\sqrt{n} \cdot c^{\log^* n})$ for some constant $c > 0$. It is similar to (and inspired by) the communication complexity protocol for the disjointness problem of Høyer and de Wolf [11].

Apart from being rather messy, this improved algorithm is still not optimal. The main result of this paper is to give a relatively clean algorithm that uses the optimal number $O(\sqrt{n})$ of repetitions to solve the bounded-error search problem. Our algorithm uses a kind of “carrot-and-stick” approach that may be of more general interest. Roughly speaking, it starts with a uniform superposition of all F_i . It then amplifies all branches of the computation that give answer 1. These

branches include solutions, but they also include “false positives”: branches corresponding to the $1/10$ error probability of F_i ’s where $f_i = 0$. We then “push these back” by testing whether a 1-branch is a real positive or a false one (i.e., whether $f_i = 1$ or not) and removing most of the false ones. Interleaving these amplify and push-back steps properly, we can amplify the weight of the solutions to a constant using $O(\sqrt{n})$ repetitions. At this point we just do a measurement, see a potential solution j , and verify it classically by running F_j a few times.

As an application of our bounded-error quantum search algorithm, in Section 4 we give optimal quantum algorithms for constant-depth AND-OR trees in the query complexity setting. For any constant d , we need only $O(\sqrt{N})$ queries for the d -level AND-OR tree, improving upon the earlier $O(\sqrt{N}(\log N)^{d-1})$ algorithms of Buhrman, Cleve, and Wigderson [9]. Matching lower bounds of $\Omega(\sqrt{N})$ were already shown for such AND-OR trees, using Ambainis’ quantum adversary method [1,2]. Finally, in Section 5 we indicate how the ideas presented here can be cast more generally in terms of amplitude amplification.

2 Preliminaries

Here we briefly sketch the basics and notation of quantum computation, referring to the book by Nielsen and Chuang [12] for more detail. An m -qubit state is a linear combination of all classical m -bit states

$$|\phi\rangle = \sum_{i \in \{0,1\}^m} \alpha_i |i\rangle,$$

where $|i\rangle$ denotes the basis state i (a classical m -bit string), the *amplitude* α_i is a complex number, and $\sum_i |\alpha_i|^2 = 1$. We view $|\phi\rangle$ as a 2^m -dimensional column vector. A measurement of state $|\phi\rangle$ will give $|i\rangle$ with probability $|\alpha_i|^2$, and the state will then collapse to the observed $|i\rangle$. A non-measuring quantum operation corresponds to applying a unitary (= linear and norm-preserving) transformation U to the vector of amplitudes. If $|\phi\rangle$ and $|\psi\rangle$ are quantum states on m and m' qubits, respectively, then the two-register state $|\phi\rangle \otimes |\psi\rangle = |\phi\rangle|\psi\rangle$ corresponds to the $2^{m+m'}$ -dimensional vector that is the tensor product of $|\phi\rangle$ and $|\psi\rangle$.

The setting of query complexity is as follows. For input $x \in \{0,1\}^n$, a *query* corresponds to the unitary transformation O that maps $|i, b, z\rangle \rightarrow |i, b \oplus x_i, z\rangle$. Here $i \in [n]$ and $b \in \{0,1\}$; the z -part corresponds to the workspace, which is not affected by the query. A T -query quantum algorithm has the form $A = U_T O U_{T-1} \cdots O U_1 O U_0$, where the U_k are unitary transformations, independent of x . This A depends on x only via the T applications of O . The algorithm starts in initial all-zero state $|\mathbf{0}\rangle$ and its output (which is a random variable) is obtained from observing some dedicated part of the final superposition $A|\mathbf{0}\rangle$.

3 Optimal Quantum Algorithm for Bounded-Error Search

In this section we describe our quantum algorithm for bounded-error search. The following two facts generalize, respectively, the Grover search and the error-reduction used in the algorithms we sketched in the introduction.

Fact 1 (Amplitude amplification [8]) *Let S_0 be the unitary that puts a ‘-’ in front of the all-zero state $|0\rangle$, and S_1 be the unitary that puts a ‘-’ in front of all basis states whose last qubit is $|1\rangle$. Let $A|0\rangle = \sin(\theta)|\phi_1\rangle|1\rangle + \cos(\theta)|\phi_0\rangle|0\rangle$ where angle θ is such that $0 \leq \theta \leq \pi/2$ and $\sin^2(\theta)$ equals the probability that a measurement of the last register of state $A|0\rangle$ yields a ‘1’. Set $G = -AS_0A^{-1}S_1$. Then $GA|0\rangle = \sin(3\theta)|\phi_1\rangle|1\rangle + \cos(3\theta)|\phi_0\rangle|0\rangle$.*

Amplitude amplification is a process that is used in many quantum algorithms to increase the success probability. Amplitude amplification effectively implements a rotation by an angle 2θ in a two-dimensional space (a space different from the Hilbert space acted upon) spanned by $|\phi_1\rangle|1\rangle$ and $|\phi_0\rangle|0\rangle$. Note that we can always apply amplitude amplification regardless of whether the angle θ is known to us or not.

Fact 2 (Error-reduction) *Suppose $A|0\rangle = \sqrt{p}|\phi_b\rangle|b\rangle + \sqrt{1-p}|\phi_{1-b}\rangle|1-b\rangle$, where $b \in \{0,1\}$ and $p \geq 9/10$. Then using $O(\log(1/\varepsilon))$ applications of A and majority-voting, we can build a unitary E such that $E|0\rangle = \sqrt{q}|\psi_b\rangle|b\rangle + \sqrt{1-q}|\psi_{1-b}\rangle|1-b\rangle$ with $q \geq 1 - \varepsilon$, and $|\psi_{b/1-b}\rangle$ possibly of larger dimension than $|\phi_{b/1-b}\rangle$ (because of extra workspace).*

We will recursively interleave these two facts to get a quantum search algorithm that searches the space $f_1, \dots, f_n \in \{0,1\}$. We assume each f_i is computed by unitary F_i with success probability at least $9/10$. Let $\Gamma = \{j : f_j = 1\}$ be the set of solutions, and $t = |\Gamma|$ its size (which is unknown to our algorithm). The goal is to find an element in Γ if $t \geq 1$, and to output ‘no solutions’ if $t = 0$.

We will build an algorithm that has a superposition of all $j \in [n]$ in its first register, a growing second register that contains workspace and other junk, and a 1-qubit third register indicating whether something is deemed a solution or not. The algorithm will successively increase the weight of the basis states that simultaneously have a solution in the first register and a 1 in the third.

Consider an algorithm A that runs all F_i once in superposition, producing the state $A|0\rangle$, which we rewrite as

$$\frac{1}{\sqrt{n}} \sum_{i=1}^n |i\rangle \left(\sqrt{p_i} |\psi_{i,1}\rangle |1\rangle + \sqrt{1-p_i} |\psi_{i,0}\rangle |0\rangle \right) = \sin(\theta) |\phi_1\rangle |1\rangle + \cos(\theta) |\phi_0\rangle |0\rangle,$$

where p_i is the probability that F_i outputs 1, the states $|\psi_{i,b}\rangle$ describe the workspace of the F_i , and $\sin(\theta)^2 = \sum_{i=1}^n p_i \geq 9t/10n$.

The idea is to apply a round of amplitude amplification to A to amplify the $|1\rangle$ -part from $\sin(\theta)$ to $\sin(3\theta)$. This will amplify both the good states $|j\rangle|1\rangle$ for $j \in \Gamma$ and the “false positives” $|j\rangle|1\rangle$ for $j \notin \Gamma$ by a factor of $\sin(3\theta)/\sin(\theta) \approx 3$

(here we didn't write the second register). We then apply an error-reduction step to reduce the amplitude of the false positives, setting "most" of its third register to 0. These two steps together form a new algorithm that puts almost 3 times as much amplitude on the solutions as A does, and that puts less amplitude on the false positives than A . We then repeat the amplify-reduce steps on this new algorithm to get an even better algorithm, and so on.

Let us be more precise. Our algorithm will consist of a number of rounds. In round k we will have a unitary A_k that produces

$$A_k|\mathbf{0}\rangle = \alpha_k|\Gamma_k\rangle|1\rangle + \beta_k|\overline{\Gamma}_k\rangle|1\rangle + \sqrt{1 - \alpha_k^2 - \beta_k^2}|H_k\rangle|0\rangle,$$

where α_k, β_k are non-negative reals, $|\Gamma_k\rangle$ is a unit vector whose first register only contains $j \in \Gamma$, $|\overline{\Gamma}_k\rangle$ is a unit vector whose first register only contains $j \notin \Gamma$, and $|H_k\rangle$ is a unit vector. If we measure the first register of the above state, we will see a solution (i.e. some $j \in \Gamma$) with probability at least α_k^2 . A_1 is the above algorithm A , which runs the F_i in superposition. Initially, $\alpha_1^2 \geq 9t/10n$ since each solution contributes at least $9/10n$. We want to make the good amplitude α_k grow by a factor of almost 3 in each round.

Amplitude amplification step. For each round k , define $\theta_k \in [0, \pi/2]$ by $\sin(\theta_k)^2 = \alpha_k^2 + \beta_k^2$. Applying amplitude amplification ($G_k = -A_k S_0 A_k^{-1} S_1$) gives us the state $G_k A_k |\mathbf{0}\rangle$, which we may write as

$$\frac{\sin(3\theta_k)}{\sin(\theta_k)}\alpha_k|\Gamma_k\rangle|1\rangle + \frac{\sin(3\theta_k)}{\sin(\theta_k)}\beta_k|\overline{\Gamma}_k\rangle|1\rangle + \sqrt{1 - \left(\frac{\sin(3\theta_k)}{\sin(\theta_k)}\right)^2}(\alpha_k^2 + \beta_k^2)|H_k\rangle|0\rangle.$$

We applied A_k twice and A_k^{-1} once, so the complexity goes up by a factor of 3.

Error-reduction step. Conditional on the qubit in the third register being 1, the error-reduction step E_k now does majority voting on $O(k)$ runs of the F_j (for all j in superposition) to decide with error at most $1/2^{k+5}$ whether $f_j = 1$. It adds one 0-qubit as the new third register and maps (ignoring its workspace, which is added to the second register)

$$\begin{aligned} E_k|j\rangle|1\rangle|0\rangle &= a_{jk}|j\rangle|1\rangle|1\rangle + \sqrt{1 - a_{jk}^2}|j\rangle|1\rangle|0\rangle \\ E_k|j\rangle|0\rangle|0\rangle &= |j\rangle|0\rangle|0\rangle \end{aligned}$$

where $a_{jk}^2 \geq 1 - 1/2^{k+5}$ if $f_j = 1$ and $a_{jk}^2 \leq 1/2^{k+5}$ if $f_j = 0$. This way, E_k removes most of the false positives.

Putting $A_{k+1} = E_k G_k A_k$ and defining α_{k+1} , β_{k+1} , $|\Gamma_{k+1}\rangle$, $|\overline{\Gamma}_{k+1}\rangle$, and $|H_{k+1}\rangle$ appropriately, we now have

$$A_{k+1}|\mathbf{0}\rangle = \alpha_{k+1}|\Gamma_{k+1}\rangle|1\rangle + \beta_{k+1}|\overline{\Gamma}_{k+1}\rangle|1\rangle + \sqrt{1 - \alpha_{k+1}^2 - \beta_{k+1}^2}|H_{k+1}\rangle|0\rangle.$$

Here the second register has grown by the workspace used in the error-reduction step E_k , as well as by the qubit that previously was the third register. The good amplitude has grown in the process:

$$\alpha_{k+1} \geq \alpha_k \frac{\sin(3\theta_k)}{\sin(\theta_k)} \sqrt{1 - 1/2^{k+5}}.$$

Since $x - x^3/6 \leq \sin(x) \leq x$, we have

$$\frac{\sin(3\theta_k)}{\sin(\theta_k)} \geq 3 - 9\theta_k^2/2.$$

Accordingly, as long as θ_k is small, α_k will grow by a factor of almost 3 in each round. On the other hand, the weight of the false positives goes down rapidly:

$$\beta_{k+1} \leq \beta_k \frac{\sin(3\theta_k)}{\sin(\theta_k)} \frac{1}{\sqrt{2^{k+5}}}.$$

We now analyze the number m of rounds that we need to make the good amplitude large. In general, we have $\sin(\theta_k)^2 = \alpha_k^2 + \beta_k^2$, hence $\theta_k^2 \leq 2(\alpha_k^2 + \beta_k^2)$ for the domain we are interested in. Here $\alpha_k^2 \leq 9^{k-1}\alpha_1^2$ and $\beta_k^2 \leq \frac{1}{10}(9/2^6)^{k-1}$. Note

$$\begin{aligned} \sum_{k=1}^{m-1} \theta_k^2 &\leq 2 \sum_{k=1}^{m-1} \alpha_k^2 + \beta_k^2 \\ &\leq 2 \sum_{k=1}^{m-1} 9^{k-1} \alpha_1^2 + 2 \sum_{k=1}^{m-1} \frac{1}{10} (9/2^6)^{k-1} \\ &\leq 2 \cdot 9^{m-1} \alpha_1^2 + 1/4. \end{aligned}$$

Therefore, m rounds of the above process amplifies the good amplitude α_k to

$$\begin{aligned} \alpha_m &\geq \alpha_1 \prod_{k=1}^{m-1} \frac{\sin(3\theta_k)}{\sin(\theta_k)} \sqrt{1 - 1/2^{k+5}} \\ &\geq \alpha_1 \prod_{k=1}^{m-1} (3 - 9\theta_k^2/2) (1 - 1/2^{k+5}) \\ &= \alpha_1 3^{m-1} \prod_{k=1}^{m-1} (1 - 3\theta_k^2/2) (1 - 1/2^{k+5}) \\ &\geq \alpha_1 3^{m-1} \left(1 - \frac{3}{2} \sum_{k=1}^{m-1} \theta_k^2 - \sum_{k=1}^{m-1} \frac{1}{2^{k+5}} \right) \\ &\geq \alpha_1 3^{m-1} \left(1 - \frac{3}{2} (2 \cdot 9^{m-1} \alpha_1^2 + 1/4) - 1/16 \right) \\ &\geq \alpha_1 3^{m-1} (1/2 - 3 \cdot 9^{m-1} \alpha_1^2). \end{aligned}$$

In particular, whenever the (unknown) number t of solutions lies in the interval $[n/9^{m+1}, n/9^m]$, equivalently $9^m \in [n/9t, n/t]$, then we have

$$\frac{1}{3^m \sqrt{10}} \leq \sqrt{\frac{9t}{10n}} \leq \alpha_1 \leq \sqrt{\frac{t}{n}} \leq \frac{1}{3^m}.$$

This implies

$$\alpha_m \geq 0.04,$$

so the probability of seeing a solution after m rounds is at least 0.0016. By repeating this classically a constant number of times, say 1000 times, we can bring the success probability close to 1 (note to avoid confusion: these 1000 repetitions are not part of the definition of A_m itself).

The complexity C_k of the operation A_k , in terms of number of repetitions of the F_i algorithms, is given by the recursion

$$C_1 = 1 \text{ and } C_{k+1} = 3C_k + O(k),$$

where the $3C_k$ is the cost of amplitude amplification and $O(k)$ is the cost of error-reduction. This implies $C_m = O(\sum_{k=1}^{m-1} k \cdot 3^{m-k-1}) = O(3^m)$.

We now give the full algorithm when the number of solutions is unknown:

Algorithm: Quantum search on bounded-error inputs

1. for $m = 0$ to $\lceil \log_9(n) \rceil - 1$ do:
 - a) run A_m 1000 times
 - b) verify the 1000 measurement results, each by $O(\log n)$ runs of the corresponding F_j
 - c) if a solution has been found, then output a solution and stop
2. Output ‘no solutions’

This finds a solution with high probability if one exists. The complexity is

$$\sum_{m=0}^{\lceil \log_9(n) \rceil - 1} 1000 \cdot O(3^m) + 1000 \cdot O(\log n) = O(3^{\log_9(n)}) = O(\sqrt{n}).$$

If we know that there is at least one solution but we don’t know how many there are, then, using a modification of our algorithm as in [7], we can find a solution using an expected number of repetitions in $O(\sqrt{N/t})$, where t is the (unknown) number of solutions. This is quadratically faster than classically, and optimal for any quantum algorithm.

4 Optimal Upper Bounds for AND-OR Trees

A d -level *AND-OR tree* on N Boolean variables is a Boolean function that is described by a depth- $d-1$ tree with interleaved ORs and ANDs on the nodes and the N input variables as leaves. More precisely, a 0-level AND-OR tree is just an

input variable, and if $f_1, \dots, f_n$ all are d -level AND-OR trees on m variables, each with an AND (resp. OR) as root, then $\text{OR}(f_1, \dots, f_n)$ (resp. AND) is a $(d+1)$ -level AND-OR tree on $N = nm$ variables. AND-OR trees can be converted easily into OR-AND trees and vice versa using De Morgan's laws, if we allow negations to be added to the tree.

Consider the two-level tree on $N = n^2$ variables with an OR as root, ANDs as its children, and fanout n in both levels. Each AND-subtree can be quantum computed by Grover's algorithm with one-sided error using $O(\sqrt{n})$ queries (we let Grover search for a '0', and output 1 if we don't find any), and the value of the OR-AND tree is just the OR of those n values. Accordingly, the construction of the previous section gives an $O(\sqrt{n} \cdot \sqrt{n}) = O(\sqrt{N})$ algorithm with *two*-sided error. This is optimal up to a constant factor [1].

More generally, for d -level AND-OR trees we can apply the above algorithm recursively to obtain an algorithm with $O(c^{d-1}\sqrt{N})$ queries. Here c is the constant hidden in the $O(\cdot)$ of the result of the previous section. For each fixed d , this complexity is $O(\sqrt{N})$, which is optimal up to a constant factor [2]. It improves upon the $O(\sqrt{N}(\log N)^{d-1})$ algorithm given in [9].

Our query complexity upper bound also implies that the minimal degree among N -variate polynomials approximating AND-OR is $O(\sqrt{N})$ [3]. Whether this upper bound on the degree is optimal remains open. The best known lower bound for the 2-level case is $\Omega(N^{1/4}\sqrt{\log N})$ [13].

5 Amplitude Amplification with Imperfect Verifier

In this section we view our construction in a more general light.

Suppose we are given some classical randomized algorithm A that succeeds in solving some problem with probability p . In addition, we are given a Boolean function χ that takes as input an output from algorithm A , and outputs whether it is a solution or not. Then, we may find a solution to our problem by repetition. We first apply algorithm A , obtaining some candidate solution, which we then give as input to the verifier χ . If χ outputs that the candidate indeed is a solution, we output it and stop, and otherwise we repeat the process by reapplying A . The probability that this process terminates by outputting a solution within the first $\Theta(\frac{1}{p})$ iterations of the loop, is lower bounded by a constant.

A quantum analogue of boosting the probability of success is to boost the *amplitude* of being in a certain subspace of a Hilbert space. Thus far, amplitude amplification [6] has assumed that we are given a perfect verifier χ : whenever a candidate solution is found, we can determine with certainty whether it is a solution or not. Formally, we model this by letting χ be computed by a deterministic classical subroutine or an exact quantum subroutine.

The main result of this paper may be viewed as an adaptation of amplitude amplification to the situation where the verifier is *not* perfect, but sometimes makes mistakes. Instead of a deterministic subroutine for computing χ , we are given a bounded-error randomized subroutine, and instead of an exact quantum subroutine, we are given a bounded-error quantum subroutine. Previously, the only known technique for handling such cases has been by straightforward

simulation of a perfect verifier: construct a subroutine for computing χ with error $\frac{1}{2^k}$ by repeating a given bounded-error subroutine of order $\Theta(k)$ times and then use majority voting. Using such direct simulations, we may construct good but sub-optimal quantum algorithms, like the $O(\sqrt{n} \log n)$ query algorithm for quantum search of the introduction. Here, we have introduced a modification of the amplitude amplification process that allows us to efficiently deal with imperfect verifiers. Essentially, our result says that imperfect verifiers are as good as perfect verifiers (up to a constant multiplicative factor in the complexity).

Acknowledgments. We thank Richard Cleve for useful discussions, as well as for hosting MM and RdW at the University of Calgary, where most of this work was done.

References

1. A. Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of 32nd ACM STOC*, pages 636–643, 2000.
2. H. Barnum and M. Saks. A lower bound on the quantum query complexity of read-once functions. quant-ph/0201007, 3 Jan 2002.
3. R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. In *Proceedings of 39th IEEE FOCS*, pages 352–361, 1998.
4. C. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.
5. E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.
6. 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’97)*, pages 12–23, 1997.
7. M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4–5):493–505, 1998.
8. G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. In Lomonaco, S. J., Jr. and Brandt, H. E. (eds.): *Quantum Computation and Quantum Information: A Millennium Volume*. AMS Contemporary Mathematics Series, **305**:53–74, 2002.
9. H. Buhrman, R. Cleve, and A. Wigderson. Quantum vs. classical communication and computation. In *Proceedings of 30th ACM STOC*, pages 63–68, 1998.
10. L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM STOC*, pages 212–219, 1996.
11. P. Høyer and R. de Wolf. Improved quantum communication complexity bounds for disjointness and equality. In *Proceedings of 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS’2002)*, *Lecture Notes in Computer Science*, Vol. 2285, pages 299–310. Springer-Verlag, 2002.
12. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
13. Y. Shi. Approximating linear restrictions of Boolean functions. Unpublished manuscript, 2002.

A Direct Sum Theorem in Communication Complexity via Message Compression

Rahul Jain¹, Jaikumar Radhakrishnan^{1*}, and Pranab Sen^{2**}

¹ School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai 400005, India.

{rahulj, jaikumar}@tcs.tifr.res.in.

² Department of Combinatorics and Optimisation, University of Waterloo, Waterloo, Ontario, N2L 3C1, Canada.

p2sen@cacr.math.uwaterloo.ca.

Abstract. We prove lower bounds for the *direct sum* problem for two-party bounded error randomised multiple-round communication protocols. Our proofs use the notion of *information cost* of a protocol, as defined by Chakrabarti et al. [CSWY01] and refined further by Bar-Yossef et al. [BJKS02]. Our main technical result is a ‘compression’ theorem saying that, for any probability distribution μ over the inputs, a k -round private coin bounded error protocol for a function f with information cost c can be converted into a k -round deterministic protocol for f with bounded distributional error and communication cost $O(kc)$. We prove this result using a *Substate* Theorem about *relative entropy* and a *rejection sampling* argument. Our direct sum result follows from this ‘compression’ result via elementary information theoretic arguments. We also consider the direct sum problem in quantum communication. Using a probabilistic argument, we show that messages cannot be compressed in this manner even if they carry small information.

1 Introduction

We consider the two-party *communication complexity* of computing a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$. There are two players Alice and Bob. Alice is given an input $x \in \mathcal{X}$ and Bob is given an input $y \in \mathcal{Y}$. They then exchange messages in order to determine $f(x, y)$. The goal is to devise a protocol that minimises the amount of communication. In the *randomised* communication complexity model, Alice and Bob are allowed to toss coins and base their actions on the outcome of these coin tosses, and are required to determine the correct value with high probability for every input. There are two models for randomised protocols: in the *private coin* model the coin tosses are private to each player; in the *public coin* model the two players share a string that is generated randomly (independently of the input). A protocol where k messages are exchanged between the two players is called a k -round protocol. One also considers protocols where the two parties send a

* Part of this work was done while visiting MSRI, Berkeley.

** This work was done while visiting TIFR, Mumbai and MSRI, Berkeley.

message each to a referee who determines the answer: this is the *simultaneous* message model.

The starting point of our work is a recent result of Chakrabarti, Shi, Wirth and Yao [CSWY01] concerning the *direct sum* problem in communication complexity. For a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$, the m -fold *direct sum* is the function $f^m : \mathcal{X}^m \times \mathcal{Y}^m \rightarrow \mathcal{Z}^m$, defined by $f^m(\langle x_1, \dots, x_m \rangle, \langle y_1, \dots, y_m \rangle) \triangleq \langle f(x_1, y_1), \dots, f(x_m, y_m) \rangle$. One then studies the communication complexity of f^m as the parameter m increases. Chakrabarti et al. [CSWY01] considered the direct sum problem in the bounded error simultaneous message private coin model and showed that for the equality function $\text{EQ}_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$, the communication complexity of EQ_n^m is $\Omega(m)$ times the communication complexity of EQ_n . In fact, their result is more general. Let $R^{\text{sim}}(f)$ be the bounded error simultaneous message private coin communication complexity of $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$, and let $\tilde{R}^{\text{sim}}(f) \triangleq \min_S R^{\text{sim}}(f|_{S \times S})$, where S ranges over all subsets of $\{0, 1\}^n$ of size at least $(\frac{2}{3})2^n$.

Theorem ([CSWY01]) $R^{\text{sim}}(f^m) = \Omega(m(\tilde{R}^{\text{sim}}(f) - O(\log n)))$. A similar result holds for two-party bounded error one-round protocols too.

The proof of this result in [CSWY01] had two parts. The first part used the notion of information cost of randomised protocols, which is the mutual information between the inputs (which were chosen with uniform distribution in [CSWY01]) and the transcript of the communication between the two parties. Clearly, the information cost is bounded by the length of the transcript. So, showing lower bounds on the information cost gives a lower bound on the communication complexity. Chakrabarti et al. showed that the information cost is additive, that is, the information cost of f^m is m times the information cost of f . The second part of their argument showed an interesting message compression result for communication protocols. This result can be stated informally as follows: if the message contains at most a bits of information about a player's input, then one can modify the (one-round or simultaneous message) protocol so that the length of the message is $O(a + \log n)$. Thus, one obtains a lower bound on the information cost of f if one has a suitable lower bound on the communication complexity f . By combining this with the first part, we see that the communication complexity of f^m is at least m times this lower bound on the communication complexity of f .

In this paper, we examine if this approach can be employed for protocols with more than one-round of communication. Let $R_\delta^k(f)$ denote the k -round private coin communication complexity of f where the protocol is allowed to err with probability at most δ on any input. Let μ be a probability distribution on the inputs of f . Let $C_{\mu, \delta}^k(f)$ denote the deterministic k -round communication complexity of f , where the protocol errs for at most δ fraction, according to the distribution μ , of the inputs. Let $C_{[\cdot], \delta}^k(f)$ denote the maximum, over all product distributions μ , of $C_{\mu, \delta}^k(f)$. We prove the following.

Theorem: Let m, k be positive integers, and $\epsilon, \delta > 0$. Let $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$ be a function. Then, $R_\delta^k(f^m) \geq m \cdot (\frac{\epsilon^2}{2k} \cdot C_{[], \delta+2\epsilon}^k(f) - 2)$.

The proof of this result, like the proof in [CSWY01], has two parts, where the first part uses a notion of information cost for k -round protocols, and the second shows how messages can be compressed in protocols with low information cost. We now informally describe the ideas behind these results. To keep our presentation simple, we will now consider the uniform distribution. So, from now on inputs to Alice and Bob are to be chosen randomly from their input sets.

The first part of our argument uses the extension of the notion of information cost to k -round protocols. The information cost of a k -round randomised protocol is the mutual information between the inputs and the transcript. This natural extension, and its refinement to *conditional information cost* by [BJKS02] has proved fruitful in several other contexts [BJKS02, JRS03]. It is easy to see that it is bounded above by the length of the transcript, and a lower bound on the information cost of protocols gives a lower bound on the randomised communication complexity. The first part of the argument in [CSWY01] is still applicable: the information cost is additive; in particular, the k -round information cost of f^m is m times the k -round information cost of f .

The main contribution of this work is in the second part of the argument. This part of Chakrabarti et al. [CSWY01] used a technical argument to compress messages by exploiting the fact that they carry low information. Our proof is based on the connection between mutual information of random variables and the relative entropy of probability distributions (see Section 2 for definition). Intuitively, it is reasonable to expect that if the message sent by Alice contains little information about her input X , then for various values x of X , the conditional distribution on the message, denoted by P_x , are similar. In fact, if we use relative entropy to compare distributions, then one can show that the mutual information is the average taken over x of the relative entropy $S(P_x \| Q)$ of P_x and Q , where $Q = E_X[P_X]$. Thus, if the information between Alice's input and her message is bounded by a , then typically $S(P_x \| Q)$ is about a . To exploit this fact, we use the Substate Theorem of [JRS02] which states (roughly) that if $S(P_x \| Q) \leq a$, then $P_x \leq 2^{-a}Q$. Using a standard *rejection sampling* idea we then show that Alice can restrict herself to a set of just $2^{O(a)}n$ messages; consequently, her messages can be encoded in $O(a + \log n)$ bits. In fact, such a compact set of messages can be obtained by sampling $2^{O(a)}n$ times from distribution Q . In fact, this method gives a more direct proof of the second part of the argument in [CSWY01].

The second part of our argument raises an interesting question in the setting of quantum communication. Can we always make the length of quantum messages comparable to the amount of information they carry about the inputs without significantly changing the error probability of the protocol? That is, for $x \in \{0, 1\}^n$, instead of distributions P_x we have density matrices ρ_x so that the expected quantum relative entropy $E_X[S(\rho_x \| \rho)] \leq a$, where $\rho \triangleq E_X[\rho_x]$. Also,

we are given measurements (POVM elements) M_y^x , $x, y \in \{0, 1\}^n$. Then, we wish to replace ρ_x by ρ'_x so that there is a subspace of dimension $n \cdot 2^{O(a/\epsilon)}$ that contains the support of each ρ'_x ; also, there is a set $A \subseteq \{0, 1\}^n$, $|A| \geq \frac{2}{3} \cdot 2^n$ such that for each $(x, y) \in A \times \{0, 1\}^n$, $|\text{Tr } M_y^x \rho_x - \text{Tr } M_y^x \rho'_x| \leq \epsilon$. Fortunately, the quantum analogue of the Substate Theorem has already been proved by Jain, Radhakrishnan and Sen [JRS02]. Unfortunately, it is the rejection sampling argument that does not generalise to the quantum setting. Indeed, we can prove the following strong negative result about compressibility of quantum information: For sufficiently large constant a , there exist ρ_x, M_y^x , $x, y \in \{0, 1\}^n$ as above such that any subspace containing the supports of ρ'_x as above has dimension at least $2^{n/6}$. This strong negative result seems to suggest that new techniques may be required to tackle the direct sum problem for quantum communication.

1.1 Previous Results

The direct sum problem for communication complexity has been extensively studied in the past (see Kushilevitz and Nisan [KN97]). Let $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$ be a function. Let $C(f)$ ($R(f)$) denote the deterministic (bounded error private coin randomised) two-party communication complexity of f . Ceder, Kushilevitz, Naor and Nisan [FKNN95] showed that there exists a partial function f with $C(f) = \Theta(\log n)$, whereas solving m copies takes only $C(f^m) = O(m + \log m \cdot \log n)$. They also showed a lower bound $C(f^m) \geq m(\sqrt{C(f)/2} - \log n - O(1))$ for total functions f . For the one-round deterministic model, they showed that $C(f^m) \geq m(C(f) - \log n - O(1))$ even for partial functions. For the two-round deterministic model, Karchmer, Kushilevitz and Nisan [KKN92] showed that $C(f^m) \geq m(C(f) - O(\log n))$ for any relation f . Feder et al. [FKNN95] also showed that for the equality problem $R(EQ_n^m) = O(m + \log n)$.

1.2 Our Results

Result 1 (Compression result, multiple-rounds) *Suppose that Π is a k -round private coin randomised protocol for $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$. Let the average error of Π under a probability distribution μ on the inputs $\mathcal{X} \times \mathcal{Y}$ be δ . Let X, Y denote the random variables corresponding to Alice's and Bob's inputs respectively. Let T denote the complete transcript of messages sent by Alice and Bob. Suppose $I(XY : T) \leq a$. Let $\epsilon > 0$. Then, there is another deterministic protocol Π' with the following properties:*

- (a) *The communication cost of Π' is at most $\frac{2k(a+1)}{\epsilon^2} + \frac{2k}{\epsilon}$ bits;*
- (b) *The distributional error of Π' under μ is at most $\delta + 2\epsilon$.*

Result 2 (Direct sum, multiple-rounds) *Let m, k be positive integers, and $\epsilon, \delta > 0$. Let $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$ be a function. Then, $R_\delta^k(f^m) \geq m \cdot \left(\frac{\epsilon^2}{2k} \cdot C_{[\cdot], \delta+2\epsilon}^k(f) - 2 \right)$.*

Result 3 (Quantum incompressibility) *Let m, n, d be positive integers and $k \geq 7$. Let $d \geq 160^2$, $1600 \cdot d^4 \cdot k 2^k \ln(20d^2) < m$ and $3200 \cdot d^5 \cdot 2^{2k} \ln d < n$. Let the underlying Hilbert space be $\mathbb{C}^m$. There exist n states ρ_l and n orthogonal projections M_l , $1 \leq l \leq n$, such that*

- (a) $\forall l \operatorname{Tr} M_l \rho_l = 1$.
- (b) $\rho \triangleq \frac{1}{n} \cdot \sum_l \rho_l = \frac{1}{m} \cdot I$, where I is the identity operator on $\mathbb{C}^m$.
- (c) $\forall l S(\rho_l \| \rho) = k$.
- (d) *For all d -dimensional subspaces W of $\mathbb{C}^m$, for all ordered sets of density matrices $\{\sigma_l\}_{l \in [n]}$ with support in W , $|\{l : \operatorname{Tr} M_l \sigma_l \leq 1/10\}| \geq n/4$.*

1.3 Organisation of the Rest of the Paper

In Section 2, we present the necessary background from information theory and communication complexity. In Section 3 we prove a version of the compression result for bounded error private coin simultaneous message protocols and state the direct sum result for such protocols. Our version is slightly stronger than the one in [CSWY01]. The main ideas of this work (i.e. the use of the Substate Theorem and rejection sampling) are already encountered in this section. In Section 4 we prove the compression result for k -round bounded error private coin protocols, and state the direct sum result for such protocols. We do not present a proof of the quantum incompressibility result due to lack of space. It can be found in the full version of the paper at <http://arxiv.org/ps/cs.CC/0304020>.

2 Preliminaries

2.1 Information Theoretic Background

In this paper, $\ln$ denotes the natural logarithm and $\log$ denotes logarithm to base 2. All random variables will have finite range. Let $[k] \triangleq \{1, \dots, k\}$. Let $P, Q : [k] \rightarrow \mathbb{R}$. The *total variation distance* (aka ℓ_1 -distance) between P, Q is defined as $\|P - Q\|_1 \triangleq \sum_{i \in [k]} |P(i) - Q(i)|$. We say $P \leq Q$ iff $P(i) \leq Q(i)$ for all $i \in [k]$. Suppose X, Y, Z are random variables with some joint distribution. The *Shannon entropy* of X is defined as $H(X) \triangleq -\sum_x \Pr[X = x] \log \Pr[X = x]$. The *mutual information* of X and Y is defined as $I(X : Y) \triangleq H(X) + H(Y) - H(XY)$. For $z \in \operatorname{range}(Z)$, $I((X : Y) \mid Z = z)$ denotes the mutual information of X and Y conditioned on the event $Z = z$ i.e. the mutual information arising from the joint distribution of X, Y conditioned on $Z = z$. Define $I((X : Y) \mid Z) \triangleq \mathbb{E}_Z I((X : Y) \mid Z = z)$. It is readily seen that $I((X : Y) \mid Z) = H(XZ) + H(YZ) - H(XYZ) - H(Z)$.

We now recall the definition of an important information theoretic quantity called *relative entropy*.

Definition 1 (Relative entropy). *Let P and Q be probability distributions on a set $[k]$. The relative entropy of P and Q is given by $S(P \| Q) \triangleq \sum_{i \in [k]} P(i) \log \frac{P(i)}{Q(i)}$.*

The following facts follow easily from the definitions.

Fact 1 *Let X, Y, Z, W be random variables with some joint distribution. Then, (a) $I(X : YZ) = I(X : Y) + I((X : Z) \mid Y)$, and (b) $I(XY : Z \mid W) \geq I(XY : Z) - H(W)$.*

Fact 2 *Let (X, M) be a pair of random variables with some joint distribution. Let P be the (marginal) probability distribution of M , and for each $x \in \text{range}(X)$, let P_x be the conditional distribution of M given $X = x$. Then $I(X : M) = \mathbb{E}_X[S(P_x \| P)]$, where the expectation is taken according to the marginal distribution of X .*

Our main information theoretic tool in this paper is the following fact proved in [JRS02].

Fact 3 (Substate theorem) *Suppose P and Q are probability distributions on $[k]$ such that $S(P \| Q) = a$. Let $r \geq 1$. Then,*

- (a) *the set $\text{Good} \triangleq \{i \in [k] : \frac{P(i)}{2^{r(a+1)}} \leq Q(i)\}$ has probability at least $1 - \frac{1}{r}$ in P ;*
- (b) *There is a distribution $\tilde{P}$ on $[k]$ such that $\|P - \tilde{P}\|_1 \leq \frac{2}{r}$ and $\alpha \tilde{P} \leq Q$, where $\alpha \triangleq \left(\frac{r-1}{r}\right) 2^{-r(a+1)}$.*

2.2 Communication Complexity Background

In the two-party private coin randomised communication complexity model [Yao79], two players Alice and Bob are required to collaborate to compute a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$. Alice is given $x \in \mathcal{X}$ and Bob is given $y \in \mathcal{Y}$. Let $\Pi(x, y)$ be the random variable denoting the entire transcript of the messages exchanged by Alice and Bob by following the protocol Π on input x and y . We say Π is a δ -error protocol if for all x and y , the answer determined by the players is correct with probability (taken over the coin tosses of Alice and Bob) at least $1 - \delta$. The communication cost of Π is the maximum length of $\Pi(x, y)$ over all x and y , and over all random choices of Alice and Bob. The k -round δ -error private coin randomised communication complexity of f , denoted $R_\delta^k(f)$, is the communication cost of the best private coin k -round δ -error protocol for f . When δ is omitted, we mean that $\delta = \frac{1}{3}$.

We also consider private coin randomised simultaneous protocols in this paper. $R_\delta^{\text{sim}}(f)$ denotes the δ -error private coin randomised simultaneous communication complexity of f . When δ is omitted, we mean that $\delta = \frac{1}{3}$.

Let μ be a probability distribution on $\mathcal{X} \times \mathcal{Y}$. A deterministic protocol Π has distributional error δ if the probability of correctness of Π , averaged with respect to μ , is least $1 - \delta$. The k -round δ -error distributional communication complexity of f , denoted $C_{\mu, \delta}^k(f)$, is the communication cost of the best k -round deterministic protocol for f with distributional error δ . μ is said to be a product distribution if there exist probability distributions $\mu_{\mathcal{X}}$ on $\mathcal{X}$ and

$\mu_{\mathcal{Y}}$ on $\mathcal{Y}$ such that $\mu(x, y) = \mu_{\mathcal{X}}(x) \cdot \mu_{\mathcal{Y}}(y)$ for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$. The k -round δ -error product distributional communication complexity of f is defined as $C_{[\cdot], \delta}^k(f) = \sup_{\mu} C_{\mu, \delta}^k(f)$, where the supremum is taken over all product distributions μ on $\mathcal{X} \times \mathcal{Y}$. When δ is omitted, we mean that $\delta = \frac{1}{3}$.

We now recall the definition of the important notion of *information cost* of a communication protocol from Bar-Yossef et al. [BJKS02].

Definition 2 (Information cost). Let Π be a private coin randomised protocol for a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$. Let $\Pi(x, y)$ be the entire message transcript of the protocol on input (x, y) . Let μ be a distribution on $\mathcal{X} \times \mathcal{Y}$, and let the input random variable (X, Y) have distribution μ . The information cost of Π under μ is defined to be $I(XY : \Pi(X, Y))$. The k -round δ -error information complexity of f under the distribution μ , denoted by $IC_{\mu, \delta}^k(f)$, is the infimum information cost under μ of a k -round δ -error protocol for f . $IC_{\delta}^{\text{sim}}(f)$ denotes the infimum information cost under the uniform probability distribution on the inputs of a private coin simultaneous δ -error protocol for f .

Remark: In Chakrabarti et al. [CSWY01], the information cost of a private coin δ -error simultaneous message protocol Π is defined as follows: Let X (Y) denote the random variable corresponding to Alice's (Bob's) input, and let M (N) denote the random variable corresponding to Alice's (Bob's) message to the referee. The information cost of Π is defined as $I(X:M) + I(Y:N)$. We note that our definition of information cost coincides with Chakrabarti et al.'s definition for simultaneous message protocols.

Let μ be a probability distribution on $\mathcal{X} \times \mathcal{Y}$. The probability distribution μ^m on $\mathcal{X}^m \times \mathcal{Y}^m$ is defined as $\mu^m(\langle x_1, \dots, x_m \rangle, \langle y_1, \dots, y_m \rangle) \triangleq \mu(x_1, y_1) \cdot \mu(x_2, y_2) \cdots \mu(x_m, y_m)$. Suppose μ is a product probability distribution on $\mathcal{X} \times \mathcal{Y}$. It can be easily seen (see e.g. [BJKS02]) that for any positive integers m, k , and real $\delta > 0$, $IC_{\mu^m, \delta}^k(f^m) \geq m \cdot IC_{\mu, \delta}^k(f)$. The reason for requiring μ to be a product distribution is as follows. We define the notion of information cost for private coin protocols only. This is because the proof of our message compression theorem (Theorem 3), which makes use of information cost, works for private coin protocols only. If μ is not a product distribution, the protocol for f which arises out of the protocol for f^m in the proof of the above inequality fails to be a private coin protocol, even if the protocol for f^m was private coin to start with. To get over this restriction on μ , Bar-Yossef et al. [BJKS02] introduced the notion of *conditional information cost* of a protocol. Suppose the distribution μ is expressed as a convex combination $\mu = \sum_{d \in K} \kappa_d \mu_d$ of product distributions μ_d , where K is some finite index set. Let κ denote the probability distribution on K defined by the numbers κ_d . Define the random variable D to be distributed according to κ . Conditioned on D , μ is a product distribution on $\mathcal{X} \times \mathcal{Y}$. We will call μ a mixture of product distributions $\{\mu_d\}_{d \in K}$ and say that κ partitions μ . The probability distribution κ^m on K^m is defined as $\kappa^m(d_1, \dots, d_m) \triangleq \kappa(d_1) \cdot \kappa(d_2) \cdots \kappa(d_m)$. Then κ^m partitions μ^m in a natural way. The random variable D^m has distribution κ^m . Conditioned on D^m , μ^m is a product distribution on $\mathcal{X}^m \times \mathcal{Y}^m$.

Definition 3 (Conditional information cost). Let Π be a private coin randomised protocol for a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$. Let $\Pi(x, y)$ be the entire message transcript of the protocol on input (x, y) . Let μ be a distribution on $\mathcal{X} \times \mathcal{Y}$, and let the input random variable (X, Y) have distribution μ . Let μ be a mixture of product distributions partitioned by κ . Let the random variable D be distributed according to κ . The conditional information cost of Π under (μ, κ) is defined to be $I((XY : \Pi(X, Y)) \mid D)$. The k -round δ -error conditional information complexity of f under (μ, κ) , denoted by $IC_{\mu, \delta}^k(f \mid \kappa)$, is the infimum conditional information cost under (μ, κ) of a k -round δ -error protocol for f .

The following facts follow easily from the results in Bar-Yossef et al. [BJKS02] and Fact 1.

Fact 4 Let μ be a probability distribution on $\mathcal{X} \times \mathcal{Y}$. Let κ partition μ . For any $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$, positive integers m, k , real $\delta > 0$, $IC_{\mu, \delta}^k(f^m \mid \kappa^m) \geq m \cdot IC_{\mu, \delta}^k(f \mid \kappa) \geq m \cdot (IC_{\mu, \delta}^k(f) - H(\kappa))$.

Fact 5 With the notation and assumptions of Fact 4, $R_\delta^k(f) \geq IC_{\mu, \delta}^k(f \mid \kappa)$.

3 Simultaneous Message Protocols

In this section, we prove a result of [CSWY01], which states that if the mutual information between the message and the input is at most k , then the protocol can be modified so that the players send messages of length at most $O(k + \log n)$ bits. Our proof will make use of the Substate Theorem and a rejection sampling argument. In the next section, we will show how to extend this argument to multiple-round protocols.

Before we formally state the result and its proof, let us outline the main idea. Fix a simultaneous message protocol for computing the function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \mathcal{Z}$. Let $X \in_U \{0, 1\}^n$. Suppose $I(X : M) \leq a$, where M be the message sent by Alice to the referee when her input is X . Let $s_{xy}(m)$ be conditional probability that the referee computes $f(x, y)$ correctly when Alice's message is m , her input is x and Bob's input is y .

We want to show that we can choose a small subset $\mathcal{M}$ of possible messages, so that for most x , Alice can generate a message M'_x from this subset (according to some distribution that depends on x), and still ensure that $E[s_{xy}(M'_x)]$ is close to 1, for all y . Let P_x be the distribution of M conditioned on the event $X = x$. For a fixed x , it is possible to argue that we can confine Alice's messages to a certain small subset $\mathcal{M}_x \subseteq [k]$. Let $\mathcal{M}_x$ consist of $O(n)$ messages picked according to the distribution P_x . Then, instead of sending messages according to the distribution P_x , Alice can send a random message chosen from $\mathcal{M}_x$. Using Chernoff-Hoeffding bounds one can easily verify that $\mathcal{M}_x$ will serve our purposes with exponentially high probability.

However, what we really require is a set of samples $\{\mathcal{M}_x\}$ whose union is small, so that she and the referee can settle on a common succinct encoding for the messages. Why should such samples exist? Since $I(X : M)$ is small, we

have by Fact 2 that for most x , the relative entropy $S(P_x \| Q)$ is bounded (here Q is the distribution of the message M , i.e., $Q = \mathbb{E}_X[P_X]$). By combining this fact, the Substate Theorem (Fact 3) and a *rejection sampling* argument (see e.g. [Ros97, Chapter 4, Section 4.4]), one can show that if we choose a sample of messages according to the distribution Q , then, for most x , roughly one in every $2^{O(a)}$ messages ‘can serve’ as a message sampled according to the distribution P_x . Thus, if we pick a sample of size $n \cdot 2^{O(a)}$ according to Q , then for most x we can get a the required sub-sample $\mathcal{M}_x$ of $O(n)$ elements. The formal arguments are presented below.

The following easy lemma is the basis of the rejection sampling argument.

Lemma 1 (Rejection sampling). *Let P and Q be probability distributions on $[k]$ such that $2^{-a}P \leq Q$. Then, there exist random variables X and χ taking values in $[k] \times \{0, 1\}$, such that: (a) X has distribution Q , (b) $\Pr[\chi = 1] = 2^{-a}$ and (c) $\Pr[X = i \mid \chi = 1] = P(i)$.*

Proof. Since the distribution of X is required to be Q , we will just describe the conditional distribution of χ for each potential value i for X : let $\Pr[\chi = 1 \mid X = i] = P(i)/(2^a Q(i))$. Then,

$$\begin{aligned} \Pr[\chi = 1] &= \sum_{i \in [k]} P[X = i] \cdot \Pr[\chi = 1 \mid X = i] = 2^{-a} \\ \Pr[X = i \mid \chi = 1] &= \frac{\Pr[X = i \wedge \chi = 1]}{\Pr[\chi = 1]} = \frac{Q(i) \cdot P(i)/(2^a Q(i))}{2^{-a}} = P(i). \end{aligned}$$

In order to combine this argument with the Substate Theorem to generate simultaneously a sample $\mathcal{M}$ of messages according to the distribution Q and several subsamples $\mathcal{M}_x$, we will need a slight extension of the above lemma. Below, the notation $B(t, q)$ stands for the binomial distribution got by t independent coin tosses of a binary coin with success probability q for each toss.

Lemma 2. *Let P and Q be probability distributions on $[k]$ such that $2^{-a}P \leq Q$. Then, for each integer $t \geq 1$, there exist correlated random variables $\mathbf{X} = \langle X_1, X_2, \dots, X_t \rangle$ and $\mathbf{Y} = \langle Y_1, Y_2, \dots, Y_R \rangle$ such that*

- (a) *The random variables $(X_i : i \in [t])$ are independent and each X_i has distribution Q ;*
- (b) *R is a random variable with binomial distribution $B(t, 2^{-a})$;*
- (c) *Conditioned on the event $R = r$, the random variables $(Y_i : i \in [r])$ are independent and each Y_i has distribution P .*
- (d) *$\mathbf{Y}$ is a subsequence of $\mathbf{X}$ (with probability 1).*

Proof. We generate t independent copies of the random variables (X, χ) promised by Lemma 1; this gives us $\mathbf{X} = \langle X_1, X_2, \dots, X_t \rangle$ and $\chi = \langle \chi_1, \chi_2, \dots, \chi_t \rangle$. Let $\mathbf{Y} \triangleq \langle X_i : \chi_i = 1 \rangle$. It is easy to verify that $\mathbf{X}$ and $\mathbf{Y}$ satisfy conditions (a)–(d). $\square$

Our next lemma uses Lemma 2 to pick a sample of messages according to the average distributions Q and find subsamples inside it for several distributions P_x . This lemma will be crucial to show the compression result for simultaneous message protocols (Theorem 1).

Lemma 3. Let Q and $P_1, P_2, \dots, P_N$ be probability distributions on $[k]$. Define $a_i \triangleq S(P_i \| Q)$. Suppose $a_i < \infty$ for all $i \in [N]$. Let $s_{ij}, s_{ij}, \dots, s_{ij}$ be functions from $[k]$ to $[0, 1]$. (In our application, they will correspond to conditional probability that the referee gives the correct answer when Alice sends a certain message from $[k]$). Let $p_{ij} \triangleq E_{y \in P_i[k]}[s_{ij}(y)]$. Fix $\epsilon \in (0, 1]$. Then, there exists a sequence $\mathbf{x} \triangleq \langle x_1, \dots, x_t \rangle$ of elements of $[k]$ and subsequences $\mathbf{y}^1, \dots, \mathbf{y}^N$ of $\mathbf{x}$ such that

- (a) $\mathbf{y}^i$ is a subsequence of $\langle x_1, \dots, x_{t_i} \rangle$ where, $t_i \triangleq \left\lceil \frac{8 \cdot 2^{(a_i+1)/\epsilon} \cdot \log(2N)}{(1-\epsilon)\epsilon^2} \right\rceil$.
- (b) For $i, j = 1, 2, \dots, N$, $\left| E_{\ell \in U[r_i]}[s_{ij}(\mathbf{y}^i[\ell])] - p_{ij} \right| \leq 2\epsilon$, where r_i is the length of $\mathbf{y}^i$.
- (c) $t \triangleq \max_i t_i$.

Proof. Using part (b) of Fact 3, we obtain distributions $\tilde{P}_i$ such that $\|P_i - \tilde{P}_i\|_1 \leq 2\epsilon$ and $(1-\epsilon)2^{-(a_i+1)/\epsilon}\tilde{P}_i \leq Q$. Using Lemma 2, we can construct correlated random variables $(\mathbf{X}, \mathbf{Y}^1, \mathbf{Y}^2, \dots, \mathbf{Y}^N)$ such that $\mathbf{X}$ is a sequence of $t \triangleq \max_i t_i$ independent random variables, each distributed according to Q , and $(\mathbf{X}[1, t_i], \mathbf{Y}^i)$ satisfying conditions (a)–(d) (with $P = P_i$, $a = (a_i+1)/\epsilon - \log(1-\epsilon)$ and $t = t_i$). We will show that with non-zero probability these random variables satisfy conditions (a) and (b) of the present lemma. This implies that there is a choice $(\mathbf{x}, \mathbf{y}^1, \dots, \mathbf{y}^N)$ for $(\mathbf{X}, \mathbf{Y}^1, \dots, \mathbf{Y}^N)$ satisfying parts (a) and (b) of the present lemma.

Let R_i denote the length of $\mathbf{Y}^i$. Using standard Chernoff bounds (see e.g. [AS00, Theorem A.13]), $\Pr[\exists i, R_i < (4/\epsilon^2)\log(2N)] < N \cdot \frac{1}{2N} = \frac{1}{2}$. Now, condition on the event $R_i \geq (\frac{4}{\epsilon^2})\log(2N)$, for all $1 \leq i \leq N$. Define $\tilde{p}_{ij} \triangleq \Pr_{y \in \tilde{P}_i[k]}[s_{ij}(y)]$. Using standard Chernoff-Hoeffding bounds (see e.g. [AS00, Corollary A.7]), we conclude that for $i, j = 1, 2, \dots, N$, $\Pr_{\mathbf{Y}^i} \left[\left| E_{\ell \in U[r_i]}[s_{ij}(\mathbf{Y}^i[\ell])] - \tilde{p}_{ij} \right| > \epsilon \right] < \frac{2}{(2N)^8}$, implying, $\Pr_{\mathbf{Y}^1, \dots, \mathbf{Y}^N} \left[\exists i, j, \left| E_{\ell \in U[r_i]}[s_{ij}(\mathbf{Y}^i[\ell])] - \tilde{p}_{ij} \right| > \epsilon \right] \leq N^2 \times \frac{2}{(2N)^8} < \frac{1}{2}$. From the fact that $\forall i, j, |p_{ij} - \tilde{p}_{ij}| \leq \epsilon$ (since $\|P_i - \tilde{P}_i\|_1 \leq 2\epsilon$), it follows that part (b) of our lemma holds with non-zero probability. Part (a) is never violated. Part (c) is true by definition of t . $\square$

Theorem 1 (Compression result, simultaneous messages). Suppose that Π is a δ -error private coin simultaneous message protocol for $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \mathcal{Z}$. Let the inputs to f be chosen according to the uniform distribution. Let X, Y denote the random variables corresponding to Alice's and Bob's inputs respectively, and M_A, M_B denote the random variables corresponding to Alice's and Bob's messages respectively. Suppose $I(X : M_A) \leq a$ and $I(Y : M_B) \leq b$. Then, there exist sets $\text{Good}_A, \text{Good}_B \subseteq \{0, 1\}^n$ such that $|\text{Good}_A| \geq \frac{2}{3} \cdot 2^n$ and

$|\text{Good}_B| \geq \frac{2}{3} \cdot 2^n$, and a private coin simultaneous message protocol Π' with the following properties:

- (a) In Π' , Alice sends messages of length at most $\frac{3a+1}{\epsilon} + \log(n+1) + \log \frac{1}{\epsilon^2(1-\epsilon)} + 4$ bits and Bob sends messages of length at most $\frac{3b+1}{\epsilon} + \log(n+1) + \log \frac{1}{\epsilon^2(1-\epsilon)} + 4$ bits.
- (b) For each input $(x, y) \in \text{Good}_A \times \text{Good}_B$, the error probability of Π' is at most $\delta + 4\epsilon$.

Proof. Let P be the distribution of M_A , and let P_x be its distribution under the condition $X = x$. Note that by Fact 2, we have $\mathbb{E}_X[S(P_x \| P)] \leq a$, where the expectation is got by choosing x uniformly from $\{0, 1\}^n$. Therefore there exists a set Good_A , $|\text{Good}_A| \geq \frac{2}{3} \cdot 2^n$, such that for all $x \in \text{Good}_A$, $S(P_x \| P) \leq 3a$.

Define $t_a \triangleq \frac{8(n+1)2^{(3a+1)/\epsilon}}{\epsilon^2(1-\epsilon)}$. From Lemma 3, we know that there is a sequence of messages $\sigma = \langle m_1, \dots, m_{t_a} \rangle$ and subsequences σ_x of σ such that on input $x \in \text{Good}_A$, if Alice sends a uniformly chosen random message of σ_x instead of sending messages according to distribution P_x , the probability of error for any $y \in \{0, 1\}^n$ changes by at most 2ϵ . We now define an intermediate protocol Π'' as follows. The messages in σ are encoded using at most $\log t_a + 1$ bits. In protocol Π'' for $x \in \text{Good}_A$, Alice sends a uniformly chosen random message from σ_x ; for $x \notin \text{Good}_A$, Alice sends a fixed arbitrary message from σ . Bob's strategy in Π'' is the same as in Π . In Π'' , the error probability of an input $(x, y) \in \text{Good}_A \times \{0, 1\}^n$ is at most $\delta + 2\epsilon$, and $I(Y : M_B) \leq b$. Now arguing similarly, the protocol Π'' can be converted to a protocol Π' by compressing Bob's message to at most $\log t_b + 1$ bits, where $t_b \triangleq \frac{8(n+1)2^{(3b+1)/\epsilon}}{\epsilon^2(1-\epsilon)}$. In Π' , the error for an input $(x, y) \in \text{Good}_A \times \text{Good}_B$ is at most $\delta + 4\epsilon$. $\square$

Corollary 1. Let $\delta, \epsilon > 0$. Let $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \mathcal{Z}$ be a function. Let the inputs to f be chosen according to the uniform distribution. Then there exist sets $\text{Good}_A, \text{Good}_B \subseteq \{0, 1\}^n$ such that $|\text{Good}_A| \geq \frac{2}{3} \cdot 2^n$, $|\text{Good}_B| \geq \frac{2}{3} \cdot 2^n$, and $IC_\delta^{\text{sim}}(f) \geq \frac{\epsilon}{3}(R_{\delta+4\epsilon}^{\text{sim}}(f') - 2\log(n+1) - 2\log \frac{1}{\epsilon^2(1-\epsilon)} - \frac{2}{\epsilon} - 8)$, where f' is the restriction of f to $\text{Good}_A \times \text{Good}_B$.

We can now prove the key theorem of Chakrabarti et al. [CSWY01].

Theorem 2 (Direct sum, simultaneous messages). Let $\delta, \epsilon > 0$. Let $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \mathcal{Z}$ be a function. Define $\tilde{R}_\delta^{\text{sim}}(f) \triangleq \min_{f'} R_\delta^{\text{sim}}(f')$, where the minimum is taken over all functions f' which are the restrictions of f to sets of the form $A \times B$, $A, B \subseteq \{0, 1\}^n$, $|A| \geq \frac{2}{3} \cdot 2^n$, $|B| \geq \frac{2}{3} \cdot 2^n$. Then, $R_\delta^{\text{sim}}(f^m) \geq \frac{m\epsilon}{3}(\tilde{R}_{\delta+4\epsilon}^{\text{sim}}(f) - 2\log(n+1) - 2\log \frac{1}{\epsilon^2(1-\epsilon)} - \frac{2}{\epsilon} - 8)$.

Proof. Immediate from Fact 5, Fact 4 and Corollary 1. $\square$

Remarks:

1. The above theorem implies lower bounds for the simultaneous direct sum complexity of equality, as well as lower bounds for some related problems as in

Chakrabarti et al. [CSWY01]. The dependence of the bounds on ϵ is better in our version.

2. A very similar direct sum theorem can be proved about two-party one-round private coin protocols.
3. All the results in this section, including the above remark, hold even when f is a relation.

4 Multiple-Round Protocols

We first prove Lemma 4, which intuitively shows that if P, Q are probability distributions on $[k]$ such that $P \leq 2^a Q$, then about it is enough to sample Q independently $2^{O(a)}$ times to produce one sample element Y according to P . In the statement of the lemma, the random variable $\mathbf{X}$ represents an infinite sequence of independent sample elements chosen according to Q , the random variable R indicates how many of these elements have to be considered till ‘stopping’. $R = \infty$ indicates that we do not ‘stop’. If we do ‘stop’, then either we succeed in producing a sample according to P (in this case, the sample $Y = X_R$), or we give up (in this case, we set $Y = 0$). In the proof of the lemma, $\star$ indicates that we do not ‘stop’ at the current iteration and hence the rejection sampling process must go further.

Lemma 4. *Let P and Q be probability distributions on $[k]$, such that $\text{Good} \triangleq \{i \in [k] : \frac{P(i)}{2^a} \leq Q(i)\}$ has probability exactly $1 - \epsilon$ in P . Then, there exist correlated random variables $\mathbf{X} \triangleq \langle X_i \rangle_{i \in \mathbb{N}_+}$, R and Y such that*

- (a) *the random variables $(X_i : i \in \mathbb{N}_+)$ are independent and each has distribution Q ;*
- (b) *R takes values in $\mathbb{N}_+ \cup \{\infty\}$ and $\mathbb{E}[R] = 2^a$;*
- (c) *if $R \neq \infty$, then $Y = X_R$ or $Y = 0$;*
- (d) *Y takes values in $\{0\} \cup [k]$, such that: $\Pr[Y = i] = \begin{cases} P(i) & \text{if } i \in \text{Good} \\ 0 & \text{if } i \in [k] - \text{Good} \\ \epsilon & \text{if } i = 0 \end{cases}$.*

Proof. First, we define a pair of correlated random variables (X, Z) , where X takes values in $[k]$ and Z in $[k] \cup \{0, \star\}$. Let $P' : [k] \rightarrow [0, 1]$ be defined by $P'(i) = P(i)$ for $i \in \text{Good}$, and $P'(i) = 0$ for $i \in [k] - \text{Good}$. Let $\beta \triangleq \epsilon 2^{-a} / (1 - (1 - \epsilon) 2^{-a})$ and $\gamma_i \triangleq P'(i) 2^{-a} / Q(i)$. The joint probability distribution of X and Z is given by $\Pr[X = i] = Q(i), \forall i \in [k]$, and $\Pr[Z = j \mid X = i]$ is equal to γ_i if $j = i$, equal to $\beta(1 - \gamma_i)$ if $j = 0$, equal to $1 - \gamma_i - \beta(1 - \gamma_i)$ if $j = \star$, and equal to 0 otherwise. Note that this implies that $\Pr[Z \neq \star] = \sum_{i \in [k]} Q(i) \cdot [\gamma_i + \beta(1 - \gamma_i)] =$

$\beta + (1 - \beta) \sum_{i \in [k]} P'(i) 2^{-a} = \beta + (1 - \beta)(1 - \epsilon) 2^{-a} = 2^{-a}$. Now, consider the sequence

of random variables $\mathbf{X} \triangleq \langle X_i \rangle_{i \in \mathbb{N}_+}$ and $\mathbf{Z} \triangleq \langle Z_i \rangle_{i \in \mathbb{N}_+}$, where each (X_i, Z_i) has the same distribution as (X, Z) defined above and (X_i, Z_i) is independent of all

$(X_j, Z_j), j \neq i$. Let $R \triangleq \min\{i : Z_i \neq \star\}$; $R \triangleq \infty$ if $\{i : Z_i \neq \star\}$ is the empty set. R is a geometric random variable with success probability 2^{-a} , and so satisfies part (b) of the present lemma. Let $Y \triangleq Z_R$ if $R \neq \infty$ and $Y \triangleq 0$ if $R = \infty$. Parts (a) and (c) are satisfied by construction.

We now verify that part (d) is satisfied. Since $\Pr[R = \infty] = 0$, we see that

$$\Pr[Y = i] = \sum_{r \in \mathbb{N}_+} \Pr[R = r] \cdot \Pr[Z_r = i \mid R = r] = \sum_{r \in \mathbb{N}_+} \Pr[R = r] \cdot \frac{\Pr[Z_r = i]}{\Pr[Z_r \neq \star]},$$

where the second equality follows from the independence of (X_r, Z_r) from all $(X_j, Z_j), j \neq r$. If $i \in [k]$, we see that

$$\begin{aligned} \Pr[Y = i] &= \sum_{r \in \mathbb{N}_+} \Pr[R = r] \cdot \frac{\Pr[Z_r = i]}{\Pr[Z_r \neq \star]} = \sum_{r \in \mathbb{N}_+} \Pr[R = r] \cdot \frac{\Pr[X_r = i] \cdot \Pr[Z_r = i \mid X_r = i]}{\Pr[Z_r \neq \star]} \\ &= \sum_{r \in \mathbb{N}_+} \Pr[R = r] \cdot \frac{Q(i)\gamma_i}{2^{-a}} = \sum_{r \in \mathbb{N}_+} \Pr[R = r] P'(i) = P'(i). \end{aligned}$$

Thus, for $i \in \text{Good}$, $\Pr[Y = i] = P(i)$, and for $i \in [k] - \text{Good}$, $\Pr[Y = i] = 0$. Finally,

$$\begin{aligned} \Pr[Y = 0] &= \sum_{r \in \mathbb{N}_+} \Pr[R = r] \cdot \frac{\Pr[Z_r = 0]}{\Pr[Z_r \neq \star]} = \sum_{r \in \mathbb{N}_+} \frac{\Pr[R = r]}{2^{-a}} \sum_{j \in [k]} \Pr[X_r = j] \cdot \Pr[Z_r = 0 \mid X_r = j] \\ &= \sum_{r \in \mathbb{N}_+} \frac{\Pr[R = r]}{2^{-a}} \sum_{j \in [k]} Q(j) \cdot \beta(1 - \gamma_j) = \sum_{r \in \mathbb{N}_+} \Pr[R = r] \epsilon = \epsilon. \end{aligned}$$

□

Lemma 5 follows from Lemma 4, and will be used to prove the message compression result for two-party multiple-round protocols (Theorem 3).

Lemma 5. *Let Q and $P_1, \dots, P_N$ be probability distributions on $[k]$. Define $S(P_i \| Q) = a_i$. Suppose $a_i < \infty$ for all $i \in [N]$. Fix $\epsilon \in (0, 1]$. Then, there exist random variables $\mathbf{X} = \langle X_i \rangle_{i \in \mathbb{N}_+}$, $R_1, \dots, R_N$ and $Y_1, \dots, Y_N$ such that*

- (a) $(X_i : i \in \mathbb{N}_+)$ are independent random variables, each having distribution Q ;
- (b) R_i takes values in $\mathbb{N}_+ \cup \{\infty\}$ and $\mathbb{E}[R_i] = 2^{(a_i+1)/\epsilon}$;
- (c) Y_j takes values in $[k] \cup \{0\}$, and there is a set $\text{Good}_j \subseteq [k]$ with $P_j(\text{Good}_j) \geq 1 - \epsilon$ such that for all $\ell \in \text{Good}_j$, $\Pr[Y_j = \ell] = P_j(\ell)$, for all $\ell \in [k] - \text{Good}_j$, $\Pr[Y_j = \ell] = 0$ and $\Pr[Y_j = 0] = 1 - P_j(\text{Good}_j) \leq \epsilon$;
- (d) if $R_j < \infty$, then $Y_j = X_{R_j}$ or $Y = 0$.

Proof. Using part (a) of Fact 3, we obtain for $j = 1, \dots, N$, a set $\text{Good}_j \subseteq [k]$ such that $P_j(\text{Good}_j) \geq 1 - \epsilon$ and $P_j(i)2^{-(a_j+1)/\epsilon} \leq Q(i)$ for all $i \in \text{Good}_j$. Now from Lemma 4, we can construct correlated random variables $\mathbf{X}$, $Y_1, \dots, Y_N$, and $R_1, \dots, R_N$ satisfying the requirements of the present lemma. □

Theorem 3 (Compression result, multiple-round). *Suppose Π is a k -round private coin randomised protocol for $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$. Let the average error of Π under a probability distribution μ on the inputs $\mathcal{X} \times \mathcal{Y}$ be δ . Let X, Y denote*

the random variables corresponding to Alice's and Bob's inputs respectively. Let T denote the complete transcript of messages sent by Alice and Bob. Suppose $I(XY : T) \leq a$. Let $\epsilon > 0$. Then, there is another deterministic protocol Π' with the following properties:

- (a) The communication cost of Π' is at most $\frac{2k(a+1)}{\epsilon^2} + \frac{2k}{\epsilon}$ bits;
- (b) The distributional error of Π' under μ is at most $\delta + 2\epsilon$.

Proof. The proof proceeds by defining a series of intermediate k -round protocols $\Pi'_k, \Pi'_{k-1}, \dots, \Pi'_1$. Π'_i is obtained from Π'_{i+1} by compressing the message of the i th round. Thus, we first compress the k th message, then the $(k-1)$ th message, and so on. Each message compression step introduces an additional additive error of at most ϵ/k for every input (x, y) . Protocol Π'_i uses private coins for the first $i-1$ rounds, and public coins for rounds i to k . In fact, Π'_i behaves the same as Π for the first $i-1$ rounds. Let Π'_{k+1} denote the original protocol Π .

We now describe the construction of Π'_i from Π'_{i+1} . Suppose the i th message in Π'_{i+1} is sent by Alice. Let M denote the random variable corresponding to the first i messages in Π'_{i+1} . M can be expressed as (M_1, M_2) , where M_2 represents the random variable corresponding to the i th message and M_1 represents the random variable corresponding to the initial $i-1$ messages. From Fact 1 (note that the distributions below are as in protocol Π'_{i+1} with the input distributed according to μ),

$$I(XY : M) = I(XY : M_1) + \mathbb{E}_{M_1} [I((XY : M_2) \mid M_1 = m_1)] = I(XY : M_1) + \mathbb{E}_{M_1 XY} [S(M_2^{xy m_1} \parallel M_2^{m_1})]$$

where $M_2^{xy m_1}$ denotes the distribution of M_2 when $(X, Y) = (x, y)$ and $M_1 = m_1$, and $M_2^{m_1}$ denotes the distribution of M_2 when $M_1 = m_1$. Note that the distribution of $M_2^{xy m_1}$ is independent of y , as Π'_{i+1} is private coin up to the i th round. Define $a_i \triangleq \mathbb{E}_{M_1 XY} [S(M_2^{xy m_1} \parallel M_2^{m_1})]$.

Protocol Π'_i behaves the same as Π'_{i+1} for the first $i-1$ rounds; hence Π'_i behaves the same as Π for the first $i-1$ rounds. In particular, it is private coin for the first $i-1$ rounds. Alice generates the i th message of Π'_i using a fresh public coin C_i as follows: For each distribution $M_2^{m_1}$, m_1 ranging over all possible initial $i-1$ messages, C_i stores an infinite sequence $\mathbf{\Gamma}^{m_1} \triangleq \langle \gamma_j^{m_1} \rangle_{j \in \mathbb{N}_+}$, where $(\gamma_j^{m_1} : j \in \mathbb{N}_+)$ are chosen independently from distribution $M_2^{m_1}$. Note that the distribution $M_2^{m_1}$ is known to both Alice and Bob as m_1 is known to both of them; so both Alice and Bob know which part of C_i to 'look' at in order to read from the infinite sequence $\mathbf{\Gamma}^{m_1}$. Using Lemma 5, Alice generates the i th message of Π'_i which is either $x_j^{m_1}$ for some j , or the dummy message 0. The probability of generating 0 is less than or equal to $\frac{\epsilon}{k}$. If Alice does not generate 0, her message lies in a set $\text{Good}_{x m_1}$ which has probability at least $1 - \frac{\epsilon}{k}$ in the distribution $M_2^{xy m_1}$. The probability of a message $m_2 \in \text{Good}_{x m_1}$ being generated is exactly the same as the probability of m_2 in $M_2^{xy m_1}$. The expected value of j is $2^{k(S(M_2^{xy m_1} \parallel M_2^{m_1})+1)/\epsilon}$. Actually, Alice just sends the value of j or the dummy message 0 to Bob, using a prefix free encoding, as the i th message of Π'_i . After Alice sends off the i th message, Π'_i behaves the same as Π'_{i+1} for

rounds $i + 1$ to k . In particular, the coin C_i is not ‘used’ for rounds $i + 1$ to k ; instead, the public coins of Π'_{i+1} are ‘used’ henceforth.

By the concavity of the logarithm function, the expected length of the i th message of Π'_i is at most $2k\epsilon^{-1}(S(M_2^{xy^{m_1}} \| M_2^{m_1}) + 1) + 2$ bits for each (x, y, m_1) (The multiplicative and additive factors of 2 are there to take care of the prefix-free encoding). Also in Π'_i , for each (x, y, m_1) , the expected length (averaged over the public coins of Π'_i , which in particular include C_i and the public coins of Π'_{i+1}) of the $(i + 1)$ th to k th messages does not increase as compared to the expected length (averaged over the public coins of Π'_{i+1}) of the $(i + 1)$ th to k th messages in Π'_{i+1} . This is because in the i th round of Π'_i , the probability of any non-dummy message does not increase as compared to that in Π'_{i+1} , and if the dummy message 0 is sent in the i th round Π'_i aborts immediately. For the same reason, the increase in the error from Π'_{i+1} to Π'_i is at most an additive term of $\frac{\epsilon}{k}$ for each (x, y, m_1) . Thus the expected length, averaged over the inputs and public and private coin tosses, of the i th message in Π'_i is at most $2k\epsilon^{-1}(a_i + 1) + 2$ bits. Also, the average error of Π'_i under input distribution μ increases by at most an additive term of $\frac{\epsilon}{k}$.

By Fact 1, $\sum_{i=1}^k a_i = I(XY : T) \leq a$, where $I(XY : T)$ is the mutual information in the original protocol Π . This is because the quantity $E_{M_1 XY}[S(M_2^{xy^{m_1}} \| M_2^{m_1})]$ is the same irrespective of whether it is calculated for protocol Π or protocol Π'_{i+1} , as Π'_{i+1} behaves the same as Π for the first i rounds. Doing the above ‘compression’ procedure k times gives us a public coin protocol Π'_1 such that the expected communication cost (averaged over the inputs as well as all the public coins of Π'_1) of Π'_1 is at most $2k\epsilon^{-1}(a + 1) + 2k$, and the average error of Π'_1 under input distribution μ is at most $\delta + \epsilon$. By restricting the maximum communication to $2k\epsilon^{-2}(a + 1) + 2k\epsilon^{-1}$ bits and applying Markov’s inequality, we get a public coin protocol Π'' from Π'_1 which has average error under input distribution μ at most $\delta + 2\epsilon$. By setting the public coin tosses to a suitable value, we get a deterministic protocol Π' from Π'' where the maximum communication is at most $2k\epsilon^{-2}(a + 1) + 2k\epsilon^{-1}$ bits, and the distributional error under μ is at most $\delta + 2\epsilon$. $\square$

Corollary 2. *Let $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$ be a function. Let μ be a product distribution on the inputs $\mathcal{X} \times \mathcal{Y}$. Let $\delta, \epsilon > 0$. Then, $IC_{\mu, \delta}^k(f) \geq \frac{\epsilon^2}{2k} \cdot C_{\mu, \delta+2\epsilon}^k(f) - 2$.*

Theorem 4 (Direct sum, k -round). *Let m, k be positive integers, and $\epsilon, \delta > 0$. Let $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$ be a function. Then, $R_\delta^k(f^m) \geq m \cdot \sup_{\mu, \kappa} \left(\frac{\epsilon^2}{2k} \cdot C_{\mu, \delta+2\epsilon}^k(f) - 2 - H(\kappa) \right)$, where the supremum is over all probability distributions μ on $\mathcal{X} \times \mathcal{Y}$ and partitions κ of μ .*

Proof. Immediate from Fact 5, Fact 4 and Corollary 2. $\square$

Corollary 3. *Let m, k be positive integers, and $\epsilon, \delta > 0$. Let $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$ be a function. Then, $R_\delta^k(f^m) \geq m \cdot \left(\frac{\epsilon^2}{2k} \cdot C_{[\cdot], \delta+2\epsilon}^k(f) - 2 \right)$.*

Remarks:

1. Note that all the results in this section hold even when f is a relation.
2. The above corollary implies that the direct sum property holds for constant round protocols for the pointer jumping problem with the ‘wrong’ player starting (the bit version, the full pointer version and the tree version), since the product distributional complexity (in fact, for the uniform distribution) of pointer jumping is the same as its randomised complexity [NW93,PRV01].

Acknowledgements. We thank Ravi Kannan, Sandeep Juneja and Siddhartha Bhattacharya for helpful discussions. We also thank the anonymous referees for their comments, which helped us to improve the presentation of the paper.

References

- [AS00] N. Alon and J. Spencer. *The probabilistic method*. John Wiley and Sons, 2000.
- [BJKS02] Z. Bar-Yossef, T. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 209–218, 2002.
- [CSWY01] A. Chakrabarti, Y. Shi, A. Wirth, and A. Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.
- [FKNN95] T. Feder, E. Kushilevitz, M. Naor, and N. Nisan. Amortized communication complexity. In *SIAM Journal of Computing*, pages 239–248, 1995.
- [JRS02] R. Jain, J. Radhakrishnan, and P. Sen. Privacy and interaction in quantum communication complexity and a theorem about the relative entropy of quantum states. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 429–438, 2002.
- [JRS03] R. Jain, J. Radhakrishnan, and P. Sen. A lower bound for bounded round quantum communication complexity of set disjointness function. Manuscript at quant-ph/0303138, 2003.
- [KKN92] M. Karchmer, E. Kushilevitz, and N. Nisan. Fractional covers and communication complexity. In *Structures in Complexity Theory '92*, pages 262–274, 1992.
- [KN97] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.
- [NW93] N. Nisan and A. Wigderson. Rounds in communication complexity revisited. *SIAM Journal of Computing*, 22:211–219, 1993.
- [PRV01] S. Ponzio, J. Radhakrishnan, and S. Venkatesh. The communication complexity of pointer chasing. *Journal of Computer and System Sciences*, 62(2):323–355, 2001.
- [Ros97] S. Ross. *Simulation*. Academic Press, 1997.
- [Yao79] A. C-C. Yao. Some complexity questions related to distributed computing. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 209–213, 1979.

Optimal Cache-Oblivious Implicit Dictionaries^{*}

Gianni Franceschini and Roberto Grossi

Dipartimento di Informatica, Università di Pisa
via Filippo Buonarroti 2, 56127 Pisa, Italy
{francesc,grossi}@di.unipi.it

Abstract. We consider the issues of implicitness and cache-obliviousness in the classical dictionary problem for n distinct keys over an unbounded and ordered universe. One finding in this paper is that of closing the longstanding open problem about the existence of an optimal implicit dictionary over an unbounded universe. Another finding is motivated by the antithetic features of implicit and cache-oblivious models in data structures. We show how to blend their best qualities achieving $O(\log n)$ time and $O(\log_B n)$ block transfers for searching and for amortized updating, while using just n memory cells like sorted arrays and heaps. As a result, we avoid space wasting and provide fast data access at any level of the memory hierarchy.

1 Introduction

The dictionary problem is a classical paradigm for studying the limitations and the characteristics of several computational models. In this problem a set of n distinct keys is maintained under insertions and deletions of individual keys while supporting searches. Several models assume that the keys are defined over an unbounded and ordered universe, and the only operations allowed on them are reads/writes and comparisons. Among others, implicit models and cache-oblivious models have recently stimulated a surge of interest.

Our first finding holds for the data structures in the implicit model [15,16], in which the only space usage is that of the keys with no waste of memory cells. Sorted arrays [14] and heaps [7,20] are the simplest form. While the term “implicit” originated in [16], it has also been the subject of papers taking a somewhat different point of view, including a long lists of results in perfect hashing [11,6], bounded-universe and succinct dictionaries [5,17,19], and cache-oblivious data structures [4]. These results use a model different from the model adopted in [16] and following papers. The latter model extends the comparison model, so that a suitable permutation of the n keys in a contiguous segment of n memory cells encodes the whole dictionary and *no* other information is explicitly required other than the keys themselves and $O(1)$ temporary RAM registers. The segment can be enlarged or shortened to the right by one cell at a

^{*} Work partially supported by the Italian MIUR project PRIN “ALINWEB: Algorithmics for Internet and the Web”.

time in constant time. It is a long-standing open problem how to implement an implicit dictionary in $O(\log n)$ time per operation. A number of results (e.g. [8, 9, 10, 15, 16]) neared to this objective. We give a positive answer by describing an implicit data structure, called *flat implicit tree*, which requires $O(\log n)$ time for searching and $O(\log n)$ amortized time for updating, with just $O(1)$ RAM registers needed to operate dynamically.

Our second finding relates to the data structures in the cache-oblivious (memory-hierarchy) model [12], such as cache-oblivious B-trees [1] and other dictionaries (e.g., [3, 4, 2]). In this ideal model there are two levels of memory hierarchy, where one level is small and fast and the other level is large but slow. Data transfers between the two levels occur in blocks of B data items; however, the value of B and the capacity of the memory level are *unknown* to the algorithms operating in the model. Apparently, the two models above are antithetic. On one hand, implicit dictionaries loose data locality when permuting the keys. Due to the dynamic maintainance of the permutation without wasting memory cells, their algorithms give rise to irregular access patterns jumping from one memory cell to another. On the other hand, cache-oblivious dictionaries carefully handle irregular access patterns for getting locality at the price of wasting $\Theta(n)$ memory cells. In order to preserve dynamically the locality of keys, these are suitably interleaved with empty cells whose purpose consists in delaying more expensive redistributions that have a small amortized cost in this way. In this paper, we show that our flat implicit trees combine together the appealing features of the two models, avoiding their contrasting drawbacks mentioned above. In fact, our data structure requires $O(\log n)$ time and $O(\log_B n)$ block transfers for searching, and $O(\log n)$ amortized time and $O(\log_B n)$ amortized block transfers for updating, while using just n memory cells like sorted arrays and heaps. Not only we avoid space wasting; we provide optimal data access at any level of the memory hierarchy.

Compared to previous work, our flat implicit tree is complementary to that in [9] achieving $O(\log_B n)$ bounds for $B = \Omega(\log n)$, which is not restrictive in real situations. The bounds in [9] are worst case in the cache-aware model and, moreover, scanning r keys takes $O(\log_B n + r/B)$ block transfers. The pointerless data structure in [4] is cache-oblivious but, as noted by its authors, it is not properly implicit as it occupies $(1 + \epsilon)n$ cells for any $\epsilon > 0$, while permitting an $O(1 + r/B)$ scanning cost. Our data structure does not support efficient scanning, but this is also an open problem in cache-oblivious data structures alone when the bounds are $O(\log_B n)$ as in our case.

The paper is organized as follows. In Section 2, we give an overview of our data structure. Section 3 describes the bottom layer while Section 4 discusses the top layer of the structure. We put all together in Section 5 for our final bounds.

2 Overview of the Cache-Oblivious Implicit Dictionary

We encode data by a pairwise (odd-even) permutation of keys [15]. A pointer or an integer of b bits is encoded by distinct keys $x_1, y_1, x_2, y_2, \dots, x_b, y_b$, so that the

i th bit is 0 when $\min\{x_i, y_i\}$ precedes $\max\{x_i, y_i\}$ and it is 1 when $\max\{x_i, y_i\}$ precedes $\min\{x_i, y_i\}$. From a high-level point of view, our implicit dictionary is a suitable collection of *chunks* [15,9,8] and *spare* keys. Each chunk contains k keys pairwise permuted for encoding a constant number of integers and pointers, each of $b = O(\log n)$ bits. The keys in any chunk belong to a certain interval of values, and the chunks are pairwise disjoint when considered as intervals. Hence, we can write $c_1 < c_2$ for any two chunks meaning that the keys in chunk c_1 are all smaller than those in chunk c_2 . Not all the keys are stored in the chunks. A number of $O(\log n)$ keys are kept in a preamble $\mathcal{P}$ to encode some bookkeeping information and the remaining ones are the spare keys, which are kept together in a contiguous segment of memory without any particular organization.

We keep the invariant that the number n of keys satisfies $n'/4 < n < n'$, where n' is a power of two, thus fixing the chunk size $k = \Theta(\log n')$. We can avoid to keep the value of n and n' explicitly, as a variable-length encoding, such as the δ -code, can represent them asymptotically in the preamble $\mathcal{P}$. The algorithms for maintaining the data structure are parametric in n' and k . We resize k only when either $n = n'/4$ or $n = n'$; this event marks the beginning of a new *epoch*. Hence, the lifetime of the data structure can be divided into epochs. At the beginning of each epoch, we start the process of in-place global rebuilding our implicit dictionary to guarantee that $n'/4 < n < n'$. After each rebuilding, the n distinct keys are organized in two layers.

The *top* layer is the *super-root* containing $a = \Theta(n/k^2)$ chunks, called *actual* chunks, where a is *always* a power of two. Also, it contains a number of *virtual* chunks stored in no particular order. There are $O(1)$ virtual chunks associated with each actual chunk. Specifically, for an actual chunk c , we require that c and its associated virtual chunks are consecutive in the total order defined over all the chunks in the top layer. We describe this layer in Section 4.

The *bottom* layer is a dynamic implicit forest, where each tree implements a bucket of $\Theta(k^2)$ keys organized into chunks, called *bucket* chunks, plus the spare keys being handled differently. Each bucket tree is organized into $O(1)$ levels. The root is either an actual chunk or a virtual chunk in the top layer. The keys in the descendants of the root are in the bottom layer. Specifically, the root has a single child, called *intermediate* node, containing $\Theta(\sqrt{k})$ bucket chunks and $\Theta(\sqrt{k})$ leaves as children. Each such leaf contains $\Theta(\sqrt{k})$ bucket chunks and has associated $\Theta(\sqrt{k})$ spare keys and a *manipule* of $\Theta(k)$ keys. The buckets are pairwise disjoint when considered as intervals. More details are in Section 3.

While searching a key in this organization, we identify an actual chunk and, hence, its $O(1)$ associated virtual chunks in the top layer. Among them, we determine the root of the bottom-layer bucket for completing the search. As customary to implicit data structures, we describe the memory layout of our dictionary since this heavily affects the complexity of the operations:

- The preamble $\mathcal{P}$ of $O(k)$ keys encoding $O(1)$ pointers and integers for book-keeping purposes.
- The *root area* for the top layer storing first the keys in the actual chunks in a suitable order and, then, the virtual chunks in no particular order. Recall

that these chunks are the roots of the bucket trees. The area may grow or shrink by k positions to its right.

- The area for the bottom layer, divided into *node area* (for the intermediate nodes and the leaves), *manipulate area* and *spare area* for the bucket trees. The whole area for the bottom layer may grow or shrink by k positions to its left and by one position to its right.

We introduce the *compactor zones* for handling the above areas as each compactor zone stores objects of the same size in a *contiguous* segment of memory, which is crucial to achieve our bounds.

Theorem 1. *There exists a dynamic data structure storing n distinct keys that is both implicit and cache-oblivious and that supports the following operations with just $O(1)$ registers:*

- *searching with a cost of $O(\log n)$ time and of $O(\log_B n)$ block transfers per operation;*
- *inserting and deleting with an amortized cost of $O(\log n)$ time and of $O(\log_B n)$ block transfers.*

We use a primitive for *cumulative shifts* for a set of contiguous keys $x_1, \dots, x_m, y_1, \dots, y_r$ in a segment of $m + r$ cells. Letting $X = x_1, \dots, x_m$, and $Y = y_1, \dots, y_r$, we want to perform an in-place operation that starts from XY and obtains YX in the same segment of memory. Since $YX = (X^R Y^R)^R$, where R denotes the reversal of the sequences, we can easily reverse the given sequences by swapping their keys in a total of $O(m+r)$ time and $O((m+r)/B)$ block transfers.

3 Bottom Layer: Buckets as Implicit Dynamic Forest

We describe the bottom layer storing the buckets in the form of an implicit dynamic forest. Each bucket is a tree whose root is either an actual or a virtual chunk in the top layer. We need to insert, to delete and to search a key in a bucket, as required by the dictionary problem. In case of bucket overflows (too many keys) and bucket underflows (too few keys), we need splitting the bucket or merging/borrowing with a neighbor bucket, while preserving implicitness and cache-obliviousness.

3.1 Bucket Organization

As previously mentioned, each bucket is a tree made up of bucket chunks and spare keys, except for the root, which is either an actual or a virtual chunk. Each intermediate node varies from $k\sqrt{k}$ to $4k\sqrt{k}$ keys (i.e., $\sqrt{k}$ to $4\sqrt{k}$ bucket chunks) and is the only child of its root. The number of leaves that are children of the intermediate node varies from $\sqrt{k}$ to $4\sqrt{k}$, one leaf per chunk. The pointer to the i th child leaf is encoded by $O(\log n)$ keys in the i th chunk of the intermediate node. Leaves contain from $k\sqrt{k}$ to $4k\sqrt{k}$ keys like intermediate

nodes. In addition, the first $\sqrt{k}$ chunks of each leaf have associated from 1 to 5 spare keys each. Given one such chunk seen as an interval, its spare keys belong to that interval. The leaf has also associated a manipule containing from k to $5k$ keys and varying by $\sqrt{k}$ keys at time. The keys in the manipule are larger than those in the leaf.

We keep the above invariants on the number of keys during the updates. When inserting a key, we increment by 1 the number of spare keys. If a chunk in a leaf v has associated 6 spare keys, we redistribute the keys in v . If v has associated a total of $5\sqrt{k} + 1$ spare keys, we redistribute the keys between v and its associated manipule z , so that v has $\sqrt{k}$ spare keys less. When v contains $4k\sqrt{k}$ keys and z contains $5k$ keys, we split v and z creating two leaves with their associated spare keys and two manipules. All satisfy the invariants on their number of keys, which is roughly half on the way between the minimum and maximum allowed.

The split may cause a chunk c to be inserted in the intermediate node u , parent of v . If u contains $4k\sqrt{k} + k$ keys, we need to split u as well. We create two buckets from the current bucket and the chunk c' resulting from the split of u becomes the root of the new bucket with the largest keys. Finally, we add c' to the root area as actual or virtual chunk. Deleting a key is analogous, except that we merge or borrow instead of splitting u and v ; borrowing in v involves an individual key whereas in u it involves an individual chunk. We now detail how to handle the keys.

3.2 Memory Layout

We postpone the layout of the roots to Section 4. Here we discuss the layout of the intermediate nodes, the leaves, the manipules, and the spare keys in the bottom layer. We store the spare keys in no particular order in the spare area. Using *compactor zones* (or, simply, *zones*) we accommodate the internal nodes and the leaves in the node area and the manipules in the manipule area. What we describe next for the nodes holds also for the manipules.

We pack together the nodes of *identical* size s , embedding them in a suitable zone devoted to nodes of size s and called *zone s* . When a node changes size, it also changes zone. Each node in zone s occupies a contiguous segment of s memory cells, except possibly the node at the beginning of each zone. In this case, we maintain the property that the node is stored in two segments of s_1 and s_2 cells, respectively, where $s_1 + s_2 = s$. The last s_2 keys of the nodes are at the beginning of zone s and the first s_1 keys of the node are at the end of zone s . We call *broken* this node and *unbroken* any other node in the zone. Hence, all nodes are unbroken except possibly one node per zone. The (encoded) pointer to a broken node contains extra bits to encode that value of s_1 .

The zones share common techniques; for instance, we employ the primitive for rotating a zone s . Suppose we have m keys to the right of zone s , and let X be the memory segment hosting the whole zone s along with the m keys to its right. A *rotation* is an in-place primitive that incrementally moves the m keys to the beginning of X and the first m keys in zone s to the end of X (or vice versa)

without scanning the whole X , contrarily to what done by the cumulative shifts. For $i = 1, 2, \dots, m$, it exchanges the i th key of zone s with the i th among the m keys. At the end, the broken node (if any) in zone s may become unbroken and at most one unbroken node may become broken. We search one key for each of the latter nodes re-encoding the $O(1)$ pointers to it that are identified through the search. We also re-encode the starting position of zone s . The cost of a rotation is $O(k+m)$ time and $O(\lceil (k+m)/B \rceil)$ block transfers, plus the cost of $O(1)$ searches. We now give more details on the zones.

Node area. It contains $3\sqrt{k} + 1$ compactor zones in increasing order of index s . Each zone s is adjacent to zone $s - k$ (to its left) and to zone $s + k$ (to its right), since the size of the nodes is a multiple of k . The starting positions of all the zones in this area are encoded in the first $\sqrt{k}$ chunks of the area itself. We support the following basic primitives in this area:

EXTRACTNODE(w) extracts node w placing it between the node area and the maniple area. Let s be the size of w . We exchange w with the rightmost unbroken node in zone s (note that w may be the broken node). Now, we have w followed by the initial portion of the broken node (if any) in zone s and by zone $s + k$ (if any). We exchange these $O(k\sqrt{k})$ keys by cumulative shifts in $O(k\sqrt{k})$ time and $O(k\sqrt{k}/B)$ block transfers, with the initial portion of the broken node in s followed by w and by zone $s + k$. We need to perform $O(1)$ searches to re-encode $O(1)$ pointers in their parents. As a result, we shorten zone s by s positions to the right and have w between zones s and $s + k$. For $s' = s + k, s + 2k, \dots, 4k\sqrt{k}$ (i.e., incrementing s' by k), we move w from the left to the right of zone s' by a rotation and update the new starting positions of zone s' . At the end, we have w to the right of zone $4k\sqrt{k}$, that is, between the node area and the maniple area. The total cost is $O(k^2)$ time and $O(\sqrt{k} \lceil k\sqrt{k}/B \rceil)$ block transfers, plus the cost of $O(\sqrt{k})$ searches.

INSERTNODE(w) inserts node w into its suitable compactor zone and is similar to **EXTRACTNODE** (with the same cost), where w is between the node area and the maniple area.

TRANSFERCHUNK(c) transfers chunk c from an area to another. Either c is between the node area and the maniple area and we want it laying between the root area and the node area, or vice versa. We proceed like in **EXTRACTNODE** and **INSERTNODE**. However, we do not insert or delete c inside any zone, but we simply rotate each zone s' to move c from one side to another of zone s' . Note that, since c has size k , the total cost is $O(k\sqrt{k})$ time and $O(\sqrt{k} \lceil k/B \rceil)$ block transfers, plus the cost of $O(\sqrt{k})$ searches.

Maniple area. It contains $4\sqrt{k} + 1$ compactor zones in increasing order of index s . Each zone s contains the maniple of identical size s . Its neighbors are zone $s - \sqrt{k}$ (to its left) and zone $s + \sqrt{k}$ (to its right), since the size of the maniples is a multiple of $\sqrt{k}$. The starting points of all the zones are encoded in the first $\sqrt{k}$ chunks of the node area. The primitives here supported are:

EXTRACTMANIPLE(z) extracts maniple z and place it either between the node area and the maniple area, or between the maniple area and the spare area.

It is analogous to `EXTRACTNODE`, but it takes $O(k\sqrt{k})$ time and $O(\sqrt{k} \lceil k/B \rceil)$ block transfers, plus the cost of $O(\sqrt{k})$ searches.

`INSERTMANIPLE( $z$ )` inserts manipule z into its zone analogously to `INSERTNODE`, where z is either between the node area and the manipule area, or between the manipule area and the spare area. Its cost is $O(k\sqrt{k})$ time and $O(\sqrt{k} \lceil k/B \rceil)$ block transfers, plus the cost of $O(\sqrt{k})$ searches.

`TRANSFERKEYS( $m$ )` transfers m contiguous keys from the left of the manipule area to its right, or vice versa, where $m \leq k$. It is like `TRANSFERCHUNK`, except that each zone rotates by m positions. The total cost is $O(k\sqrt{k})$ time and $O(\sqrt{k} \lceil k/B \rceil)$ block transfers, plus the cost of $O(\sqrt{k})$ searches.

Spare area. Here there are no compactor zones, but the spare keys are stored contiguously without any specific order. So, the basic primitives are simple to describe. One primitive inserts a new spare key to the right of the spare area and extends the right border of the area to include the new spare key. Another primitive extracts a spare key leaving a hole inside the spare area that is filled with the rightmost spare key in the area, thus shortening the right border of the spare area by one position. In all cases, we search the key to update its pointer encoded in a suitable chunk of a leaf. So, its cost is $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers plus the cost of $O(1)$ searches. We may also want to collect m spare keys between the manipule and the spare area. In that case, we can see this operation as a sequence of m extractions, giving a total cost of $O(km)$ time and $O(m \lceil k/B \rceil)$ block transfers plus the cost of $O(m)$ searches.

3.3 Node Management

The internal structure of the nodes in the bucket trees allows for quick searching and updating even though their size is $\Theta(k\sqrt{k})$. We give more details on how to achieve this goal.

The simplest organization is that of the internal nodes. Given an internal node u , containing $t = \Theta(\sqrt{k})$ chunks $c_1, \dots, c_t$, we keep a directory in the first $2t$ positions of u containing the smallest and the greatest key of $c_1, \dots, c_t$, respectively, in this order. If the keys in a chunk c_j change, it is a simple task to update c_j and the directory in $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers. Moreover, given a node u with all of its keys in sorted order, we can build the directory for u by shifting the leftmost and rightmost key in each chunk, in a total of $O(k^2)$ time and $O(\sqrt{k} \lceil k\sqrt{k}/B \rceil)$ block transfers. Routing a search for key x in u examines some of the $O(t)$ keys in the directories to identify a pair of keys x_1 and x_2 such that $x_1 \leq x \leq x_2$. These two keys belong to at most two consecutive chunks c_j and c_{j+1} that are accessed by simple offsetting. The cost for routing is $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers.

The leaves undergo a more involved organization because they have associated spare keys. We give a two-step description, in which we first describe how to maintain the invariant on the number of spare keys in $O(\log n)$ time and then how to make this organization cache-oblivious in $O(\log_B n)$ block transfers. For the sake of discussion, let's assume that the number of spare keys in a leaf v is

non-maximum (i.e., less than $5\sqrt{k}$ keys in the first $\sqrt{k}$ chunks) and that we have to add one more key x to a chunk c in v that has the maximum number of spare keys, i.e., 5. Among the first $\sqrt{k}$ chunks in v , let c' be the nearest chunk, say, at some position to the right of c , so that c' has less than 5 spare keys associated. What we have to do is inserting x into c by shifting its keys while extracting the maximum key in c , which we insert into the chunk just to the right of c . For any chunk c'' lying between c and c' exclusive, we want to insert into c'' the minimum key (extracted as the greatest from the chunk to the left of c'') while extracting the maximum key (being inserted as the smallest into the chunk to the right of c''). When we reach c' , we insert into it the key arriving from the chunk to its left and we shift its keys to add one more spare key into c' . Consequently, associating one more spare key with c can be translated into increasing the number of spare keys in c' (if any). While we can shift the keys in c and c' , we cannot afford the $O(k)$ cost of shifting all the keys in the intermediate chunks c'' between c and c' , as they can be $\Theta(\sqrt{k})$ in number.

We organize each chunk c'' in v as follows, denoting by x_1 the minimum key to be inserted into c'' and by x_2 the maximum key to be extracted from c'' in the generic iteration step:

1. Keys $a_1, a_2, \dots, a_k$ are kept *rotated* by an *offset* r , occurring as $a_{r+1} \cdots a_k \cdot a_1 \cdots a_r$ in c'' .
2. Inserting x_1 while extracting $x_2 = a_k$ gives $a_{r+1} \cdots a_{k-1} \cdot x_1 a_1 \cdots a_r = a'_{r+2} \cdots a'_k \cdot a'_1 \cdots a'_r a'_{r+1}$.
3. Either $1 \leq r \leq \sqrt{k}$ or $k - \sqrt{k} + 1 \leq r \leq k$ (i.e., keys $a_{\sqrt{k}+1} \cdots a_{k-\sqrt{k}}$ are not rotated).
4. $a_{\sqrt{k}+1}, \dots, a_{k-\sqrt{k}}$ encode the pointers to the spare keys y for c'' , and $a_{\sqrt{k}+1} < y < a_{k-\sqrt{k}}$.

The cost of implementing the above rotation is $O(\log k)$ time since we need to recover the value of the offset r encoded in $O(\log k)$ keys of c'' . Immediately before that the condition 3 is violated, we can restore that condition by scanning all the keys in c'' in $O(k)$ time, possibly changing the spare keys and re-encoding the pointers to them. The symmetric operation of deleting x_1 and inserting x_2 is analogous.

Our organization is not yet suitable for cache-obliviousness because encoding and decoding the offsets for rotations of the chunks require an access to the keys in each chunk, which is a problem for small values of the (unknown) block size B . To make it cache-oblivious, we form a directory at the beginning of leaf v by a cumulative shift of $a_1, \dots, a_{\sqrt{k}}$ and $a_{k-\sqrt{k}+1}, \dots, a_k$ for each c'' . We obtain a larger directory than that of internal nodes, still the number of keys in the directory is $O(k)$. As a result, the rotation of each chunk c'' will always occur inside that directory. We can build from scratch the directory for v in sorted order by applying a cumulative shift to the $\sqrt{k}$ leftmost and the $\sqrt{k}$ rightmost keys in each chunk, in $O(k^2)$ time and $O(\sqrt{k} \lceil k\sqrt{k}/B \rceil)$ block transfers. Note that all rotations are set to zero after the operation. Routing a search key x

inside v uses the directory, in $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers. The primitives here supported are:

INSERTKEY(x, v) with the algorithms just mentioned. Note the number of keys in v does not change. The new spare key of c' is hosted in the spare area described in Section 3.2. The cost is $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers plus $O(1)$ searches, unless condition 3 is violated for a chunk in v . In the latter case, at least $\Omega(\sqrt{k})$ updates occurred in that chunk and the cost of $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers for restoring that condition with a cumulative shift amortizes (divided by $\sqrt{k}$).

EXTRACTKEY(v, x) is executed analogously to **INSERTKEY**, so that the cost is the same. After the operation, x is the rightmost key in the spare area.

3.4 Bucket Management

We now discuss the operations on the buckets. Searching key x after visiting the root of the bucket in the top layer consists of routing x inside the intermediate node u , which is the only child of the root. If we find the key as a member of a chunk in u , we are done. Otherwise, we identify a chunk c_j in u , reaching the leaf v whose pointer is encoded in c_j . We route x also inside v . Here, either x is a member of a chunk in v or it is a spare key in it; or it is larger than the keys in v and so we scan the keys in its manipule z . Otherwise, we can infer that x is not stored in the bucket. Since routing requires $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers in u and v (see Section 3.3), searching x in a bucket takes $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers.

We now discuss the insertion of x , in which we first search x identifying a leaf v (if x 's position is inside a chunk c of the intermediate node u , we extract the rightmost key in c and set x to be this key, which should be inserted to v). If the number of spare keys in v is less than $5\sqrt{k}$, we can insert x into v using **INSERTKEY**(x, v). Otherwise, we have to reorganize v , its associated manipule z and its spare keys, so that the number of spare keys in v reduces to less than $5\sqrt{k}$.

Case 1: Manipule z contains less than $5k$ keys. We move the largest $\sqrt{k}$ keys in v to z as follows. We remove the largest $\sqrt{k}$ keys in v using **EXTRACTKEY** for $\sqrt{k}$ times. Each time we extract one such key, we find it as the rightmost key in the spare area. We then move incrementally these $\sqrt{k}$ extracted keys from the end of the spare area to its beginning, thus shortening the spare area by $\sqrt{k}$ positions to the left. Now, these keys are between the manipule area and the spare area. We run **EXTRACTMANIPULE**(z) moving z between the manipule area and the spare area, so that z and the $\sqrt{k}$ keys are contiguous. At this point, we add the latter keys to z and then insert back z into the manipule area by performing **INSERTMANIPULE**(z).

Case 2: Manipule z contains exactly $5k$ keys and leaf v contains less than $4k\sqrt{k}$ keys. We lower the number of spare keys in v by redistributing some keys between v and z . We proceed as in case 1, except that **EXTRACTMANIPULE**(z)

moves z between the node area and the maniple area. Then, we execute $\text{TRANSFERKEYS}(\sqrt{k})$ to move the $\sqrt{k}$ keys from their position between the maniple area and the spare area to their new position between the node area and the maniple area, near to z . We remove the first k keys of the just extended z to form a chunk c , with the smallest keys in it. We then insert back z in the maniple area by performing $\text{INSERTMANIPLE}(z)$. Next, we run $\text{EXTRACTNODE}(v)$ bringing v near to c between the node area and the maniple area. We add c to v by a cumulative shift to move c to its correct positions inside v . We then perform a cumulative shift of the leftmost and the rightmost $\sqrt{k}$ keys in c to their position in the directory of v described in Section 3.3. We then insert back v with $\text{INSERTNODE}(v)$.

Case 3: Maniple z contains exactly $5k$ keys and leaf v contains exactly $4k\sqrt{k}$ keys. We create two leaves v_1 and v_2 , their maniples z_1 and z_2 and their spare keys from those in v and z and from the spare keys associated with v . We possibly propagate the split to u , the parent of v . If also u splits, we create two buckets from the current one. We now detail these steps.

We proceed in a way similar to cases 1–2, moving v , z and the spare keys to the position between the node area and the maniple area. We perform an in-place merge of the keys in v and the sorted sequence of the spare keys. As a result, we have a sorted sequence of keys by reading v , the spare keys and z in this order. We divide this sequence in six parts: $v_1, s_1, z_1, c, v_2, s_2, z_2$, where c is the median chunk to be inserted in v 's parent u . Leaf v_1 contains $2k\sqrt{k}$ keys and has associated maniple z_1 of $2k$ keys and $2\sqrt{k}$ spare keys in s_1 . Leaf v_2 contains $2k\sqrt{k}$ keys and has associated maniple z_2 of $2k + \sqrt{k}$ keys and $2\sqrt{k} + 1$ spare keys in s_2 . We perform cumulative shifts in these $O(k\sqrt{k})$ keys obtaining $v_1, v_2, c, s_1, s_2, z_1, z_2$ in this order (we fix $m = k$ or $m = \sqrt{k}$ in the cumulative shifts according to the cases). We build the directories of v_1 and v_2 as described in Section 3.3. We then reinsert v_1 and v_2 into the node area with INSERTNODE , and z_2 and z_1 into the maniple area with INSERTMANIPLE . We are left with the keys in c, s_1, s_2 between the node area and the maniple area. We apply TRANSFERKEYS moving the keys in s_1, s_2 to the positions between the maniple area and the spare area. We extend the spare area by $|s_1| + |s_2|$ positions to the left (it was shortened by the several calls to EXTRACTKEY). Since the keys in s_1 and s_2 are not spare keys, we execute $\text{INSERTKEY}(x, v_1)$ for $x \in s_1$ and $\text{INSERTKEY}(x, v_2)$ for $x \in s_2$. As mentioned in Section 3.2, INSERTKEY adds a spare key x' at the end of the spare area. We move x' to the memory cell freed by x after its insertion.

At this point, we are left with handling c between the node area and the maniple area. Let u be the intermediate node, parent of v . We move u near to c between the node area and the maniple area by executing $\text{EXTRACTNODE}(u)$. If u has less than $4k\sqrt{k}$ keys, we add c to u by a cumulative shift, analogously to what done for v in case 2, noting that its reorganization is much simpler (see Section 3.3). If u has exactly $4k\sqrt{k}$ keys, we add c to u by a cumulative shift and apply an in-place merging of the resulting directory and chunks. Now the keys in the just extended u are sorted, so we split them into u_1, c' and u_2 in this

order. We build the internal directories of u_1 and u_2 as described in Section 3.3. Each of u_1 and u_2 contains $2k\sqrt{k}$ keys, while c' is the median chunk. We have thus created two buckets, and c' is the root of the new bucket containing u_2 . We insert u_1 and u_2 into the node area using INSERTNODE. We also move c' using TRANSFERCHUNK so that it reaches the position between the root area and the node area, becoming part of the root area as actual or virtual chunk as described in Section 4.

The amortized cost for the insertion is informally given by the worst-case cost spread among $\Omega(\sqrt{k})$ updates in case 1, among $\Omega(k)$ updates in case 2, and among $\Omega(k\sqrt{k})$ updates in case 3, respectively. As a result, the amortized cost for inserting a key into a bucket is $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers.

It remains to discuss how to implement the deletion of key x in a bucket. Let u be the intermediate node and v be the leaf reached by the search. If x belongs to a chunk c of u , we delete it, set x to be the leftmost key in v , add x to c , and recursively remove x from v . Hence we solve the problem of deleting x from v . We have four cases; three of them are the exact counterpart of cases 1–3 discussed for the insertion and dealing with merging leaves and nodes. In addition, the fourth case deals with borrowing, provided that borrowing in v involves an individual key whereas in u it involves an individual chunk. The amortized cost for deleting a key from a bucket is $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers.

Theorem 2. *In the bottom layer, each bucket contains $\Theta(k^2)$ keys at any time. Searching a bucket takes $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers. The amortized cost of updating a bucket is $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers per insert/delete operation. At any time, only $O(1)$ RAM registers are required to operate dynamically.*

4 Top Layer: Cache-Obliviousness

The top layer contains the super-root of the flat implicit tree and collects all the actual chunks and the virtual chunks. We remark that these chunks are the roots of the buckets in the bottom layer discussed in Section 3. The memory layout in the super-root area is simple; first, all the actual chunks in sorted order and, then, all the virtual chunks in no particular order. We describe in this section how to handle efficiently the super-root in a cache-oblivious fashion.

4.1 Actual Chunks and Virtual Chunks

The a actual chunks are stored in sorted order in the first ak positions of the super-root area, where $a = O(n'/k^2)$ is *always* a power of two. Each actual chunk has associated at most $\alpha = O(1)$ virtual chunks, which are the nearest in the order of the (actual and virtual) chunks. They are kept in a linked (sorted) list starting from the actual chunk. The rest of the area contains the virtual chunks in no particular order as the linked lists allow their retrieval. The super-root area resizes by k positions to the right at a time to make room for one more or less

chunk after bucket splitting or merging. The number a of actual chunks changes only when rebuilding (see Section 5) or when performing a full redistribution of actual and virtual chunks. Since the actual chunks are kept sorted, we can route a key to its (actual or virtual) chunk in the top layer in $O(\log n')$ time.

We now discuss how to make cache-oblivious the access to the actual chunks since the virtual chunks are not much of a problem. In the following, we assume that the rightmost actual chunk is treated separately and we are left with $a - 1 = 2^h - 1$ actual chunks. The main idea is to build an internal directory for the root similarly to what done for internal nodes u in Section 3.3. We refer to actual chunks both when they contain k keys or when they have only $k - 2$ keys since their smallest and greatest key are in the directory. It is worth noting that the directory is permuted while the actual chunks are kept in increasing order. Moreover, the keys in the directory are located between the actual chunks and the virtual chunks. Conceptually, we treat each pair of keys in the directory as a single interval. When searching a key x , we compare it to each interval by exploiting the fact that the intervals are disjoint: either x is inside the interval, or it is to the left or the right of the interval. If we have cache-oblivious access to the directory, we can access an actual chunk and its associated virtual chunks in $O(k)$ time and $O(\lceil k/B \rceil)$ block transfers.

We focus therefore on how to permute the directory assuming to have just $2^h - 1$ keys (rather than $2^h - 1$ pairs of keys encoding disjoint intervals) for the sake of discussion.

4.2 Building the VEB-Permutation

We define the *Van Emde Boas permutation* (shortly, VEB-permutation) of $2^h - 1$ keys following Prokop's recursive scheme for Van Emde Boas trees [18]. Suppose to have a complete binary tree with $h \geq 1$ levels and $2^h - 1$ nodes, where $h = 1$ indicates that the tree has just one node. The nodes store the sorted sequence of $2^h - 1$ keys in symmetric order. Since we do not keep this tree anywhere, we permute the keys recursively according to the tree structure. In what follows, let A denote the memory segment hosting these $2^h - 1$ keys with the scheme of the VEB-permutation and let VEB-tree indicate the complete binary tree mentioned above. If $h = 1$, we simply store the key associated with the unique node in the VEB-tree. Otherwise, let $h_T = \lceil h/2 \rceil$ and $h_B = h - h_T$. We recursively store the $2^{h_T} - 1$ entries in the top tree of height h_T in the first $2^{h_T} - 1$ cells of A . Then, for $i = 0, 1, \dots, 2^{h_T} - 1$, we recursively store the $2^{h_B} - 1$ entries of the bottom tree number i (from left to right) in the i th portion of A (i.e., starting from $A[2^{h_T} + i \cdot (2^{h_B} - 1)]$).

We now describe how to build a VEB-permutation *in-place* in $O(2^h h^2) = O(a \log^2 a) = O(n)$ time and block transfers. (We can achieve a bound of $O(a \log a)$ but this does not improve the final bounds.) We first run heapsort on the sequence of $2^h - 1$ entries. We then apply the recursive scheme mentioned above. The base case for $h = 1$ is easy to handle, so let's suppose $h > 1$ and compute h_T and h_B . For $j = 1, \dots, 2^{h_T} - 1$, we swap the key in position j with the key in position $j \cdot 2^{h_B}$. Now, the keys associated with the top tree are in

```

FIND( $x, h$ ):
1: IF  $h = 1$  THEN
2:   IF  $x \leq A[i]$  THEN
3:      $bfs \leftarrow bfs \times 2$ 
4:   ELSE
5:      $bfs \leftarrow bfs \times 2 + 1$ 
6: ELSE
7:    $h_T \leftarrow \lceil h/2 \rceil$ ,  $h_B \leftarrow h - h_T$ 
8:   FIND( $x, h_T$ )
9:    $rank \leftarrow bfs \bmod 2^{h_T}$ 
10:   $i \leftarrow i + (2^{h_B} - 1) \times rank + 1$ 
11:  FIND( $x, h_B$ )
12:   $bfsroot \leftarrow bfs / 2^{h_B+1}$ 
13:   $rankroot \leftarrow bfsroot \bmod 2^{h_T}$ 
14:   $i \leftarrow i + (2^{h_B} - 1) \times (2^{h_T} - 1 - rankroot)$ 

```

Fig. 1. Procedure FIND to search x in a VEB-permutation of $2^h - 1$ keys stored.

order in the first $2^{h_T} - 1$ positions of A . We execute the heapsort of the keys in the rest of A (i.e., $A[2^{h_T} \dots 2^h - 1]$). We then recursively apply our construction to the keys in $A[1, \dots, 2^{h_T} - 1]$ (the top tree) and, for $i = 0, 1, \dots, 2^{h_T} - 1$, to the keys in $A[2^{h_T} + i(2^{h_B} - 1) \dots 2^{h_T} + (i + 1) \cdot (2^{h_B} - 1) - 1]$ (the bottom subtree number i).

The cost of this construction is asymptotically upper bounded by the solution to the recurrence $C(2^h - 1) = C(2^{h_T} - 1) + 2^{h_T} C(2^{h_B} - 1) + O(2^h h)$. For a suitable constant d , that solution is $C(2^h - 1) \leq d 2^h h^2$ by substitution on the right hand side of the recurrence. We remark that the construction of the VEB-permutation is not fully in-place as it uses the recursion. We therefore handle directly the recursion by using a stack storing only the pairs of values h_T, h_B thus found at each recursive level. Since h_T and h_B can be encoded in $O(\log h)$ bits and we keep $O(\log h)$ of them during the recursion, the total number of bits for the full stack is $O(\log^2 h) = o(\log n)$. We can handle this stack in $O(1)$ registers in constant time per operation, using simple arithmetic operations for push and pop. We do not violate the assumptions of the implicit model, as temporary information can be stored in $O(1)$ RAM registers. Using this “implicit” stack in a register and additional $O(1)$ register, we are able to build in-place the VEB-permutation in $O(n)$ time and block transfers.

4.3 Searching the VEB-Permutation

We now get to the main point, namely, how to search a key x in the VEB-permutation for $2^h - 1$ keys. In [18] it is shown that traversing the path from the root to a node takes $O(\log_B(2^h - 1)) = O(\log_B n)$ block transfers. We show how to do it without the extra information required in [4], which is not permitted in the model for implicit data structures. We use procedure FIND(x, h) in Figure 1 to achieve our goal. Before invoking it, we know that the segment $M = A[i \dots i +$

$2^h - 2]$ of $2^h - 1$ keys corresponds to a subtree S of the VEB-tree. We also know that $A[i]$ is the key in the root of S , and that the breadth-first number of $A[i]$ in the VEB-tree is bfs . (Initially, S is the VEB-tree, $M = A$, and so $i = 1$ and $bfs = 1$.)

When $\text{FIND}(x, h)$ completes, it has traversed the path from the root of S to one leaf v of S and i has reached the position of the last key in M . The routing of key x must go on either to the left or to the right of v in the rest of the VEB-tree, and we crucially know the bfs of the next node to visit. In any case, $bfs \bmod 2^h$ gives the rank of x among the keys in M . Identifying the position j such $A[j] = x$ (if any) is a minor modification.

We now show how to keep the invariant on $\text{FIND}(x, h)$ by induction on h (see Figure 1). If $h = 1$, this is immediate. Let's take the case of $h > 1$. We compute the number of levels h_T and h_B for the top and bottom subtrees of S , respectively called S_T and S_B . We want to identify their corresponding segments M_T and M_B in A . First, note that $M_T = A[i \dots i + 2^{h_T} - 2]$. So, we can invoke directly $\text{FIND}(x, h_T)$ to route x in S_T . By induction, $rank = bfs \bmod 2^{h_T}$ tells us the number of S_B (starting from 0 and going from left to right in S). Since each bottom subtree has size $2^{h_B} - 1$, we can infer that M_B starts at position $i + (2^{h_B} - 1) \cdot rank + 1$ of A . So we update i to this new value. We can invoke $\text{FIND}(x, h_B)$ to route x in S_B . By induction, we correctly compute bfs in the VEB-tree for the next node below S_B that is also the next node below S . In order to preserve the invariant, we need to update i so that it is the last position of M in A . We have to compute $rank$ again, since we cannot keep this value due to the implicitness. We therefore, compute the breadth-first number of the root of S_B , which is the current value of bfs divided by 2^{h_B+1} (the exponent $h_B + 1$ comes from the fact that bfs refers to one level below the leaves of S_B). We then take the modulo of the resulting breadth-first number in $rankroot$, as we did for $rank$ (indeed lines 9 and 13 compute the same value but at different times in the recursion; so we cannot keep the values of the variables $rank$ in the recursive levels to obtain an in-place algorithm). We finally increment i noting that we have to jump over the keys of $(2^{h_T} - 1 - rankroot)$ bottom subtrees of size $2^{h_B} - 1$. As a result, we keep the invariant for S . Finally, we observe that we traversed a path from the root of S to a leaf of it. The only accessed keys in A are at line 2 updating bfs , so that the next key to be compared with x is either the left child or the right child of $A[i]$ but not both.

The cost of searching is asymptotically upper bounded by the solution to the recurrence $C(2^h - 1) = C(2^{h_T} - 1) + C(2^{h_B} - 1) + O(1)$. The crucial observation is that FIND traverses a downward path from the root to an internal node, whose length is asymptotically bounded by $C(2^h - 1)$. For suitable constant d_1 and d_2 , the solution is $C(2^h - 1) \leq d_1 h - d_2$ by substitution on the right hand side of the recurrence. Hence, a VEB-permutation for $2^h - 1$ keys can be searched in $O(h)$ time and $O(h/\log B)$ block transfers.

4.4 Maintaining the Top Layer and the VEB-Permutation

The association of virtual chunks with actual chunks is dynamically maintained when bucket splitting creates new chunks and bucket merging removes chunks. As long as we can keep at most α virtual chunks associated with each actual chunk, we do not need to redistribute chunks. However, when removing an actual chunk that has no virtual chunk associated with it, or when creating a new chunk from a chunk in a maximal list of size α , we have to redistribute the chunks by reclassifying them as actual and virtual, while preserving their order. We employ a notion of density [1,4,13] for this purpose, with the further requirements of avoiding to produce empty slots and of distributing virtual chunks among the actual chunks without violating their order. Note that we are allowed to pay a search in the VEB-directory for each chunk involved in the redistribution with its two smallest and greatest keys, replacing them when the corresponding chunk is exchanged. We refer the reader to the full version for the several details on the redistribution.

5 In-Place Rebuilding and Final Bounds

In our algorithms we assumed that $n'/4 < n < n'$ at any time, where n' is a power of two. That value of n' is important to fix the parameter k discussed in Section 2. Note that the time complexity of our algorithms is parametric in k and that we fixed $k = \Theta(\log n')$ to get our claimed bounds. To preserve the invariant on $n'/4 < n < n'$ when $n = n'$, we double n' , update the value of k and rebuild by a sequence of $O(n)$ insertions, with the only difference that we fix $k = \Theta(\log n')$ even if we may have re-inserted less than $n'/4$ keys during the rebuilding (this is important to avoid triggering a sequence of recursive rebuilding). Analogously, when $n = n'/4$ we halve n' , update the value of k , and rebuild. In both case, we have $n = n'/2$ for the new value of n' after rebuilding, and so our invariant is maintained with n half on the way between $n'/4$ and n' . The amortized cost of rebuilding is given by the total cost of $O(n')$ insertions divided by the number of insertions and deletions performed in an epoch, which is $\Omega(n')$. As a result, the amortized cost of the rebuilding is the cost of $O(1)$ insertions (here it should be clear why we do not start a nested sequence of rebuilding operations, since we keep $k = \Theta(\log n')$ unchanged for all the rebuilding). Apart from the rebuilding, the cost of search, insert and delete is that stated in Sections 3 and 4. From these costs, it follows our main result, Theorem 1.

References

1. M. A. Bender, E. D. Demaine, and M. Farach-Colton. Cache-oblivious b -trees. In *Proc. 41st Symposium on Foundations of Computer Science*, pages 399–409, 2000.
2. Michael A. Bender, Richard Cole, and Rajeev Raman. Exponential structures for efficient cache-oblivious algorithms. *Lecture Notes in Computer Science*, 2380:195–207, 2002.

3. Michael A. Bender, Ziyang Duan, John Iacono, and Jing Wu. A locality-preserving cache-oblivious dynamic dictionary. In *Proc. 13th Annual Symposium On Discrete Mathematics*, pages 29–38, 2002.
4. Gerth Stølting Brodal, Rolf Fagerberg, and Riko Jacob. Cache-oblivious search trees via trees of small height. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 39–48, 2002.
5. Andrej Brodnik and J. Ian Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28(5):1627–1640, 1999.
6. Amos Fiat, Moni Naor, Jeanette P. Schmidt, and Alan Siegel. Nonoblivious hashing. *Journal of the ACM*, 39(4):764–782, October 1992.
7. Robert W. Floyd. Algorithm 245 (TREESORT). *Communications of the ACM*, 7:701, 1964.
8. Gianni Franceschini and Roberto Grossi. Implicit dictionaries supporting searches and amortized updates in $O(\log n \log \log n)$. In *Proc. 14th Annual Symposium on Discrete Algorithms*, 2003.
9. Gianni Franceschini, Roberto Grossi, J. Ian Munro, and Linda Pagli. Implicit B-trees: New results for the dictionary problem. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.
10. Greg N. Frederickson. Implicit data structures for the dictionary problem. *Journal of the ACM*, 30(1):80–94, 1983.
11. Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984.
12. M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Annual Symposium on Foundations of Computer Science*, pages 285–297, 1999.
13. Alon Itai, Alan G. Konheim, and Michael Rodeh. A sparse table implementation of priority queues. *Proc. Intern. Colloquium on Automata, Languages and Programming*, LNCS 115, pages 417–431, 1981.
14. D. E. Knuth. *The Art of Computer Programming III: Sorting and Searching*. Addison-Wesley, Reading, Massachusetts, 1973.
15. J. Ian Munro. An implicit data structure supporting insertion, deletion, and search in $O(\log^2 n)$ time. *Journal of Computer and System Sciences*, 33(1):66–74, 1986.
16. J. Ian Munro and Hendra Suwanda. Implicit data structures for fast search and update. *Journal of Computer and System Sciences*, 21(2):236–250, 1980.
17. Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal on Computing*, 31(2):353–363, 2002.
18. H. Prokop. Cache-oblivious algorithms. Master’s thesis. MIT, Cambridge, MA, 1999.
19. Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k -ary trees and multisets. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 233–242, 2002.
20. J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7:347–348, 1964.

The Cell Probe Complexity of Succinct Data Structures

Anna Gál¹ and Peter Bro Miltersen²

¹ Dept. of Computer Science, University of Texas at Austin. panni@cs.utexas.edu

² Dept. of Computer Science, University of Aarhus. bromille@brics.dk

Abstract. We show lower bounds in the cell probe model for the redundancy/query time tradeoff of solutions to static data structure problems.

1 Introduction

In the cell probe model (e.g., [1,3,4,6,7,9,18,19,20,21]), a boolean static data structure problem is given by a map $f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}$, where $\{0,1\}^n$ is a set of possible data to be stored, $\{0,1\}^m$ is a set of possible queries and $f(x,y)$ is the answer to question y about data x . For natural problems, we have $m \ll n$: the question we pose to the database is much shorter than the database itself. Examples of natural data structuring problems include:

Substring Search: Given a string x in $\{0,1\}^n$ we want to store it in a data structure so that given a query string y of length m , we can tell whether y is a substring of x by inspecting the data structure. This problem is modeled by the function f defined by $f(x,y) = 1$ iff y is a substring of x .

Prefix Sum: Given a bit vector $x \in \{0,1\}^n$, store it in a data structure so that queries “What is $(\sum_{i=1}^k x_i) \bmod 2$?” can be answered. This problem is modeled by the function f defined by $f(x,y) = (\sum_{i=1}^{v_y} x_i) \bmod 2$ where y is the binary representation of the integer v_y .

For Substring Search, both the data to be stored and the query are bit strings, as our framework requires. The only reason for this requirement is that to make our discussion about current lower bound techniques and their limitations clear, we want the parameter n to always refer to the number of bits of the data to be stored, the parameter m to always refer to the number of bits of a query and the output of the query to be a single bit. In general, we don’t necessarily expect the data we want to store to be bit strings, but an arbitrary encoding as bit strings may take care of this, as in the following example.

Membership: Given a set S of k binary strings each of length m , store S as a data structure so that given a query $y \in \{0,1\}^m$, we can tell whether $y \in S$. To make this problem fit into the framework above, the function f would be defined by letting $n = \lceil \log_2 \binom{2^m}{k} \rceil$ and fixing, in some arbitrary way, a compact encoding of k -sets as n -bit strings and letting $f(S,y) = 1$ iff $y \in S$.

The framework captures not only the classical “storage and retrieval” static data structure problems but also more general problems of dealing with pre-processed information, such as the classical algebraic problem of polynomial evaluation with preprocessing of coefficients ([15, pp. 470-479], see also [19]):

Polynomial Evaluation: Store $g \in \mathbf{F}[x]$, $|\mathbf{F}| = 2^k$, g of degree $\leq d$ as a memory image so that queries “What is $g(x)$?” can be answered for any $x \in \mathbf{F}$. This problem is non-boolean, but can be modeled as a boolean problem by letting $n = (d+1)k$, $m = k + \log k$, fixing an arbitrary compact encoding of polynomials and field elements as bit strings and letting $f(g, x \cdot y) = v_y$ ’th bit of $g(x)$, where y is the binary notation of v_y and $\cdot$ denotes concatenation.

In the cell probe model with word size 1 (the bit probe model), a *solution* with space bound s and time bound t to a problem f is given by a storage scheme $\phi : \{0, 1\}^n \rightarrow \{0, 1\}^s$, and a query algorithm q so that $q(\phi(x), y) = f(x, y)$. The time t of the query algorithm is its bit probe complexity, i.e., the worst case number of bits it reads in $\phi(x)$.

Every problem possesses two trivial solutions: The solution of explicitly storing the answer to every query (this solution has space $s = 2^m$ and time $t = 1$) and the solution of storing the data verbatim and reading the entire data when answering queries (this solution has space $s = n$ and time $t = n$, as we only charge for reading bits in $\phi(x)$, not for computation). The study of cell probe complexity concerns itself with the tradeoff between s and t that may be obtained by solutions somewhere between the two extremes defined by the trivial solutions. Such solutions may be quite non-trivial and depend strongly on the problem considered. A *polynomial* solution satisfies $s = n^{O(1)}$ and $t = m^{O(1)}$. For instance, perfect hashing schemes form solutions to Membership with $s = O(n)$ and $t = O(m)$ [11] and even $s = n + o(n)$ and $t = O(m)$ [5,25]. Substring Search also admits an $s = O(n)$, $t = O(m)$ solution [12] and very recently a solution with $s = n + o(n)$ and $t = m^{O(1)}$ was constructed [13] but no solution with $s = n + o(n)$ and $t = O(m)$ is known. For a problem such as Polynomial Evaluation (and many natural data structure problems, such as partial match type problems [3,4,7]), we know of no solution with $s = n^{O(1)}$, $t = m^{O(1)}$. Thus, a main concern is to prove that such solutions do not exist.

For $s = O(n)$, lower bounds of the form $t = \Omega(m)$ may be obtained for explicit and natural problems by simple counting arguments [6]. For $s = n^{O(1)}$, we can do almost as good: Lower bounds of the form $t = \Omega(m/\log n)$ can be obtained using communication complexity [20]. But no very good (i.e., $\omega(m)$) lower bounds are known on t for *any* explicit problem f for the case of $s = O(n)$ or $s = n^{O(1)}$ even though counting arguments prove the existence of (non-explicit) problems f with lower bounds of the form $t = \Omega(n)$, even for $m \approx (\log n)^2$ [18]. Thus, it is consistent with our current knowledge that solutions with $s = O(n)$ and $t = O(m)$ exist for *all* explicit (e.g., all exponential time computable) problems, though it is certainly a generally believed conjecture that this is not the case!

Given our lack of tools strong enough to show statements such as $s = O(n) \Rightarrow t = \omega(m)$ for explicit problems, it seems appropriate to lower our ambitions

slightly and try to show such lower bounds for t for *any* non-trivial value of s . Achieving such goals is well in line with the current trend in the theoretical as well as practical studies of data structures (e.g., [17,5,25,13]) of focusing on *succinct* data structures where $s = n + r$ for some *redundancy* $r \ll n$, i.e., on structures whose space requirement is close to the information theoretic minimum. Restricting our attention to such succinct structures by no means trivializes obtaining the lower bounds we want to show. For instance, it is open (and remains open, also after this work) whether a solution with $r = 0$ and $t = O(m)$ exists for the Membership problem. However, in this paper we show that for certain explicit (polynomial computable) problems it *is* possible to show lower bounds of the form $t = \omega(m)$ and even $t = \Omega(n)$ for structures with a sufficiently strong upper bound on r :

Theorem 1. *Let k, d be integers larger than 0 so that $d < 2^k/3$. Let $\mathbf{F} = \text{GF}(2^k)$ and let $n = (d + 1)k$. Let a storage scheme $\phi : \{f \mid f \in \mathbf{F}[x], \text{degree}(f) \leq d\} \rightarrow \{0, 1\}^{n+r}$ and associated query scheme for “What is $f(x)$?”, $x \in \mathbf{F}$ with bit probe complexity t be given. Then, $(r + 1)t \geq n/3$.*

In particular, for very small redundancies, we get an almost optimal lower bound stating that the query algorithm has to inspect almost the entire data structure. The theorem is for the (more natural) non-boolean version of the polynomial evaluation problem. A lower bound of $(r + 1)t \geq n/3k$ for the boolean version of polynomial evaluation we defined previously immediately follows.

The proof of Theorem 1 (presented in Section 2) is based on the fact that the problem of polynomial evaluation hides an error correcting code: The strings of query answers for each possible data (i.e., each polynomial) form the Reed-Solomon code. We can generalize Theorem 1 to any problem hiding an error correcting code in a similar way (see Theorems 4 and 5 in Section 2). However, not many natural data structuring problems contain an error correcting code in this way. In Section 2, we introduce a parameter of data structuring problems called *balance* and, using the sunflower lemma of Erdős and Rado show that for problems having constant balance, we get a lower bound of the form $t(r + 1)^2 \geq \Omega(n)$ (Theorem 6). A problem hiding a good error correcting code in the way described above has constant balance, but the converse statement is not necessarily true. Hence Theorem 6 has the potential to prove lower bounds on a wider range of problems than Theorems 4 and 5, though we do not have any natural data structuring problems as examples of this at the moment.

The results above are based on combinatorial properties of a coding theoretic flavor of the problems f to be solved. We don’t know how to prove similar lower bounds for natural storage and retrieval problems such as Substring Search. However, we get a natural restriction of the cell probe model by looking at the case of *systematic* or *index* structures. These are storage schemes ϕ satisfying $\phi(x) = x \cdot \phi^*(x)$ for some map ϕ^* , i.e., we require that the original data is kept “verbatim” in the data structure. We refer to $\phi^*(x)$ as the *index part* of $\phi(x)$. The restriction only makes sense if there is a canonical way to interpret the data to be stored as a bit-string. It is practically motivated: The data to be encoded may be in read-only memory or belong to someone else or it may be necessary

to keep it around for reasons unrelated to answering the queries defined by f . For more discussion, see, e.g. Manber and Wu [17]. In the systematic model, we prove a tight lower bound for Prefix Sum (in fact, we show that the lower bound is implicit in work of Nisan, Rudich and Saks [24]) and a lower bound for Substring Search.

Theorem 2. $\Theta(n/(r+1))$ bit probes are necessary and sufficient for answering queries in a systematic structure for Prefix Sum with r bit redundancy.

Theorem 3. Consider Substring Search with parameters n, m so that $2\log_2 n + 5 \leq m \leq 5\log_2 n$. For any systematic scheme solving it with redundancy r and bit probe complexity t , we have $(r+1)t \geq \frac{1}{800}n/\log n$.

Both proofs are presented in Section 3. We are aware of one paper previous to this one where lower bounds of the form $t = \omega(m)$ were established for succinct, systematic data structures: Demaine and Lopez-Ortiz [8] show such a lower bound for a variation of the Substring Search problem. In their variation, a query does not just return a boolean value but an index of an occurrence of the substring if it does indeed occur in the string. For this variation, they prove the following lower bound for a value of m which is $\Theta(\log n)$ as in our bound: $t = o(m^2/\log m) \Rightarrow (r+1)t = \Omega(n\log n)$. Thus, they give a lower bound on the query time even with *linear* redundancy which our method cannot. On the other hand, their method cannot give lower bounds on the query time better than $\Omega(m^2/\log m)$ even for very small redundancies which our method can. Furthermore, our lower bound applies to the boolean version of the problem.

2 Lower Bounds for Non-systematic Structures

Proof of Theorem 1. Let a storage scheme ϕ with redundancy r and an associated query scheme with bit probe complexity t be given. Let $s = n + r$. Assume to the contrary that the scheme satisfies $(r+1)t < n/3$. As $r \geq 0$ in any valid scheme, we have $t < n/3$. We make a randomized construction of another storage scheme ϕ' by randomly removing $r+1$ bits of the data structures of storage scheme ϕ . That is, we pick $S \subset \{1, \dots, n+r\}$ of size $r+1$ at random and let $\phi'(x) = \phi(x)$ with bits in positions $i \in S$ removed. Thus, $\phi'(x) \in \{0, 1\}^{n-1}$. We make an associated query scheme for ϕ' by simulating the query scheme for ϕ , but whenever a bit has to be read that is no longer there, we immediately answer “Don’t know”. Clearly, if we use our new storage scheme ϕ' and the associated query scheme, we will on every query, either get the right answer or the answer “Don’t know”. Now fix a polynomial f and a query x and let us look at the probability that the randomized construction gives us the answer “Don’t know” on this particular data/query-pair. The probability is equal to the probability that the random set S intersects the fixed set T of bits that are inspected on query x in structure $\phi(f)$ according to the old scheme. As $|S| = r+1$ and $|T| \leq t$, the probability of no intersection can be bounded as $\Pr[S \cap T = \emptyset] \geq \left(\frac{s-t}{s}\right) \left(\frac{s-1-t}{s-1}\right) \dots \left(\frac{s-(r+1)+1-t}{s-(r+1)+1}\right)$

$\geq (1 - \frac{t}{n})^{r+1} \geq 1 - \frac{(r+1)t}{n} > 2/3$. This means that if we fix f and count the number of answers that are not “Don’t know” among all answers to “What is $f(x)$?”, $x \in \mathbf{F}$, the expected number of such valid answers is $> 2|\mathbf{F}|/3$, and the expected number of “Don’t know” answers is $< |\mathbf{F}|/3$. Thus, for fixed f , the probability that the number of valid answers for this f is $< |\mathbf{F}|/3$ is $< 1/2$. Define f to be “good” for a particular choice of S if the number of valid answers for f is at least $|\mathbf{F}|/3$. Thus, for random S , the probability that a particular fixed f is good is $> 1/2$, by the above calculation, so if we count among all 2^n possible f ’s the number of good f ’s, the expectation of this number is $> 2^n/2$. Thus, we can fix a value of S so that the number of good f ’s is $> 2^n/2$. Let the set of good f ’s relative to this choice of S be called G . We now argue that the map $\phi' : G \rightarrow \{0, 1\}^{n-1}$ is a 1-1 map: Given the value $\phi'(f)$ for a particular $f \in G$, we can run the query algorithm for $f(x)$ for all $x \in \mathbf{F}$ and retrieve a valid answer in at least $|\mathbf{F}|/3$ cases - in the other cases we get the answer “Don’t know”. Since the degree of f is less than $|\mathbf{F}|/3$, the information we retrieve is sufficient to reconstruct f . Thus, we have constructed a 1-1 map from G with $|G| > 2^n/2$ to the set $\{0, 1\}^{n-1}$ which has size 2^{n-1} . This violates the pigeonhole principle, and we conclude that our assumption $(r+1)t < n/3$ was in fact wrong. This completes the proof of Theorem 1.

Theorem 1 can be generalized to any problem based on some error correcting code. Consider an arbitrary boolean static data structure problem, given by a map $f : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$. Let $N = 2^n$, and $M = 2^m$. Then the problem can be represented by an $N \times M$ Boolean matrix A_f , with the entry at the row indexed by x and the column indexed by y being equal to $f(x, y)$.

Theorem 4. *Let A_f be the N by M ($N = 2^n$) matrix of a data structure problem such that the rows of A_f have pairwise distance at least δM . If the problem can be solved with redundancy r and query time t , then $t(r+1) \geq \delta n/2$.*

The argument can also be extended to problems where the minimum distance may not be large, but instead we require that within any ball of radius ρM there are at most L codewords (i.e., codes with certain *list decoding* properties). In fact, the even weaker property of having only few codewords in every *subcube* of dimension ρM is sufficient for our purposes. (Note that this property corresponds to the problem of list decoding from erasures, rather than from errors.)

Let $\alpha_{i_1}, \dots, \alpha_{i_{M-d}}$ be an arbitrary 0/1 assignment to $M - d$ coordinates. The set $S \subseteq \{0, 1\}^M$ of size $|S| = 2^d$ formed by all possible vectors from $\{0, 1\}^M$ agreeing with $\alpha_{i_1}, \dots, \alpha_{i_{M-d}}$ and arbitrary in the remaining coordinates is called a *subcube of dimension d* .

Theorem 5. *Let A_f be the N by M ($N = 2^n$) matrix of a data structure problem such that within any subcube of dimension ρM there are at most L row vectors from A_f . If the problem can be solved with redundancy r and query time t , then $t(r+1 + \log L) \geq \rho(n - \log L)/2$.*

The proofs of Theorems 4 and 5 are very similar to the proof of Theorem 1 and appear in the full version of this paper.

We next give a general lower bound for any problem whose matrix satisfies certain conditions. Informally, we require that the submatrix formed by any small subset of rows contains a balanced column.

Definition 1. *Let A be a matrix with 0/1 entries. We say that A has balance at least λ for parameter k , if for any k rows of the matrix A there exists a column that contains at least λk 0-s and at least λk 1-s among the entries of the given k rows.*

Lemma 1. *Given a code with N words in $\{0,1\}^l$, let A be the N by l matrix formed by the words as rows. If the minimum distance of the code is δl , then A has balance at least $\delta/8$ for every $1 < k \leq N$.*

Proof. Look at the k by l table formed by k rows of A . Let $\gamma = \delta/8$. Suppose that each column in the table has either $< \gamma k$ 0-s or $< \gamma k$ 1-s. Let a be the number of mostly 1 columns and b be the number of mostly 0 columns. Then $< k/2$ rows have $> 2\gamma a$ 0-s on the mostly 1 part. Restrict the table to the other $k' > k/2$ rows. In this table, the b mostly 0 columns still have $< 2\gamma k'$ 1-s. So, $< k'/2$ rows have $> 4\gamma b$ 1-s on the mostly 0 part. Thus, $> k/4$ rows have both $< 2\gamma a$ 0-s on the mostly 1 part and $< 4\gamma b$ 1's on the mostly 0 part, respectively. The distance of any two of these rows is $< 4\gamma a + 8\gamma b < \delta l$, which is a contradiction.

The proof of Lemma 1 also extends to codes where the minimum distance may not be large, but instead we require that within any ball of certain radius there are not too many words, i.e., to problems satisfying the condition of Theorem 5. We can, however, construct codes that satisfy the property of having large balance for every k , *without* the property of having few codewords in every Hamming ball of a given radius, and even without the weaker property of having few codewords in every subcube of a given dimension. Consider the following example of such construction. Let ρ be any constant, and L any integer, such that $\rho + \frac{1}{L} < 1/20$. We will construct a set of words in $\{0,1\}^M$ with at least L words in some subcube of dimension ρM , such that for any set of rows of the corresponding matrix there is a column with balance $> \rho + \frac{1}{L}$. Start with any family that has balance at least $5(\rho + \frac{1}{L})$. (We know the existence of such families, from the existence of good error correcting codes.) Add L words to this family as follows. Take a code of L words on $c \log L$ coordinates for some constant c , with relative minimum distance $1/4$. (Such code exists for some constant c .) Let the first $c \log L$ coordinates of the extra L words to be words from this code of size L , and let the L words be identical in the remaining $M - c \log L$ coordinates. Unless L is huge (compared to M), we have $c \log L < \rho M$, thus we have L words in a subcube of dimension ρM . It is not hard to see that the corresponding matrix has balance at least $\rho + \frac{1}{L}$ for any k . Thus, the following theorem has the potential of giving lower bounds for a wider range of problems than the theorems of Section 2. Consider an arbitrary boolean static data structure problem, given by a map $f : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}$.

Theorem 6. *Let A_f be the N by M ($N = 2^n$, $M = 2^m$) matrix of f . If A_f has balance at least λ for every $1 < k \leq \log N$. and the problem defined by f can be solved with redundancy r and query time t , then $t(r+1)^2 \geq \lambda n$.*

Proof. A solution to the data structure problem is given by a representation $\phi : \{0, 1\}^n \rightarrow \{0, 1\}^s$ and a query algorithm. We consider a matrix B of size $N \times s$, such that the row of B indexed by x is the vector $\phi(x)$. We use the following standard observation.

Observation 1. *Given a set $\mathcal{C}$ of $N = 2^{s-r}$ vectors in $\{0, 1\}^s$, for every $0 \leq w \leq s$ there is a vector $v \in \{0, 1\}^s$, such that there are at least $\binom{s}{w}/2^r$ vectors in $\mathcal{C}$ at distance w from v .*

Proof. Let $\chi(u, v) = 1$ if u and v differ in w coordinates, and $\chi(u, v) = 0$ otherwise. We have $\sum_{u \in \mathcal{C}} \sum_{v \in \{0, 1\}^s} \chi(u, v) = |\mathcal{C}| \binom{s}{w}$. On the other hand, $\sum_{v \in \{0, 1\}^s} \sum_{u \in \mathcal{C}} \chi(u, v) \leq 2^s \max_{v \in \{0, 1\}^s} |\mathcal{C}_{v, w}|$, where $\mathcal{C}_{v, w} = \{z \in \mathcal{C} \mid z \text{ and } v \text{ differ in } w \text{ coordinates}\}$. This completes the proof of Observation 1.

Let $w = r + 1$ (note that $r + 1 \geq 1$), and let $v \in \{0, 1\}^s$, guaranteed to exist by the observation, such that there are at least $\binom{s}{r+1}/2^r$ rows of B at distance $r + 1$ from v . Let B_v be the matrix obtained from B by adding v to each row of B (taking bitwise XOR). With each vector $u \in \{0, 1\}^s$ we associate a set $U \subseteq [s]$, such that $i \in [s]$ belongs to U if and only if the i -th entry of u is 1. Then the matrix B_v specifies a family $\mathcal{B}$ of N sets, such that at least $\binom{s}{r+1}/2^r$ members of $\mathcal{B}$ have cardinality $r + 1$.

A family of k sets $S_1, \dots, S_k$ is called a *sunflower* with k petals and core T , if $S_i \cap S_j = T$ for all $i \neq j$. We also require that the sets $S_i \setminus T$ are nonempty.

Lemma 2 (Erdős and Rado, [10]). *Let $\mathcal{F}$ be a family of sets each with cardinality w . If $|\mathcal{F}| > w!(k-1)^w$, then $\mathcal{F}$ contains a sunflower with k petals.*

Since $\binom{s}{r+1}/2^r > (r+1)!(s/(r+1)^2)^{r+1}$, Lemma 2 implies that $\mathcal{B}$ contains a sunflower with $k = s/(r+1)^2$ petals. Let $S_1, \dots, S_k$ be the sets of the sunflower, and let T be its core. Then, the sets $S_i \Delta T$ are pairwise disjoint. ($S_i \Delta T$ denotes the symmetric difference of the sets S_i and T .) Let z and $u_1, \dots, u_k$ be the vectors obtained by adding the vector v to the characteristic vectors of the set T and $S_1, \dots, S_k$, respectively. Then the vectors $u_1, \dots, u_k$ are rows of the matrix B , and they have the property that the vectors $z \oplus u_1, \dots, z \oplus u_k$ have no common 1's, since the set $S_i \Delta T$ is exactly the set of coordinates where the vectors z and u_i differ from each other. Let $x_1, \dots, x_k$ be the data such that $u_i = \phi(x_i)$, $i = 1, \dots, k$. Consider now the k rows of A_f indexed by $x_1, \dots, x_k$. By our assumption on A_f , there is a question y , such that at least λk of the answers $f(x_i, y)$ are 0, and at least λk of the answers $f(x_i, y)$ are 1. We think of the query algorithm as a decision tree, and show that it has large depth. In particular, we show that the path consistent with the vector z has to be at least λk long. (Note that the vector z may not be a row of the matrix B . However, we can assume that the decision tree has been trimmed, so that there are no long paths that can be cut off without affecting the correctness of the algorithm. This

implies that there is at least one path corresponding to a vector $\phi(x)$ that the algorithm may actually have to follow, and is at least λk long.) Assume that the query algorithm reads at most $t < \lambda k$ bits on any input when trying to answer the question y , and assume that the bits read are consistent with the vector z . Since the sets of coordinates where z differs from u_i for $i = 1, \dots, k$ are pairwise disjoint, after asking at most t questions, the algorithm can rule out at most t of the data $x_1, \dots, x_k$, and the remaining $k - t$ are still possible. If $t < \lambda k$, then among the data that are still not ruled out, both the answer 0 and the answer 1 is possible, and the algorithm cannot determine the answer to the given question y . This completes the proof of Theorem 6.

It is not hard to find examples of matrices with large balance for $k \leq \log N$, if we are not worried about the number of rows N being large enough compared to the number of columns M . We should mention that there are well known constructions (e.g. [2,14,22,23,26]) for the much stronger property requiring that all possible 2^k patterns appear in the submatrix formed by arbitrary k rows. However, in such examples, $N \leq M$ or $2^k \leq M$ must trivially hold. Error correcting codes provide examples where N can be very large compared to M . Let $n(k, \lambda, M)$ denote the largest possible number n , such that 2^n by M 0/1 matrices exist with balance at least λ for k . Lower bounds on the largest achievable rate of error-correcting codes or list decodable codes provide lower bounds on $n(k, \lambda, M)$. For example, the Gilbert-Varshamov bound (see e.g. [16]) together with Lemma 1 implies $n(k, \lambda, M) \geq (1 - H(8\lambda))M$, for every $k > 1$. Note that while error correcting codes give large balance for every $k > 1$, for our purposes matrices that have large balance for only certain values of k may already be useful. It would be interesting to know if $n(k, \lambda, M)$ can be significantly larger (for certain values of k) than what is achievable by error-correcting or list decodable codes. If this is the case, then our techniques might help to achieve lower bounds for the Membership problem.

3 Lower Bounds for Systematic Structures

Proof of Theorem 2. Upper bound: For $r = 0$, the upper bound is obvious. For $r \geq 1$, divide the input vector into r equal sized blocks and let y_i be the parity of the i 'th block. Now store for each $j = 1, \dots, r$, the parity of $y_1, y_2, \dots, y_j$. Given a prefix sum query, it can be answered by reading a non-systematic bit, that gives the parity of a collection of blocks and XORing it with a number of individual input bits, all found in a single block of size n/r . The bit probe complexity is $O(n/r)$.

Lower bound: Let a scheme of redundancy r be given and suppose the queries can be answered with t bit probes, i.e., we can find $x_1 \oplus \dots \oplus x_j$ using a decision tree of depth t over the input bits and the index bits. Split the input into $r + 1$ blocks of about equal length, each block containing at least $\lfloor \frac{n}{r+1} \rfloor$ bits. It is possible to determine the parity of one of the blocks by a decision tree of depth $2t$ over the input bits and the index bits. We now apply a theorem of Nisan,

Rudich and Saks [24]: Given $l + 1$ instances of computing parity of k bits, with l *help bits* (which can be arbitrary functions of the $(l + 1)k$ input bits), given for free. At least one of the $l + 1$ parity functions has decision tree complexity $\geq k$. We immediately get the desired bound.

Proof of Theorem 3. Since we must have $r \geq 0$ and $t \geq 1$ in a valid scheme, we can assume that $1 \leq t \leq \frac{n}{800 \log n}$ otherwise there is nothing to prove. We need to prove a claim about a certain two-player game. Let $b \geq a \geq 40$ be integers and assume b is even. The game is played with b boxes labeled $0, \dots, b - 1$ and a slips of papers, labeled $0, \dots, a - 1$. Player I colors each slip of paper either red or blue and puts each slip of paper in a box (with no two slips going into one box) without Player II watching. Now Player II can open at most $b/2$ boxes using any adaptive strategy and based on this must make a guess about the color of every slip of paper. Player II wins the game if he correctly announces the color of every slip of paper. Suppose Player I adopts the strategy of coloring each slip of paper uniformly and independently at random and putting them at random into a boxes chosen uniformly at random. We claim that no matter which strategy Player II adopts, the probability that Player II wins the game is at most $2^{-a/20}$. To prove the claim, note that when Player I is playing uniformly at random in the way described, by symmetry the adaptiveness of Player II is useless and the optimal strategy for Player II is to open boxes $1, 2, \dots, b/2$, announce the colors of the slips of papers found and make an arbitrary guess for the rest. The probability that he finds more than $\frac{9}{10}a$ slips of papers is $\sum_{j > \frac{9}{10}a} \frac{\binom{b/2}{j} \binom{b/2}{a-j}}{\binom{b}{a}} = \sum_{i=0}^{\lfloor \frac{1}{10}a \rfloor} \frac{\binom{b/2}{i} \binom{b/2}{a-i}}{\binom{b}{a}}$. Since $a \leq b$, for $i \leq \frac{1}{10}a$ we have $\frac{b}{2(b-i)} \leq 5/9$. Then, $\frac{\binom{b/2}{i} \binom{b/2}{a-i}}{\binom{b}{a}} \leq \binom{a}{i} (1/2)^i \left(\frac{b}{2(b-i)}\right)^{a-i} \leq \binom{a}{i} (5/9)^a$ and $\sum_{i=0}^{\lfloor \frac{1}{10}a \rfloor} \frac{\binom{b/2}{i} \binom{b/2}{a-i}}{\binom{b}{a}} \leq (5/9)^a \sum_{i=0}^{\lfloor \frac{1}{10}a \rfloor} \binom{a}{i} \leq (5/9)^a 2^{H(1/10)a} \leq 2^{(H(1/10) - \log_2(3/2))a} \leq 2^{-0.115a}$. The probability that he guesses the colors of all remaining slips correct, given that at least $a/10$ was not found is at most $2^{-a/10}$. Thus, the probability that Player II correctly guesses the color of every slip of paper is bounded by $2^{-0.115a} + 2^{-a/10} \leq 2^{-a/20}$, as $a \geq 40$. This completes the proof of the claim.

We show that a good scheme for Substring Search leads to a good strategy for Player II in the game. So given a scheme with parameters n, m, r, t , we let $a = \lfloor \frac{n}{4tm} \rfloor$ and $b = 4ta$. Since $t \leq n/(800 \log n)$ and $m \leq 5 \log n$, we have $a \geq 40$. We consider a string of length n as consisting of b concatenated chunks of length m , padded with 0's to make the total length n (note that $bm = 4tam \leq n$). We can now let such a string encode a move of Player I (i.e. a coloring of slips of papers and a distribution of them into boxes) as follows: The content of Box i is encoded in chunk number i . If the box is empty, we make the chunk 000000..000. If the box contains paper slip number j , colored blue, we make the chunk 001 j_1 1 j_2 1 j_3 1...1 j_k 0, padded with zeros to make the total length m , where $j_1 \dots j_k$ is the binary representation of j with $\lceil \log a \rceil$ binary digits (note that $3 + 2\lceil \log a \rceil \leq 2 \log n + 5 \leq m$). Similarly, if the box contains paper slip number j , colored red, we make the chunk 001 j_1 1 j_2 1 j_3 1...1 j_k 1, padded with zeros. Now

consider the set X of strings encoding all legal moves of player I. Each element x of X has some systematic data structure $\phi(x) = x \cdot \phi^*(x)$ where $\phi^*(x) \in \{0, 1\}^r$. Pick the most likely setting z of $\phi^*(x)$ of these among elements of X , i.e., if we take a random element x of X , the probability that $\phi^*(x) = z$ is at least 2^{-r} . We now make a strategy for Player II in the game. Player II will pretend to have access to a Substring Search data structure which he will hope encodes the move of Player I. The index part of this data structure will be the string z which is fixed and independent of the move of Player I and hence can be hardwired into the protocol of Player II. Player II shall simulate certain query operations on the pretend data structure. However, he has only access to the index part of the structure (i.e., z). Thus, whenever he needs to read a bit of the non-index bits, he shall open the box corresponding to the chunk of the bit from which he can deduce the bit (assuming that the entire data structure really does encode the move of Player I). In this way, Player II simulates performing query operations “Is $001j_11j_21j_31\dots1j_k0$ a substring?” and “Is $001j_11j_21j_31\dots1j_k1$ a substring?” with $j = j_1j_2\dots j_k$ being the binary representations of all $y \in \{0, \dots, a-1\}$, i.e., $2a$ query operations. From the answers to the queries, he gets a coloring of the slips of papers. All answers are correct for those cases where his index part was the correct one, i.e., for those case where $z = \phi^*(x)$ where x is an encoding of the move of Player I, i.e., with probability at least 2^{-r} . Thus, since the total number of boxes opened is at most $t2a \leq b/2$, we have by the claim that $r \geq a/20$, i.e., $20r \geq \lfloor n/4tm \rfloor$, and, since r is an integer and $m \leq 5 \log n$ we have $(r+1)t \geq \frac{1}{400}n/\log n$. This completes the proof of Theorem 3.

We could potentially get a better lower bound by considering a more complicated game taking into account the fact that the different query operations do not communicate. Again we have b boxes labeled $0, \dots, b-1$ and a slips of paper, labeled $0, \dots, a-1$. The modified game is played between Player I and a *team* consisting of Player $\Pi_0, \Pi_1, \dots, \Pi_{a-1}$. Again, Player I colors each slip of paper either red or blue and puts each slip of paper in a box without Players $\Pi_0, \Pi_1, \dots, \Pi_{a-1}$ watching. Now Player Π_i can look in at most $b/2$ boxes using any adaptive strategy and based on this must make a guess about the color of the slip labeled i . This is done by each player on the team individually without communication or observation between them. The team wins if *every* player in the team correctly announces the color of “his” slip. About this game we can state the following hypothesis.

Hypothesis: Let $b \geq 2a$. Suppose Player I adopts the strategy of coloring each slip of paper uniformly at random and independently putting them at random into a boxes chosen uniformly at random. Then no matter which strategy the team adopts, the probability that they win is at most $2^{-\Omega(a)}$.

The intuition for the validity of the hypothesis is the fact that the players of the team are unable to communicate and each will find his own slip of paper with probability $\leq \frac{1}{2}$. If the hypothesis can be verified it will lead to a tradeoff for Substring Search of the form $t = o(n/\log n) \Rightarrow s = \Omega(n/\log n)$. However, Sven Skyum (personal communication) has pointed out that if the hypothesis is true, the parameters under which it is true are somewhat fragile: If $b = a$,

the team can win the game with probability bounded from below by a constant (roughly 0.3) for arbitrary large values of a . The catch is that even though each player will find his own slip of paper with probability only $\frac{1}{2}$, one can make these events highly dependent (despite the fact that the players do not communicate). We leave finding Skyum's protocol as an exercise to the reader.

4 Open Problems

It is interesting that all our best bounds, both in the non-systematic and in the systematic case, are of the form “ $(r+1)t$ must be linear or almost linear in n .” We don't see any inherent reason for this and in general do not expect the lower bounds obtained to be tight. Thus, it would be nice to prove a lower bound of, say, the form, $t < n/\text{polylog } n \Rightarrow r > n/\text{polylog } n$ for Polynomial Evaluation in the non-systematic case or Substring Search in the systematic case. For the latter result, it would be sufficient to verify the hypothesis about the game defined above. It is also interesting to note that our lower bound for Substring Search and the lower bound of Demaine and Lopez-Ortiz are incomparable. Can the two techniques be combined to yield a better lower bound? We have only been able to prove lower bounds in the non-systematic case for problems satisfying certain coding theoretic properties. It would be very nice to extend the non-systematic lower bounds to more natural search and retrieval problems, such as Substring Search. A prime example of a problem for which we would like better bounds is Membership as defined in the introduction. As the data to be stored has no canonical representation as a bitstring, it only makes sense to consider this problem in the non-systematic model. The lower bound $r = O(n) \Rightarrow t = \Omega(m)$ was shown by Buhrman et al [6]. On the other hand, a variety of low-redundancy dictionaries with $r = o(n)$ and $t = O(m)$ has been constructed [5, 25]. We conjecture that any solution for membership with $t = O(m)$ must have *some* redundancy, i.e., that $t = O(m) \Rightarrow r \geq 1$. It would be very nice to establish this. The main open problem of cell probe complexity remains: Show, for some explicit problem, a tradeoff of the form $r = O(n) \Rightarrow t = \omega(m)$. Clearly, for such tradeoffs the distinction between systematic and non-systematic structures is inconsequential.

Acknowledgements. Anna Gál is supported in part by NSF CAREER Award CCR-9874862 and an Alfred P. Sloan Research Fellowship. Peter Bro Miltersen is supported by BRICS, Basic Research in Computer Science, a centre of the Danish National Research Foundation.

References

1. M. Ajtai. A lower bound for finding predecessors in Yao's cell probe model. *Combinatorica*, 8:235–247, 1988.
2. N. Alon, O. Goldreich, J. Håstad, R. Peralta: Simple constructions of almost k -wise independent random variables. *Random Structures and Algorithms* 3 (1992), 289–304.

3. O. Barkol and Y. Rabani, Tighter bounds for nearest neighbor search and related problems in the cell probe model. In *Proc. 32th Annual ACM Symposium on Theory of Computing (STOC'00)*, pages 388–396.
4. A. Borodin, R. Ostrovsky, Y. Rabani, Lower bounds for high dimensional nearest neighbor search and related problems. In *Proc. 31th Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 312–321.
5. A. Brodnik and J.I. Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28:1627–1640, 1999.
6. H. Buhrman, P.B. Miltersen, J. Radhakrishnan, S. Venkatesh. Are bitvectors optimal? In *Proc. 32th Annual ACM Symposium on Theory of Computing (STOC'00)*, pages 449–458.
7. A. Chakrabarti, B. Chazelle, B. Gum, and A. Lvov. A lower bound on the complexity of approximate nearest-neighbor searching on the Hamming Cube. In *Proc. 31th Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 305–311.
8. E.D. Demaine and A. Lopez-Ortiz. A Linear Lower Bound on Index Size for Text Retrieval. In *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 289–294.
9. P. Elias and R. A. Flower. The complexity of some simple retrieval problems. *Journal of the Association for Computing Machinery*, 22:367–379, 1975.
10. P. Erdős and R. Rado. Intersection theorems for systems of sets. *Journal of London Mathematical Society* 35 (1960), pages 85–90.
11. M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the Association for Computing Machinery*, 31:538–544, 1984.
12. R. Grossi, J.S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proc. 32th Annual ACM Symp. on Theory of Computing (STOC'00)*, pages 397–406.
13. R. Grossi, A. Gupta, and J.S. Vitter. High-Order Entropy-Compressed Text Indexes. In *Proc. 14th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'03)*, pages 841–850.
14. D. J. Kleitman and J. Spencer: Families of k -independent sets. *Discrete Math.* 6 (1973), pp. 255–262.
15. D.E. Knuth, *The Art of Computer Programming, Vol. II: Seminumerical Algorithms* (Addison-Wesley, Reading, MA, 2nd ed., 1980).
16. F. J. MacWilliams and N. J. A. Sloane. The theory of error correcting codes. Elsevier/North-Holland, Amsterdam, 1981.
17. U. Manber, S. Wu. GLIMPSE – A Tool to Search Through Entire Filesystems. White Paper. Available at <http://glimpse.cs.arizona.edu/>.
18. P.B. Miltersen. The bitprobe complexity measure revisited. In *10th Annual Symposium on Theoretical Aspects of Computer Science (STACS'93)*, pages 662–671, 1993.
19. P.B. Miltersen, On the cell probe complexity of polynomial evaluation, *Theoretical Computer Science*, 143:167–174, 1995.
20. P.B. Miltersen, N. Nisan, S. Safra, and A. Wigderson: On data structures and asymmetric communication complexity, *Journal of Computer and System Sciences*, 57:37–49, 1998.
21. M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, Mass., 1969.
22. J. Naor and M. Naor: Small-bias probability spaces: efficient constructions and applications. *SIAM J. Comput.*, Vol. 22, No. 4, (1993), pp. 838–856.
23. M. Naor, L. Schulman, A. Srinivasan: Splitters and near optimal derandomization. In *Proc. of 36th IEEE FOCS*, (1995), pp. 182–191.

24. N. Nisan, S. Rudich, and M. Saks. Products and Help Bits in Decision Trees, *SIAM J. Comput.* 28:1035–1050, 1999.
25. R. Pagh. Low redundancy in static dictionaries with $O(1)$ lookup time. In *International Colloquium on Automata Languages and Programming (ICALP'99)*, *Lecture Notes in Computer Science*, Volume 1644, pages 595–604, 1999.
26. G. Seroussi and N. Bshouty: Vector sets for exhaustive testing of logic circuits. *IEEE Trans. Inform. Theory*, 34 (1988), pp. 513–522.

Succinct Representations of Permutations^{*}

J. Ian Munro¹, Rajeev Raman², Venkatesh Raman³, and Satti Srinivasa Rao¹

¹ School of Computer Science, Univ. of Waterloo, Waterloo ON, Canada N2L 3G1.
{[imunro](mailto:imunro@uwaterloo.ca),[ssrao](mailto:ssrao@uwaterloo.ca)}@uwaterloo.ca.

² Department of CS, Univ. of Leicester, Leicester LE1 7RH, UK. rr29@le.ac.uk.

³ Institute of Mathematical Sciences, Chennai, India 600 113. vraman@imsc.res.in

Abstract. We investigate the problem of succinctly representing an arbitrary permutation, π , on $\{0, \dots, n-1\}$ so that $\pi^k(i)$ can be computed quickly for any i and any (positive or negative integer) power k . A representation taking $(1 + \epsilon)n \lg n + O(1)$ bits suffices to compute arbitrary powers in constant time. A representation taking the optimal $\lceil \lg n! \rceil + o(n)$ bits can be used to compute arbitrary powers in $O(\lg n / \lg \lg n)$ time, or indeed in a minimal $O(\lg n)$ bit probes.

1 Introduction

We consider the problem of representing permutations (abbreviated hereafter as *perms* [7]) of $[n] = \{0, \dots, n-1\}$. Perms are fundamental in computer science and have been the focus of extensive study. A number of papers have dealt with issues pertaining to perm generation, membership in perm groups etc. Our aim here is to develop a “perm data structure” that is, we are given a specific and arbitrary (static) perm that arises in some application, and have to represent this perm so that operations on it can be performed rapidly. Initially motivated by being able to compute π or π^{-1} quickly, we consider the more general operation of computing $\pi^k(i)$ for any integer k , where $\pi^0(i) = i$ for all i ; $\pi^k(i) = \pi(\pi^{k-1}(i))$ when $k > 0$ and $\pi^k(i) = \pi^{-1}(\pi^{k+1}(i))$ when $k < 0$.

Certainly, for static perms the above problem is trivial if space is not an issue. Our interest here is in *succinct* or very space-efficient representations that approach the information-theoretic lower bound of $\mathcal{P}(n) = \lceil \lg n! \rceil^1$. Given a perm π in its most natural representation, i.e. the sequence $\pi(i)$, for $i = 0, \dots, n-1$, $\pi^k(i)$ is easily computable in $k-1$ steps. Indeed, for this representation, a $\Theta(n)$ lower bound follows for computing $\pi^k(i)$ when k is large and i is on a large cycle. To facilitate the computation in constant time, one could store $\pi^k(i)$ for all i and k ($|k| \leq n$, along with its cycle length), but that would require $\Theta(n^2 \lg n)$ bits. The most natural compromise is to retain $\pi^k(i)$ with $|k| \leq n$ a power of 2. This $n(\lg n)^2$ 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 $\Theta(\lg n)$ algorithm. Our main result removes this logarithmic

^{*} Work supported in part by UISTRF project 2001.04/IT

¹ $\lg$ denotes logarithm to the base 2

factor from both the time and the space terms, giving $\pi^k(i)$ in constant time and essentially minimum space. To be more specific, we demonstrate:

1. a representation of a perm π that takes $(1+\epsilon)n \lg n + O(1)$ bits of space, and supports $\pi()$ in $O(1)$ time and $\pi^k()$, for any k , in $O(1/\epsilon)$ time, for any $\epsilon > 0$. We also show a restricted lower bound matching this time-space trade-off.
2. a second representation of a perm π that takes $\mathcal{P}(n) + o(n)$ bits of space, and supports $\pi^k()$ for any k in $O(\lg n / \lg \lg n)$ time.

Along the way, we show that answering $\pi()$ and $\pi^{-1}()$ queries suffices to compute queries of arbitrary perm powers. In addition, we can use $\pi()$ and $\pi^{-1}()$ operations to space-efficiently represent a perm of any set S of n elements, i.e. a bijection from S to itself. This is done by combining the results above with an *indexable dictionary* representation of S [15], which essentially implements an (efficiently invertible) bijection from S to $[n]$. This idea easily extends to space-efficient representations of bijections between distinct sets S and T .

One sub-routine we develop here is a representation of a sequence of n integers from $[r]$, for some integer $r \geq 1$, that takes $n \lg r + o(n)$ bits and allows the i -th integer to be accessed in $O(1)$ time. Note that this is $\Theta(n)$ bits better than the naive bound of $n \lceil \lg r \rceil$ bits. This immediately implies an improvement of a similar magnitude for storing satellite information in Pagh's dictionary [14].

There are a number of motivations for succinct data structures in general, many to do with text indexing or representing huge graphs [5,6,12,15]. Indeed, there has already been work on space-efficient representation of restricted classes of perms, such as the perms representing the lexicographic order of the suffixes of a string [5] or so-called approximately min-wise independent perms, used for document similarity estimation [2]. Work on succinct representation of a perm and its inverse was, for one of the authors, originally motivated by a data warehousing application. Under the indexing scheme in the system, the perm corresponding to the rows of a relation sorted under any given key was explicitly stored. It was realized that to perform certain joins, the inverse of a segment of this perm was precisely what was required. The perms in question occupied a substantial portion of the several hundred gigabytes in the indexing structure and doubling this space requirement (for the perm inverses) for the sole purpose of improving the time to compute certain joins was inappropriate. Other applications arise in Bioinformatics [1]. The more general problem of quickly computing $\pi^k()$ also has number of applications. An interesting one is determining the r^{th} root of a perm [13]. Our techniques not only solve the r^{th} power problem immediately, but can also be used to find the r^{th} root, if one exists.

The remainder of the paper is organized as follows. The next section describes some previous results on indexable dictionaries used in later sections, as well as the representation of a sequence of n integers from $[r]$, for some integer $r \geq 1$. Section 3 describes the 'shortcut' method and a matching lower bound (item (1) above) and Section 4 describes item (2) above. We assume a standard *word RAM* model with word size $\Theta(\lg n)$ bits for most of our results. Some results are in the *bit probe* model, where we only count the number of bits in the data structure that are read by a query [11].

2 Preliminaries

Given a set $S \subseteq [m]$, $|S| = n$, a *fully indexable dictionary (FID)* representation for S supports the operations below in $O(1)$ time:

rank(x, S): Given $x \in [m]$, return -1 if $x \notin S$ and $|\{y \in S | y < x\}|$ otherwise;
select(i, S): Given $i \in [n]$, return the $i + 1$ -st smallest element in S .

In addition, a FID for S also supports the operations **rank**($x, \bar{S}$) and **select**($i, \bar{S}$) in constant time, where $\bar{S}$ is the complement of S . This implies that a FID can support the operation **fullrank**(x, S), which returns $|\{y \in S | y < x\}|$ for *all* $x \in [m]$, in constant time as well. Using the characteristic vector of S , and an auxiliary $o(m)$ bit structure to support **rank** and **select** operations in a bit vector [6,12], it is known that:

Theorem 1. *Given a set $S \subseteq [m]$, there is a FID on S that uses $m + o(m)$ bits.*

Raman, Raman and Rao [15] show the following:

Theorem 2 (Lemma 4.1 of [15]). *There is a FID for a set $S \subseteq [m]$ of size n using at most $\lceil \lg \binom{m}{n} \rceil + O(m \lg \lg m / \lg m)$ bits.*

Representing Numbers. We now show how to represent n numbers $a_1, \dots, a_n$ from $[r]$ in $n \lg r + o(n)$ bits, so that we can access the i -th number $O(1)$ time. (Recall that a straightforward representation takes $n \lceil \lg r \rceil$ bits.)

First assume that $r \leq \lg n$. For some $z \geq 1$, we partition the input numbers into contiguous subsequences of z input numbers. We view each subsequence as an integer from $[r^z]$ and represent it using at most $\lceil z \lg r \rceil \leq z \lg r + 1$ bits. We choose z as large as possible so that $z \lg r \leq \frac{1}{2} \lg n$; this allows an individual number in a subsequence to be accessed in $O(1)$ time by looking up a pre-computed table of size at most $z \cdot \lceil \lg r \rceil \cdot 2^{z \lg r + 1} = O(\sqrt{n} \lg n)$ bits. The space used is $(n/z)(z \lg r + 1) + O(\sqrt{n} \lg n) = n \lg r + O(n \lg \lg n / \lg n)$ bits, since $z = \Omega(\lg n / \lg \lg n)$. Now assume that $r > \lg n$ and let $l \geq 1$ be the smallest integer such that $k = \lfloor r/2^l \rfloor \leq \lg n - 1$. We store the sequence $\{a_i \bmod 2^l\}$ using nl bits in the obvious way. As the values $a_i \text{ div } 2^l$ are from $[k + 1]$, where $k + 1 \leq \lg n$, we can store the sequence $\{a_i \text{ div } 2^l\}$ using $n \lg(k + 1) + O(n \lg \lg n / \lg n)$ bits using the above method. Given i , we can easily reconstruct a_i from its “div” and “mod” values in $O(1)$ time. The space used is $n(l + \lg(k + 1)) + O(n \lg \lg n / \lg n)$ bits. Since $(k + 1)2^l > r \geq k2^l$, we have $\lg(k + 1) + l > \lg r \geq \lg k + l$. However, $\lg(k + 1) = \lg k + O(1/k)$, so $n(l + \lg(k + 1)) \leq n \lg r + O(n/k)$. Since $k = \Theta(\lg n)$, the space used in this case is also $n \lg r + O(n \lg \lg n / \lg n)$ bits.

Theorem 3. *A sequence of n numbers from $[r]$ can be represented using $n \lg r + o(n)$ bits so that we can access the i -th element of the sequence in $O(1)$ time.*

From the theorem, we get a representation for an arbitrary permutation on $[n]$ taking $n \lg n + o(n)$ bits supporting $\pi()$ in constant time. To the best of our knowledge, this is the first such representation taking less than $n \lceil \lg n \rceil$ bits.

3 Near-Optimal Representations

3.1 The Shortcut Method

First we design a space efficient data structure that can support both π and π^{-1} in constant time. Let $t \geq 2$ be a parameter. We first represent the sequence $\pi(i)$, for $i = 0$ to $n - 1$ using the representation of Theorem 3 taking $n \lg n + o(n)$ bits. In addition, we trace the cycle structure of the perm, and for every cycle whose length is at least t , we store a shortcut pointer with the elements which are at a distance of a multiple of t steps from an arbitrary starting point (this idea was used in the representation of an implicit multikey table to support logarithmic searches under any key [4]). The shortcut pointer points to the element which is t steps before it in the cycle of the perm.

More precisely, let $c_0, c_1, \dots, c_{k-1}$ be the elements of a cycle of the perm (i.e. $\pi(c_i) = c_{i+1 \bmod k}$, for $i = 0, 1, \dots, k - 1$) where $k \geq t$. Then the indices whose π values are c_{it} , for $i = 0, 1, \dots, l = \lfloor k/t \rfloor$, are called indices with shortcut pointers, and the shortcut pointer value at c_{it} stores the index whose π value is $c_{(i+1 \bmod l)t}$, for $i = 0, 1, \dots, l$ (see Fig. 1). Let $s \leq n/t$ be the number of shortcut pointers after doing this for every cycle of the perm. By Theorem 3 we can store the pointer values in the order of the indices with shortcut pointers (regardless of which cycle each element belongs to), using $s \lg n + O(s \lg \lg s / \lg s)$ bits. Since $s \leq n/t$, we have used $(1 + 1/t)n \lg n + O(n \lg \lg n / \lg n)$ bits along with the representation for π .

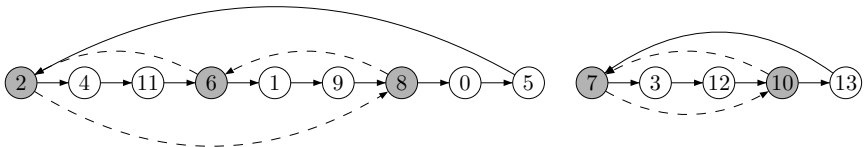


Fig. 1. Shortcut method. Solid lines denote the perm, and the dotted lines denote the back pointers. The shaded nodes indicate the positions having shortcut pointers.

We need to identify in $O(1)$ time, indices having shortcut pointers and for those indices, their pointer values. The pointer value of an index can be found from the representation S of the sequence of pointer values, if we know the rank of the index (having a shortcut pointer) among those having shortcut pointers. This can be supported in constant time by storing the indices having shortcut pointers using the FID of Theorem 2 using $\lceil \lg \binom{n}{s} \rceil + O(n \lg \lg n / \lg n)$ bits, which is $O((n \lg t)/t) + o(n)$ bits as $s \leq n/t$.

Let A , S and D be the representations of the permutation, the sequence of pointer values and the FID containing indices with shortcut pointers, respectively. The following procedure computes $\pi^{-1}(x)$ for a given x :

```

i := x;
  while  $\pi(i) \neq x$  do
    if  $i \in D$  and  $\text{rank}(i, D) = r$            // both found by querying D
      then  $j := r$ -th pointer value // found by querying S
      else  $j := \pi(i)$                 // found by querying A
    i := j;
  endwhile
return i

```

Since we have a shortcut pointer for every t elements of a cycle, the number of π computations made by the algorithm is at most $t + 1$. So the algorithm to compute π^{-1} takes at most $O(t)$ steps. Thus we have

Theorem 4. *There is a representation of an arbitrary perm π on $[n]$ using at most $(1 + 1/t)n \lg n + O(n \lg \lg n / \lg n)$ bits that can support the operations $\pi()$ in constant time, and $\pi^{-1}()$ in $O(t)$ time, for any parameter $t > 0$.*

Choosing t to be approximately $2/\epsilon$ for any positive constant $\epsilon < 1$, we have

Corollary 1. *There is a representation to store a perm π on $[n]$ using at most $(1 + \epsilon)n \lg n + O(1)$ bits in which one can support $\pi()$ in $O(1)$ time and $\pi^{-1}()$ in $O(1/\epsilon)$ time, for any positive constant ϵ less than 1.*

Choosing t to be $f(n) \lg n$ for some increasing function f of n we have:

Corollary 2. *There is a representation to store an arbitrary perm π on $[n]$ using at most $n \lg n + o(n)$ bits that can support $\pi()$ in constant time, and $\pi^{-1}()$ in $O(f(n) \lg n)$ time where $f(n)$ is any increasing function of n . (The $o(n)$ term is $O(n/f(n) + n \lg \lg n / \lg n)$.)*

Optimality. Demaine and López-Ortiz [3] showed that any text index supporting linear time substring searches requires about as much space as the original text. Here, given a text T , we want to construct an index I such that given any pattern P , one can find an occurrence of P in T in $O(|P|)$ time. They show that any index I supporting a search for a pattern P using $O(|P|)$ bit probes to the text T should have size $|I| = \Omega(|T|)$. They also show the following trade-off:

Theorem 5 (Corollary 3.1 of [3]). *If there is an algorithm supporting substring searches of length $|P| = \lg n + o(\lg n)$ using at most $\mathcal{S} = o(\lg^2 n / \lg \lg n)$ bit probes to a text of size $|T| = n \lg n + o(n \lg n)$, then $|I| = \Omega(|T| \lg n / \mathcal{S})$.*

They show this by considering texts that are obtained by writing a random perm π (with high Kolmogorov complexity) as $T_\pi = \pi(0)\#\pi(1)\#\dots\#\pi(n-1)$, and restrict the patterns to be $i\#$ for some $i \in [n]$. Note that searching for $i\#$ in T_π is equivalent to finding $\pi^{-1}(i)$ (i.e., $\pi^{-1}(i)$ is the position of $i\#$ in T_π). From their proof, for the RAM model with word size $\Theta(\lg n)$, one can show that

Corollary 3. *Let P be a structure that stores a perm π and answers $\pi(i)$ queries in $O(1)$ time. Then any data structure that answers $\pi^{-1}(i)$ queries using t queries to P , where t is $o(\lg n / \lg \lg n)$, requires an additional index structure taking at least $\Omega((n \lg n)/t)$ bits of space.*

Proof Sketch. The t queries to the structure P can be simulated with $t(\lg n + o(\lg n))$ bit probes to the text T_π . $\square$

This, in particular, implies that the structure of Theorem 4 is ‘essentially’ optimal.

3.2 Supporting Arbitrary Powers

There is no easier way, in the structure of Theorem 4, to compute π^k for $k > 1$ (or $k < 1$) than by repeated application of π or π^{-1} . Here we develop a succinct structure to support all powers of π (including π and π^{-1}).

Theorem 6. *Suppose there is a representation R taking $s(n)$ bits to store an arbitrary perm π on $[n]$, that supports $\pi()$ in p steps, and $\pi^{-1}()$ in q steps. Then there is a representation for an arbitrary perm on $[n]$ taking $s(n) + n + o(n)$ bits in which $\pi^k()$ for any k can be supported in time $p + q + O(1)$.*

Proof. Let π be the given perm to be represented to support all its powers. Consider its cycle representation, which is a collection of disjoint cycles of the perm (where the cycles are ordered arbitrarily). Remove the brackets and consider the resulting sequence as an array A of length n . Let ψ be the perm that maps i to the position j of i in the array; i.e. j such that $A[j] = i$. Equivalently, $\psi^{-1}(j) = A[j]$. For example, if π on 12 elements is given by $(1\ 5\ 8\ 3)(2\ 4\ 11)(6\ 10)(7\ 0\ 9)$, then the resulting sequence is $1\ 5\ 8\ 3\ 2\ 4\ 11\ 6\ 10\ 7\ 0\ 9$, and ψ is the perm given by $\psi(0) = 10; \psi(1) = 0, \psi(2) = 4$ and so on.

Now we will represent the perm ψ using the representation R taking $s(n)$ bits where we can support $\psi(i)$ and $\psi^{-1}(i)$ in time p and q respectively. In addition, we need to store the starting points (or the lengths) of each cycle of π efficiently. Let F be the indices of the starting points of the cycles of π . We store F using the FID representation of Theorem 1 taking $n + o(n)$ bits. This justifies the space usage in the theorem, and we are ready to explain how powers of π can be computed. To compute $\pi^k(i)$, we first find $j = \psi(i)$. Next we need to find the cycle C that contains i , and its length. Querying $\text{fullrank}(j, F) = p$ gives the number of elements of F less than j which gives the cycle number (in the left to right order of the cycles) of the cycle C . Then the length l of the cycle C_j is $\text{select}(p + 1, F) - \text{select}(p, F)$. Let $r = \text{select}(p, F)$ be the index where the p -th cycle starts. We find $s = r + ((j - r + k) \bmod l)$ and return $\psi^{-1}(s)$ which gives $\pi^k(i)$. Note that this works for both $k > 0$ and $k < 0$. Since the FID representation supports select and fullrank operation in constant time, we have the theorem. $\square$

It follows directly from Corollary 1 that:

Corollary 4. *There is a data structure to represent any perm π on $[n]$ using $(1 + \epsilon)n \lg n + O(1)$ bits in which we can support the operation $\pi^k(i)$ for any k in constant time, for any positive constant ϵ less than 1.*

4 Optimal-Space Representation

4.1 Representations Based on the Benes Network

Our results in this section are based on the Benes network, which is a communication network composed of a number of switches, and which we now outline (see [10] for details). Each switch has 2 inputs x_0 and x_1 and 2 outputs y_0 and y_1 and can be configured either so that x_0 is connected to y_0 (i.e. a packet that is input along x_0 comes out of y_0) and x_1 is connected to y_1 , or the other way around. An r -Benes network has 2^r inputs and outputs, and is defined as follows. For $r = 1$, the Benes network is a single switch with 2 inputs and 2 outputs. An $(r + 1)$ -Benes network is composed of 2^{r+1} switches and two r -Benes networks, connected as shown in Fig. 2(a). A particular setting of the switches of a Benes network *realises* a perm π if a packet introduced at input i comes out at output $\pi(i)$, for all i (Fig. 2(b)). The following properties are either easy to verify or well-known [10].

- An r -Benes network has $r2^r - 2^{r-1}$ switches, and every path from an input to an output passes through $2r - 1$ switches;
- For every perm π on $[2^r]$ there is a setting of the switches that realises π .

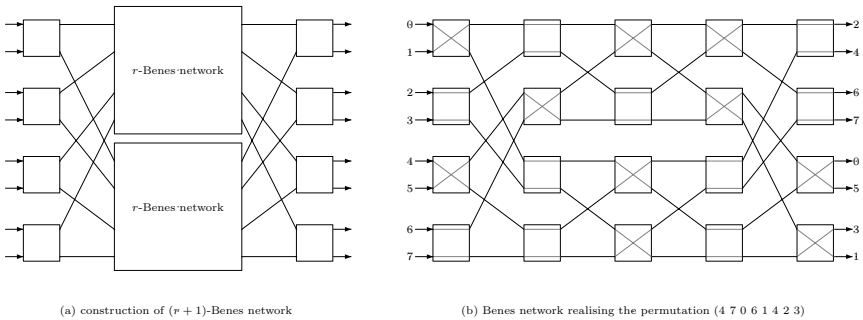


Fig. 2. The Benes network (construction) and an example

The restriction that the number of inputs be a power of 2 will prove to be a severe one in our context. We now define a family of Benes-like networks that admit greater flexibility in the number of inputs, namely the (q, r) -Benes networks,

for integers $r \geq 0, q > 0$. First, we define a q -permuter to be a communication network that has q inputs and q outputs, and realises any of the $q!$ perms of its inputs by some settings of its switches (an r -Benes network is a 2^r -permuter). Taking $p = q2^r$, a (q, r) -Benes network is a q -permuter for $r = 0$, and for $r > 0$ it is composed of p switches and two $(q, r - 1)$ -Benes networks, connected together in exactly the same way as a standard Benes network.

Lemma 1. *Let $q > 0, r \geq 0$ be integers and take $p = q2^r$. Then:*

1. *A (q, r) -Benes network consists of $qr2^r$ switches and 2^r q -permuters;*
2. *For every perm π on $[p]$ there is a setting of the switches of the (q, r) -Benes network that realises π .*

Proof. (1) is obvious; (2) can be proved in the same way as for a standard Benes network. $\square$

Representing Perms. Clearly, Benes networks may be used to represent perms. For example, if $n = 2^r$, a representation of a perm π on $[n]$ may be obtained by configuring an r -Benes network to realize π and then listing the settings of the switches in some canonical order (e.g. level-order). This represents π using $r2^r - 2^{r-1} = n \lg n - n/2$ bits. Given i , one can trace the path taken by a packet at input i by inspecting the appropriate bits in this representation, and thereby calculate $\pi(i)$ in $O(\lg n)$ time². In fact, by tracing the path back from output i we can also compute $\pi^{-1}(i)$ in $O(\lg n)$ time. To summarise:

Proposition 1. *When $n = 2^r$ for some integer $r > 0$, there is a representation of an arbitrary perm π on $[n]$ that uses $n \lg n - n/2$ bits and can support the operations $\pi()$ and $\pi^{-1}()$ in $O(\lg n)$ time.*

We now consider representations based on (q, r) -Benes networks; these will replace the central q -permuters with alternative representations of perms.

Proposition 2. *If $q \leq \lg n / (2 \lg \lg n)$, there is a representation of an arbitrary perm π on $[q]$ that supports $\pi()$ and $\pi^{-1}()$ in $O(1)$ time. This assumes access to a pre-computed table of size $O(\sqrt{n} \lg n)$ bits that does not depend upon π .*

Proof. We represent π implicitly, e.g. as the index of π in a canonical enumeration of all perms on $[q]$. The calculation of $\pi()$ (or $\pi^{-1}()$) is done by table lookup; the size of the required table is easily seen to be $O(\sqrt{n} \lg n)$ bits. $\square$

Using the representation of Proposition 2, we now obtain:

Lemma 2. *If $p = q2^r$ for integers $\lg n / (4 \lg \lg n) < q \leq \lg n / (2 \lg \lg n)$ and $r \geq 0$, then there is a representation of an arbitrary perm π on $[p]$ that uses $\mathcal{P}(p) + \Theta((p \lg p)/q)$ bits, and supports $\pi()$ and $\pi^{-1}()$ in $O(r)$ time each. This assumes access to a pre-computed table of size $O(\sqrt{n} \lg n)$ bits that does not depend upon π .*

² Indeed, in $O(\lg n)$ bit-probes.

Proof. Consider a (q, r) -Benes network that realises π ; we list all the switch settings of the outer $2r$ layers of switches as in Proposition 1. For each of the q -permuters we represent the perm realised by it using Proposition 2. Computing $\pi()$ or $\pi^{-1}()$ involves the inspection of $2r$ bits in the outer layers, plus a table lookup in the centre. We now calculate the space used. Note that:

$$\begin{aligned}\mathcal{P}(p) &= p \lg(p/e) + \Theta(\lg p) = q2^r(r + \lg(q/e)) + \Theta(\lg p) \\ &= qr2^r + 2^r(q \lg(q/e)) + \Theta(\lg p)\end{aligned}$$

By Lemma 1, space used by the above representation (excluding lookup tables) is $qr2^r + 2^r\mathcal{P}(q) = qr2^r + 2^r q \lg(q/e) + \Theta(2^r \lg pq) = \mathcal{P}(p) + \Theta((p \lg p)/q)$. $\square$

For perms on arbitrary $[n]$, we need the following proposition:

Proposition 3. *For all integers $p, t \geq 0$, $p \geq t$ there is an integer $p' \geq p$ such that $p' = q2^r$ for integers $t < q \leq 2t$ and $r \geq 0$, and $p' < p(1 + 1/t)$.*

Proof. Take q to be $\lceil p/2^r \rceil$, where r is the power of 2 that satisfies $t < p/2^r \leq 2t$. Note that $p' < (p/2^r + 1) \cdot 2^r = p(1 + 2^r/p) < p(1 + 1/t)$. $\square$

Theorem 7. *An arbitrary perm π on $[n]$ may be represented using $\mathcal{P}(n) + o(n)$ bits, such that $\pi()$ and $\pi^{-1}()$ can both be computed in $O(\lg n / \lg \lg n)$ time.*

Proof. Let $t = (\lg n)^2$. We first consider representing a perm ψ on $[l]$ for some integer l , $t < l \leq 2t$. To do this, we find an integer $p = l(1 + O(\lg \lg n / \lg n))$ that satisfies the preconditions of Lemma 2; such a p exists by Proposition 3. An elementary calculation shows that $\mathcal{P}(p) = \mathcal{P}(l)(1 + O(\lg \lg n / \lg n)) = \mathcal{P}(l) + O(\lg n (\lg \lg n)^2)$. We extend ψ to a perm on $[p]$ by setting $\psi(i) = i$ for all $l \leq i < p$ and represent ψ . By Lemma 2, ψ can be represented using $\mathcal{P}(p) + \Theta(\lg n (\lg \lg n)^2) = \mathcal{P}(l) + \Theta(\lg n (\lg \lg n)^2)$ bits such that $\psi()$ and $\psi^{-1}()$ operations are supported in $O(\lg \lg n)$ time, assuming access to a pre-computed table of size $O(\sqrt{n} \lg n)$ bits.

Now we represent π as follows. We choose an $n' \geq n$ such that $n' = n(1 + 1/(\lg n)^2)$ and $n' = q2^r$ for some integers q, r such that $t < q \leq 2t$. Again we extend π to a perm on $[n']$ and represent this extended perm. As in Lemma 2 we start with a (q, r) -Benes network that realises π and write down the switch settings of the $2r$ outer levels in level-order. The perms realised by the central q -permuters are represented as above. Ignoring any pre-computed tables, the space requirement is $qr2^r + 2^r(\mathcal{P}(q) + \Theta(\lg n (\lg \lg n)^2))$ bits, which is again easily shown to be $\mathcal{P}(n') + \Theta((n' \lg n')/q + 2^r \lg n (\lg \lg n)^2) = \mathcal{P}(n') + \Theta(n (\lg \lg n)^2 / (\lg n))$ bits. Finally, as above, $\mathcal{P}(n') = (1 + O(1/(\lg n)^2))\mathcal{P}(n)$, but the space requirement is still $\mathcal{P}(n) + \Theta(n (\lg \lg n)^2 / (\lg n)) = \mathcal{P}(n) + o(n)$ bits.

The running time for $\pi()$ and $\pi^{-1}()$ is clearly $O(\lg n)$. To improve this to $O(\lg n / \lg \lg n)$, we now explain how to step through multiple levels of a Benes network in $O(1)$ time, taking care not to increase the space consumption significantly. Consider a (q, r) -Benes network and let $t = \lfloor \lg \lg n - \lg \lg \lg n \rfloor - 1$.

Consider the case when $t \leq r$ (the other case is easier), and consider input number 0 to the (q, r) -Benes network. Depending upon the settings of the switches, a packet entering at input 0 may reach any of 2^t switches in t steps. A little thought shows that the only packets that could appear at the inputs to these 2^t switches are the 2^{t+1} packets that enter at inputs $0, 1, k, k+1, 2k, 2k+1, \dots$, where $k = q2^{r-t}$. The settings of the $t2^t$ switches that could be seen by any one of these packets suffice to determine the next t steps of *all* of these packets. Hence, when writing down the settings of the switches of the Benes network in the representation of π , we write all the settings of these switches in $t2^t \leq (\lg n)/2$ consecutive locations. Using table lookup, we can then step through t of the outer $2r$ layers of the (q, r) -Benes network in $O(1)$ time. Since computing the effect of the central q -permuter takes $O(\lg \lg n)$ time, we see that the overall running time is $O(r/t + \lg \lg n) = O(\lg n / \lg \lg n)$. $\square$

4.2 Powers of π

Using Theorems 6 and Theorem 7, one can get a structure that supports arbitrary powers of π in $O(\lg n / \lg \lg n)$ time using $\mathcal{P}(n) + n + o(n)$ bits of space. We show how to reduce the space to optimal $\mathcal{P}(n) + o(n)$ bits while retaining the query time bounds. Recall that in Theorem 6 we store the set of cycle start points $F \subseteq [n]$ using $n + o(n)$ bits. Simply using the FID of Theorem 2 to represent F will not work, as $|F|$ could be about $n/2$, and so the FID would take space at least $\left\lceil \lg \binom{n}{|F|} \right\rceil = \Theta(n)$ bits.

We develop below a different structure that takes $o(n)$ bits. We first order the cycles in non decreasing order of their lengths. Then we distinguish between *long* cycles whose length is greater than $\lceil \lg^2 n \rceil$ and *short* cycles whose length is at most $\lceil \lg^2 n \rceil$, and represent their starting points differently.

We take the representation of the starting points of long cycles first. Let S be the set of all starting points of long cycles. Let $|S| = k \leq n/\lg^2 n$. For this range of k , Theorem 2 gives an $o(n)$ bit FID structure for S , but we develop a simpler structure here.

To support $\text{select}(i, S)$ operation we simply store the elements in sorted order using $k \lceil \lg n \rceil$ bits which is $o(n)$. To support $\text{rank}(j, S)$ operation on S , we first divide the universe $[n]$ into blocks of size $\lceil \lg n \rceil$. Then we keep the set E of indices (from 1 to $n/\lceil \lg n \rceil$) having non-empty blocks in an FID of Theorem 1 using $n/\lceil \lg n \rceil + o(n)$ bits. I.e. we have a bit vector for each block indicating whether or not it is non-empty and keep an auxiliary structure for this bit vector E (of size $n/\lceil \lg n \rceil$) to support the rank operation on E . Then we represent the non-empty blocks completely using a bit vector F of at most $k \lceil \lg n \rceil$ (since at most k of the blocks can be non-empty) and build an auxiliary structure for this bit vector (of size at most $n/\lg n$) to support rank and select operations on F .

Now to find $\text{fullrank}(i, S)$, we find the number of non-empty blocks up to the block containing i by querying $r = \text{fullrank}(i/\lceil \lg n \rceil, E)$. If the block containing i is non-empty, this gives the rank of the block among the non-empty blocks and the position of i in the bit vector F . So querying rank up to that position in F

gives the $\text{fullrank}(i, S)$. If the block containing i is empty, then $s = \text{select}(r, E)$ gives the position of the previous non-empty block. Then $\text{fullrank}(s \lceil \lg n \rceil, F)$ gives the answer to $\text{fullrank}(i, S)$.

To represent the starting points of the short cycles, we first construct the multiset M that contains, for every $i = 1$ to $\lceil \lg^2 n \rceil$, $m_i = \sum_{j \leq i} j * n_j$ where n_j is the number of cycles of length j . This is a multiset since if there is no cycle of length $i + 1$, then $n_{i+1} = 0$ and hence $m_i = m_{i+1}$. For example, suppose $\lceil \lg^2 n \rceil = 8$ and in π there are 5 cycles of length 1, 4 cycles of length 4, 5 cycles of length 5 and 6 cycles of length 8, and 0 cycles of remaining lengths (up to 8). Then the multiset M we need to store is $\{5, 5, 5, 21, 46, 46, 46, 94\}$.

Let D be the set of distinct elements of M , and let R be the sequence of multiplicities of elements of D in increasing order. I.e., the i -th element of R is the number of occurrences of the i -th smallest element of D . Let P be the sequence of partial sums of elements of R . I.e., the i -th element of P is the sum of the first i elements of R . For the example outlined above, $D = \{5, 21, 46, 94\}$ and $P = \{3, 4, 7, 8\}$. We represent D and P using the set representation outlined earlier (to represent S), that can support $\text{fullrank}()$ and $\text{select}()$ operations in constant time. We will also explicitly store the last element L of M (which gives the starting point of the first long cycle). Note that $|D| = |P| \leq \lceil \lg^2 n \rceil$ and so the representation for D and P (and hence M) takes $O(\lg^3 n)$ bits. Hence along with the space for representing S ($O(n/\lg n)$ bits) the space used is $o(n)$.

From the proof of Theorem 6, we see that, to compute $\pi^k(i)$, we need to find the following two quantities: l , the length of the cycle containing i and r , the starting position of the cycle containing i . If i is in a long cycle (which can be found by comparing the position $j = \psi(i)$ of i with L), then the $\text{fullrank}()$ and $\text{select}()$ operations on S gives these information as in the proof of Theorem 6. We should just remember to add L to the starting point of these cycles.

If i falls in a short cycle, we first find $d = \text{fullrank}(j, D)$ (note j is the position of i in the list). This gives the number of distinct elements in M less than j . Then $s = \text{select}(d, P)$ gives the total number of elements less than j in M . So $l = s + 1$ is the length of the cycle containing i . If $\text{select}(d, D) = t$, then $t + 1$ is the starting point of the groups of cycles of length l in π . So $(j - t) \bmod l$ gives the position of i in its cycle, and so $r = j - ((j - t) \bmod l) + 1$ is the starting point r of its cycle. With these operations supported in constant time, we have:

Theorem 8. *Suppose there is a representation R taking $s(n)$ bits to store an arbitrary perm π on $[n]$, that supports $\pi()$ in p steps, and $\pi^{-1}()$ in q steps. Then there is a representation for an arbitrary perm π on $[n]$ taking $s(n) + o(n)$ bits in which $\pi^k()$ for any k can be supported in time $p + q + O(1)$.*

As an immediate corollary, we get, from Theorem 7

Corollary 5. *There is a representation to store an arbitrary perm π on $[n]$ using at most $\lceil \lg n! \rceil + o(n)$ bits that can support $\pi^k()$ for any k in $O(\lg n / \lg \lg n)$ time.*

Acknowledgements. The authors thank Rasmus Pagh for directing attention to the problem of representing a sequence of numbers, and Rick Thomas and Eugene Zima for several useful discussions.

References

1. D. A. Bader, M. Yan, B. M. W. Moret. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. University of New Mexico Technical Report HPCERC2001-005 (August 2001): <http://www.hpcerc.unm.edu/Research/tr/HPCERC2001-005.pdf>
2. A. Z. Broder, M. Charikar, A. M. Frieze and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer System Sciences*, **60** 630–659 (2000).
3. E. D. Demaine and A. López-Ortiz. A linear lower bound on index size for text retrieval. *Journal of Algorithms*, to appear.
4. A. Fiat, J. I. Munro, M. Naor, A. A. Schäffer, J.P. Schmidt and A. Siegel. An implicit data structure for searching a multikey table in logarithmic time. *Journal of Computer and System Sciences*, **43** 406–424 (1991).
5. R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proceedings of the ACM Symposium on Theory of Computing*, 397–406, 2000.
6. G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 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. D. E. Knuth. *The Art of Computer Programming, vol. 1: Fundamental Algorithms*. Computer Science and Information Processing. Addison-Wesley, 1973.
9. D. E. Knuth. *The Art of Computer Programming, vol. 3: Sorting and Searching*. Computer Science and Information Processing. Addison-Wesley, 1973.
10. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*. Computer Science and Information Processing. Morgan Kaufman, 1992.
11. P. B. Miltersen. The bit probe complexity measure revisited. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science*, LNCS **665** 662–671, Springer-Verlag, 1993.
12. J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, **31**(3) 762–776 (2002).
13. N. Pouyanne. On the number of permutations admitting an m -th root. *The Electronic Journal of Combinatorics*, **9** (2002), #R3.
14. R. Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal on Computing*, **31**(2) 353–363 (2001).
15. R. Raman, V. Raman and S. S. Rao. Succinct indexable dictionaries with applications to encoding k -ary trees and multisets. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 233–242, 2002.

Succinct Dynamic Dictionaries and Trees^{*}

Rajeev Raman¹ and Satti Srinivasa Rao²

¹ Dept. of CS, Univ. of Leicester, Leicester LE1 7RH, UK. r.raman@mcs.le.ac.uk.

² School of CS, Univ. of Waterloo, Canada N2L 3G1. ssrao@monod.uwaterloo.ca

Abstract. We consider space-efficient solutions to two dynamic data structuring problems. We first give a representation of a set $S \subseteq U = \{0, \dots, m-1\}$, $|S| = n$ that supports membership queries in $O(1)$ worst case time and insertions into/deletions from S in $O(1)$ expected amortised time. The representation uses $\mathcal{B} + o(\mathcal{B})$ bits, where $\mathcal{B} = \lceil \lg \binom{m}{n} \rceil$ is the information-theoretic minimum space to represent S . This improves upon the $O(\mathcal{B})$ -bit solutions of Brodnik and Munro [2] and Pagh [16], and uses up to a log-factor less space than search trees or hash tables. The representation can also associate *satellite* data with elements of S .

We also show that a binary tree on n nodes, where each node has $b = O(\lg n)$ -bit data stored at it, can be maintained under node insertions while supporting navigation in $O(1)$ time and updates in $O((\lg \lg n)^{1+\epsilon})$ amortised time, for any constant $\epsilon > 0$. The space used is within $o(n)$ bits of the information-theoretic minimum. This improves upon the equally space-efficient structure of Munro et al. [15], in which updates take $O(\lg^c n)$ time, for some $c \geq 1$.

1 Introduction

Computer science deals with the storage and processing of information. With the rapid proliferation of information, it is increasingly important to focus on the storage requirements of this information, especially as it may be transmitted and copied several times over. Recently there has been a renewal of interest in the study of *succinct* representations of data [1,2,9,13,14,15,18,19], whose space usage is close to the information-theoretic lower bound, but which support operations as efficiently as their usual counterparts. Succinct representations have been found for dictionary operations on static sets, trees, tries, and bounded-genus graphs. With few exceptions [15,19] succinct representations are not dynamic.

In this paper, we consider representing *dynamic dictionaries* succinctly. A dictionary is arguably the single most important abstract data type, a basic formulation of which is as follows. Given a set $S \subseteq U = [m]^1$ support the operations **member**(x, S), which determines whether $x \in S$, and **insert**(x, S), which adds x to S . In many applications, one stores *satellite* data with each element of S , which is retrieved by a successful **member** query. Letting $|S| = n$, perfect hash tables take $O(1)$ worst-case and amortised expected time respectively, for

^{*} Supported in part by UISTRF project 2001.04/IT and EPSRC GR L/92150.

¹ For non-negative integers i , $[i] = \{0, 1, \dots, i-1\}$.

member and insert [5] and balanced trees take $\Theta(\lg n)$ worst-case time for both operations.

The information-theoretic lower bound on the space needed to represent S is $\mathcal{B}(m, n) = \lceil \lg \binom{m}{n} \rceil = n \lg m - n \lg n + O(n)$ (in what follows we abbreviate $\mathcal{B}(m, n)$ by $\mathcal{B}$). Unfortunately, standard solutions use significantly more than $\mathcal{B}$ bits. For example, a solution based on balanced search trees uses at least $n(\lceil \lg m \rceil + \lceil \lg n \rceil)$ bits (one key and one pointer field per node), which can be $\Theta(\log n)$ times more than necessary. A similar situation occurs for other standard solutions, e.g., ‘Cuckoo’ hashing [17] requires $(2 + \epsilon)n \lg m$ bits of storage².

Succinct dictionaries were studied by Cleary [3], who showed how to achieve $(1 + \epsilon)\mathcal{B} + O(n)$ bits with $O(1)$ *expected* time for **member** and **insert** using the strong assumption of *simple uniform hashing* [4, pp. 224]. Brodnik and Munro [2] and Pagh [16] gave optimal $(\mathcal{B} + o(\mathcal{B}))$ -bit representations for static dictionaries, and noted that their solutions may be dynamised, by increasing the space usage to $O(\mathcal{B})$. Obtaining an optimum-space dynamic representation was stated as an open problem by [2], which we solve in this paper. Our solution takes $\mathcal{B} + o(\mathcal{B})$ bits of storage and supports **member** in $O(1)$ worst-case time; **insert** takes $O(1)$ amortised expected time. When s -bit satellite data are associated with each $x \in S$, the space usage becomes $\mathcal{B} + ns + o(\mathcal{B} + ns)$ bits. The *load factor* of a dictionary is the ratio of the number of keys currently in the dictionary to the capacity of the table. Conventional wisdom holds that the time performance of hashing degrades at load factors exceeding $1 - \Omega(1)$. Our result shows that this is wrong in a very real sense.

The model of memory allocation is very important in succinct dynamic data structures. Our result is robust across two models, of theoretical and practical relevance. Earlier dynamic succinct data structures [1,15,19] assumed the existence of a ‘system’ memory manager that would allocate and free memory in variable-sized chunks (see model $\mathcal{M}_A$ below). This approach does not charge the data structure for space wastage due to *external fragmentation* [20, Ch 9]. It is known that if N is the maximum total size of the chunks in use simultaneously, the memory manager needs $\Omega(N \lg N)$ words of memory to serve the requests in the worst case. A decision version of the corresponding offline problem is known to be NP-complete, even when all block sizes are 1 or 2. Hence we also analyse the space requirement under the standard way of measuring space in the RAM model (see model $\mathcal{M}_B$ below).

We also consider the problem of representing dynamic n -node binary trees. The nodes of a binary tree have slots for left and right children; slots that do not point to other tree nodes are said to point to *external* nodes, of which there are $n + 1$. The user may associate $b = O(w)$ -bit *satellite* data with internal nodes, or external nodes, or both. The operations we would like to support, in $O(1)$ time, on the tree are: given a node, finding its parent, left child and right child, and in the course of traversing the tree, the size of the subtree rooted at the current node, the satellite datum associated with the current node (if any) and its pre-order number. We assume, as do previous authors [15], that all traversals

² Recently, Fotakis et al. [6] have improved this to $(1 + \epsilon)n \lg m$

start from the root and end at the root as well. The updates we consider are: adding or deleting a leaf, or inserting a node along an edge.

Unlike a pointer-based representation that uses $\Theta(n \lg n)$ bits, we are interested in representations that come close to the information-theoretic lower bound of $\lg \binom{2n}{n} = 2n - O(\lg n)$ plus the storage for satellite data. Our representation uses $(b+2)n + o(n)$ or $(2b+2)n + o(n)$ bits, depending on whether data is associated only with internal nodes, external nodes, or both. It supports navigation in $O(1)$ time and updates (insertion/deletion of leaf or internal nodes) in $O((\lg \lg n)^{1+\epsilon})$ amortised time, for any constant $\epsilon > 0$. This improves upon the equally space-efficient structure of Munro et al. [15], in which updates take $O(\lg^c n)$ time, for some $c \geq 1$. Munro and Raman [14] earlier showed how to represent a binary tree using $2n + o(n)$ bits and support operations in $O(1)$ time, but this representation is static.

For simplicity, we assume in the dictionary problem that m is a power of 2. We assume the standard *word RAM* model [10] with a word size $w = \lg m$ or $w = \Theta(\lg n)$ for the dictionary and tree problems, respectively. Our dynamic dictionary is based on the static one of Brodnik and Munro [2]. We use their high-level approach, considering several cases depending upon the relative values of n and m , using bucketing and applying table lookup for small universes. We overcome the main obstacle mentioned by [2], namely, how to use dynamic perfect hashing in our data structure while absorbing its high space cost. The dynamic binary tree has a high-level similarity to that of Munro et al. [15].

The rest of the paper is organised as follows: in Section 2 we describe our memory models and resolve management issues. In Section 3 we describe our dynamic dictionary, and Section 4 deals with dynamic trees.

2 Preliminaries

Memory Models. We consider two memory models denoted by $\mathcal{M}_A$ and $\mathcal{M}_B$. In $\mathcal{M}_A$ the algorithm calls built-in “system” procedures *allocate* and *free*. The call *allocate*(k) for some integer $k \geq 0$ returns a pointer p to a block of 2^k consecutive memory locations all initialised to 0. Each memory location is w bits long, as is the pointer p . The call requires $O(2^k)$ time and increases the space usage of the algorithm by $w \cdot 2^k$ bits. The call *free*(p) frees the block of consecutive locations that p was pointing to and reduces the space usage appropriately.

In $\mathcal{M}_B$, the algorithm has access to words numbered $0, \dots, 2^w - 1$. The space usage at any given time is simply $s + 1$ where s is the highest-numbered word currently in use by the algorithm (see [12] for details).

Collections of extendible arrays. An *extendible array* (EA) maintains a sequence of n equal-sized records, each assigned a unique index between 0 and $n - 1$, under the following operations:

- *access*(i): access the record with index i (for reading or writing),
- *grow*: increment n , creating a new record with index n , and
- *shrink*: decrement n , discarding the record with index n .

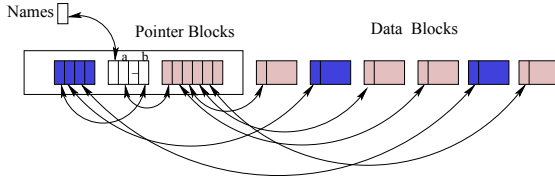


Fig. 1. Maintaining a collection of EAs in $\mathcal{M}_B$

We say that an EA with n records of r bits each has *nominal size* nr bits, and the nominal size of a collection of EAs is the sum of the nominal sizes of the EAs in the collection. We consider the problem of maintaining a collection of EAs under the following operations (the name of an EA is a w -bit integer):

- **create**(r): create an new empty EA with record size r and return its name,
- **destroy**(A): free the array A , and
- **grow**(A), **shrink**(A), **access**(i, A), which are as above.

We first look at some existing implementations. Brodnik et al. [1] gave an implementation of an EA with w -bit records that takes $n + O(\sqrt{n})$ words of space, where n is the current number of records in the array. This structure supports access in $O(1)$ time while supporting **grow** and **shrink** in $O(1)$ amortised time in the model $\mathcal{M}_A$ [1, p4]. Hagerup et al. [11] showed how to maintain a collection of EAs with w -bit records in $\mathcal{M}_B$, using $O(n)$ words of space where n is an *a priori* upper bound on the total size of the EAs. They supported all operations in $O(1)$ worst-case time, but assumed that the application keeps track of the starting position (which may change) of each EA. The space bound is $O(n + a)$ words where a is the number of EAs, if our interface is supported [12]. We now show how to maintain a collection of EAs with small memory overhead while supporting all the operations efficiently in both $\mathcal{M}_A$ and $\mathcal{M}_B$:

Lemma 1. *Let a be the current number of EAs in a collection of EAs and let s be the current nominal size of the collection. Then, this collection can be stored in $s + O(aw + \sqrt{sa}w)$ bits of space in $\mathcal{M}_A$, while supporting **access**, in $O(1)$ worst-case time, **create**, **grow** and **shrink** in $O(1)$ amortised time, and **destroy**(A) in amortised $O(s'/w)$ time, where s' is the current nominal size of A . In $\mathcal{M}_B$ the same holds, except that the space bound is $s + O(a^*w + \sqrt{sa^*}w)$ bits, where a^* is the maximum number of EAs that ever existed simultaneously in the past.*

Proof. We first discuss space usage in $\mathcal{M}_A$. In our implementation, an EA in the collection, whose nominal size is s' , consists of $O(\sqrt{s'/w})$ data blocks of sizes $1, 2, \dots, k$, where k is the minimum integer such that $\sum_{i=1}^k i \geq s'/w$, pointed to by a *pointer block* of size $O(\sqrt{s'/w})$. All the records of the EA are stored consecutively by considering the data blocks, in the increasing order of their sizes, as a sequence of (at least s') bits. Thus, the overhead of an EA with nominal size s' is $O(1 + \sqrt{s'/w})$ words. Now, suppose the i^{th} EA currently has

n_i records each of size r_i bits, for $1 \leq i \leq a$, and thus $s = \sum_{i=1}^a n_i r_i$. The total overhead in the EAs is $O(\sum_{i=1}^a (1 + \sqrt{n_i r_i / w}))$ words. This is maximised when all arrays have nominal size s/a , and is bounded by $O(a + \sqrt{sa/w})$ words. So the total space overhead is $O(aw + \sqrt{sa/w})$ bits as claimed.

For $\mathcal{M}_B$, we use the representation shown in Figure 1. The data in the EAs are stored in equal-sized data blocks of $k = \Theta(\sqrt{s/a^*w})$ bits each; k is changed if s or a^* doubles or halves since the last change to k . The data blocks of an EA need not be stored consecutively, and each EA has a pointer block that holds the location of its data blocks. Finally a *name* array provides the indirection needed to allow EAs to be referred to by their names. Briefly, the pointer blocks and name array are stored “loosely”, wasting a constant factor space, but this only affects the lower-order terms. $\square$

Corollary 1. *If in Lemma 1 at all times $a = \Omega(a^*)$, then the space overhead is the same for $\mathcal{M}_A$ and $\mathcal{M}_B$. If a (or a^*) is $o(s/w)$ at all times, the space used is $s + o(s)$ in $\mathcal{M}_A$ (or $\mathcal{M}_B$).*

Dynamic array. A key subroutine we use is for the *dynamic array* (DA) problem: Given a sequence of l records of $O(w)$ bits each, to support the operations of inserting a new record after, accessing a record at, and deleting a record at, a given index. In contrast to an EA, records may be inserted in the middle of the sequence, making it impossible to handle all operations in $O(1)$ time in general [8]. Hence we consider small DAs; the challenge is that we require the memory overhead to be small. We show (proof omitted):

Lemma 2. *For any constants $c, \epsilon > 0$, there is a DA that stores a sequence of length $l = w^c$, and supports accesses in $O(1)$ time and updates (insert/delete) in $O((\lg w)^{1+\epsilon})$ amortised time, using $o(l)$ bits of extra space.*

3 A Succinct Dynamic Dictionary

As mentioned above, we consider several—in fact, three—cases based on the *density* of the set, namely how close n is to m . We begin with a simple but key observation regarding the dynamic dictionary of Dietzfelbinger et al. [5] that enables us to absorb its high space usage. Their dictionary takes at most $35(1+c)n$ words of $\lg m$ bits each, for any fixed $c > 0$, where n is the current size of the set, and supports **member** in $O(1)$ time and **insert** in $O(1)$ expected amortised time. The ideas are to store the keys in an EA and store only pointers to this EA in the hash table, and to use universe reduction [7] to represent secondary hash functions more compactly. We get the following (proof omitted):

Lemma 3. *A set $S \subseteq [m]$ can be stored using two extendible arrays of total nominal size at most $(n+1) \lg m + 280n \lg n$ bits of space, while supporting **insert** in $O(1)$ expected amortised time and **member** in $O(1)$ time, where $n = |S|$. This structure requires $O(n)$ expected initialisation time.*

Sparse sets. We give an approach that works for all densities but is optimal only for sufficiently sparse sets. If $n \leq w^4$, we use the structure of Lemma 3 to represent the set. Otherwise, the data structure is a tree of constant depth, whose leaves contain sets of keys. We describe the tree in terms of a recursive construction algorithm for a static set S , and sketch the dynamisation later.

If $n > w^4$, we let N be a power of 2 such that $n \in [N, 2N)$. The root of the tree consists of a *bucket array* B of size N/w^2 , each element of which can store a w -bit pointer. We place each key in S into bucket i , $i = 0, \dots, N/w^2 - 1$ depending upon the value of its top $\lg N - 2 \lg w$ bits, and denote by S_i the set of keys placed into the i -th bucket. For $i = 0, \dots, N/w^2 - 1$, we recurse on S_i . Since all keys in S_i have the same top-order $\lg N - 2 \lg w$ bits, we omit them; thus, e.g., all keys in $S_0, \dots, S_{N/w^2-1}$ are of length $\lg m - \lg N + 2 \lg w$ bits. A pointer to the root of the representations of S_i is placed in the i -th location of the root's bucket array, for all i . If ever a recursive problem has size $\leq w^4$, the set of (shortened) keys is represented using Lemma 3. In addition, recursion is terminated perforce after λ levels where λ is a constant to be determined later, and any remaining sets are represented using Lemma 3.

Lemma 4. *The total sizes of all the bucket arrays put together is $O(n/w^2)$.*

Proof. A set T at some level of recursion is refined using a bucket array of $O(|T|/w^2)$ locations. We ‘charge’ $O(1/w^2)$ locations to each element of T . Since the depth of recursion is $O(1)$, each element is charged for $O(1/w^2)$ locations overall. $\square$

Lemma 5. *The space used by the data structure is $\mathcal{B} + O(n \lg w + \sqrt{nw})$ bits.*

Proof. If $n < w^4$ the claim follows easily: the total nominal size of the EAs used is $s = (n+1) \lg m + O(n \lg n)$ bits. Since $O(1)$ EAs are used, by Lemma 2, the space overhead in either model is $O(w + \sqrt{sw})$ bits. Since $\sqrt{s} = O(\sqrt{n \lg m} + \sqrt{\lg m} + \sqrt{n \lg n})$, $\sqrt{sw} = O(\sqrt{nw})$. Thus the total space usage is $n \lg m + O(n \lg n + \sqrt{nw}) = \mathcal{B} + O(n \lg w + \sqrt{nw})$.

Now we assume that $n \geq w^4$. We allocate the bucket arrays as EAs by means of **create** and **grow** operations. By Lemma 4 the total nominal size of these EAs is $O(n/w) = o(n)$ bits. The data structure also consists of a number of leaves where Lemma 3 is applied. Arbitrarily number the leaves $1, 2, \dots, l$, let the set at the i -th leaf be L_i , and let $n_i = |L_i|$. Note that $l = O(n/w^2)$ by Lemma 4, and that $\sum_i n_i = n$. We now add up the nominal space usage in the leaves.

Consider the i -th leaf, and suppose the set L_i stored there is obtained by refining (through bucketing) a series of sets $S \supseteq S^{(1)} \supseteq \dots \supseteq S^{(k)} \supseteq L_i$. For $j = 1, \dots, k$ let $N^{(j)}$ be a power of 2 such that $N^{(j)} \leq |S^{(j)}| < 2N^{(j)}$. Recall that the keys in $S^{(j)}$ are placed into $N^{(j)}/w^2$ buckets, and so all keys in $S^{(j+1)}$ have the same value in their $\lg N^{(j)} - 2 \lg w$ (next) most significant bits, and these bits may be omitted. Thus, the keys in L_i are of length $\lg m - \lg N - \lg N^{(1)} - \dots - \lg N^{(k)} + 2k \lg w \leq \lg m - \lg |S| - \lg |S^{(1)}| - \dots - \lg |S^{(k)}| + (2k+1) \lg w \leq \lg(m/n) - k \lg n_i + (2k+1) \lg w$.

If $|L_i| \leq w^4$ then clearly L_i is stored in EAs of total nominal size $(n_i + 1) \lg(m/n) + O(n_i(\lg n_i + \lg w)) = n_i \lg(m/n) + O(w + n_i \lg w)$ bits. Otherwise, L_i has undergone bucketing λ times, and it is stored in $(n_i + 1) \lg(m/n) + O(n_i \lg n_i) - (\lambda - 1)n_i \lg n_i + n_i(2k + 1)\lambda \lg w = n_i \lg(m/n) + O(w + n_i \lg w)$ bits if λ is large enough. Summing over all i gives that the overall nominal space usage is $n \lg(m/n) + O(n/w + n \lg w) = \mathcal{B} + O(n \lg w)$ bits.

By Corollary 1, the overhead of maintaining the EAs in both models is the same. Letting $s = \mathcal{B} + O(n \lg w) = O(nw)$ and $a = O(n/w^2)$, this overhead is $O(\sqrt{saw} + aw) = O(n + n/w) = O(n)$ bits, which is negligible. $\square$

Dynamising this data structure is straightforward. Insertions into the leaves are handled by Lemma 3. We also maintain the invariant that a bucket array x of size b has between bw^2 and $2bw^2$ keys under it. When this is violated, we rebuild the entire data structure under x , growing x to size $2b$. The amortised cost of this rebuilding at x is clearly $O(1)$, and as the depth of the tree is constant, the overall cost incurred by an insertion is also $O(1)$. Hence, we have:

Theorem 1. *A set $S \subseteq [m]$, $|S| = n$, can be represented using $\mathcal{B} + O(n \lg w + \sqrt{nw})$ bits, while supporting member in $O(1)$ worst-case time and insert in $O(1)$ expected amortised time. The space bound is valid in both $\mathcal{M}_A$ and $\mathcal{M}_B$.*

Denser Sets. Let k be a power of 2 such that $w^{\lg w} \leq 2^k < w^{2 \lg w}$. If $m/n > 2^k$, then $\mathcal{B} = \Omega(n(\lg w)^2)$, and the bound of $\mathcal{B} + O(n \lg w + \sqrt{nw})$ is easily seen to be $\mathcal{B} + o(\mathcal{B})$. Hence, we focus on the case $m/n \leq 2^k$.

Again, we proceed via bucketing. We allocate a bucket array of size $b = m/2^{2k}$ and divide S into b buckets, based on the top $\lg b = \lg m - 2k$ bits of each key. Within each bucket the keys may be truncated to their lowest-order $2k$ bits alone. The keys in bucket i are represented in an EA with $2k$ -bit records, and the name of this EA is placed in the i -th location of the bucket array. The representation of the keys in this EA is somewhat unusual. We use the EA to simulate the memory of a word RAM with word size $2k$. We say that a non-negative function f is *smooth* if $|f(t) - f(t')| = O(t' - t)$ for any integers t, t' with $t' > t \geq 0$. It is easy to show (proof omitted):

Proposition 1. *Let $m(t)$ be a smooth and $O(1)$ -time computable upper bound on the space usage (in $\mathcal{M}_B$) of a RAM algorithm. Then the algorithm can be simulated with a constant factor amortised time overhead, in an EA comprising $m(t)$ records of w' bits each, where $w' \leq w$ is the word size of the simulated RAM.*

We simulate the algorithm of Theorem 1 running in $\mathcal{M}_B$ to represent the keys in each bucket. Since this algorithm ultimately relies on Lemma 1 to manage memory, an inspection of the proof of this lemma shows that there is an accurate and easily-computable upper bound on its space usage, and that the bound is smooth. Letting $\mathcal{B}_i = \mathcal{B}(2^{2k}, n_i)$, where n_i is the number of keys in the i -th bucket, we see that the algorithm of Theorem 1 can be simulated in an EA, whose nominal size is at most $\mathcal{B}_i + O(n_i \lg k + \sqrt{n_i k})$ bits.

We now sum up these nominal sizes. Firstly, $\sum_{i=1}^b \sqrt{n_i} \leq \sqrt{bn}$; since $b = m/2^{2k} < n/2^k$, we see that $\sum_{i=1}^b k\sqrt{n_i} < kn/2^{k/2} = o(n)$. As noted in [2, Lemma 3.2], $\sum_{i=1}^b \mathcal{B}_i \leq \mathcal{B} + O(b)$. Thus, the total nominal size of all the EAs is $\mathcal{B} + O(n \lg k) = \mathcal{B} + O(n \lg \lg w)$ bits whenever $n \geq m/2^k$. Since the number of these EAs is $o(n/w)$, but their total nominal size is $\Omega(n)$ bits, the overhead of managing these is negligible. We have thus shown:

Lemma 6. *A set $S \subseteq [m]$, $|S| = n \geq m/(w^{2 \lg w})$, can be represented using $\mathcal{B} + O(n \lg \lg w)$ bits, while supporting member in $O(1)$ worst-case time and insert in $O(1)$ expected amortised time. The space bound is valid in both $\mathcal{M}_A$ and $\mathcal{M}_B$.*

Very Dense Sets. If $m/w^{\Omega(1)} > n \geq m/w^{2 \lg w}$, $\mathcal{B} = \Omega(n \lg w)$, so the space used by the data structure of Lemma 6 is $\mathcal{B} + o(\mathcal{B})$. Smaller values of n are covered by Theorem 1. We now focus on the case $n \geq m/w^\epsilon$, for positive constant $\epsilon < 1/4$ and show (proof omitted):

Lemma 7. *A set $S \subseteq [m]$, $|S| = n \geq m/w^\epsilon$, for some $0 < \epsilon < 1/4$, can be represented using $\mathcal{B} + O(m/w^{1/4})$ bits, while supporting member in $O(1)$ worst-case time and insert in $O(1)$ expected amortised time. The space bound is valid in both $\mathcal{M}_A$ and $\mathcal{M}_B$.*

The space bound of Lemma 7 is $\mathcal{B} + o(\mathcal{B})$ for $n \geq m/w^{1/5}$. For $m/(w^{2 \lg w}) \leq n < m/w^{1/5}$, the space bound of Lemma 6 is $\mathcal{B} + o(\mathcal{B})$. Smaller values of n are covered by Theorem 1. Thus we have:

Theorem 2. *A set $S \subseteq [m]$, $|S| = n$, can be represented using $\mathcal{B} + o(\mathcal{B})$ bits, while supporting member in $O(1)$ worst-case time and insert in $O(1)$ expected amortised time. The space usage may be measured either in $\mathcal{M}_A$ or $\mathcal{M}_B$.*

Satellite data. Here we describe how to augment our dynamic dictionary, so that s -bit satellite data can be associated with the keys, without excessive space overhead. We show:

Theorem 3. *A set $S \subseteq [m]$, $|S| = n$ where each element has an associated satellite data of $s \leq w$ bits, can be represented using $\mathcal{B} + ns + o(\mathcal{B} + ns)$ bits of space, while supporting member and finding the data associated with an element present in $O(1)$ time, and insert in $O(1)$ expected amortised time. The space bound is valid in both $\mathcal{M}_A$ and $\mathcal{M}_B$.*

Proof. We sketch the modifications required to the structure of Theorem 2.

case (i) $s = \omega(\lg w)$ or $n \leq m/w^{O(\lg w)}$: In this case, we represent the dictionary using the structure of Theorem 1. This consists of a constant depth tree structure whose leaves contain sets of keys which are stored using Lemma 3. This structure stores the actual keys in an EA and stores pointers to these keys in the hash table. In addition, we now store the satellite data corresponding to these keys in another EA, in the same order as their corresponding keys. Thus the pointer used

to index into the EA containing keys is also used to index into EA containing the satellite data. The overall extra space used by these EAs (containing the satellite data) is $O(n/w + \sqrt{(ns)(n/w^2)(w)}) = O(n/w + n\sqrt{s/w}) = O(n)$ bits (since $s \leq w$). Thus the overall space bound is $\mathcal{B} + ns + O(n \lg w + \sqrt{nw})$. This is $\mathcal{B} + ns + o(\mathcal{B} + n \lg s)$ whenever $s = \omega(\lg w)$ or $n = m/w^{O(\lg w)}$.

case (ii) ($s = O(\lg w)$ and $s = \omega(\lg \lg w)$) or $(m/w^{O(\lg w)} \leq n \leq m/w^{\Omega(1)})$: Here we use the solution of Lemma 6, i.e., recursively apply the above solution with word size $w' = O(\lg^2 w)$. Since $s \leq w'$ in this case, we can apply this recursively even for the satellite data. Thus a bucket containing n_i keys can be stored in an EA with nominal size $\mathcal{B}(n_i, 2k) + n_i s + O(n_i \lg w' + \sqrt{n_i w'})$. Summing this over all buckets, we get the space required in this case to be $\mathcal{B} + ns + O(n \lg k) + o(n)$. This is $\mathcal{B} + ns + o(\mathcal{B} + ns)$ whenever $s = \omega(\lg \lg w)$ or $n = m/w^{\Omega(1)}$.

case(iii) $s = O(\lg \lg w)$ and $n > m/w^{1/4}$: In this case we use the structure of Lemma 7 for very dense sets. In addition, we store the satellite data corresponding to the keys in poly-log sized blocks using a dynamic array that supports access in $O(1)$ time and updates in $O(1)$ amortized time, with a $o(1)$ bit space overhead per satellite datum. We improve upon the bounds of Lemma 2 by using precomputed tables, exploiting the fact that the record size is ‘small’. $\square$

Deletions and Further Refinements. Our approach handles deletions in the same time bound, but the details are messier. One problem is that Lemma 1 need not hold for deletions (in $\mathcal{M}_B$), as an EA with a very large name (given out when the number of EAs was large) may remain active while most lower-numbered EAs are deleted. This necessitates re-naming EAs by introducing back-pointers into the (fortunately $O(1)$) locations where the names of the EAs are held.

An issue that we have glossed over for lack of space is how to avoid excessive temporary blow-ups in space (e.g. as we move from one representation to another, or as we re-build some part of the data structure). The details vary depending on the instance. For example, when moving from the representation of Theorem 1 to the representation of Lemma 6, we build the new structure bucket by bucket, destroying one or more buckets in the old structure as each new bucket is completed. Since the buckets are of size $o(n)$ any temporary duplication is negligible. Finally, the constant factors can be greatly improved, at the expense of getting a slightly weaker result, by using [6] or [17] in place of [5].

4 Succinct Dynamic Binary Trees

We now consider the problem of representing dynamic n -node binary trees. We divide the given binary tree into connected *blocks* of size $\Theta(\lg^3 n)$ and each block into connected *subblocks* of size $\Theta(\lg n)$ (see [15]). Thus, there are $\Theta(n/\lg^3 n)$ blocks, each consisting of $\Theta(\lg^2 n)$ subblocks. Edges are classified as *intra-subblock pointers*, *inter-subblock pointers (ISPs)*, *inter-block pointers (IBPs)* and *external pointers*, in a natural way based on the endpoints of the edge. We now discuss the representations of blocks and sub-blocks. A block is represented as follows:

1. An EA A of $O(\lg^3 n)$ IBPs leaving the block, with each IBP taking $O(\lg n)$ bits. All IBPs leaving the same subblock are stored consecutively in A . Along with each IBP, we also store its pre-order number among all the IBPs leaving the block.
2. A DA B of the $\Theta(\lg^2 n)$ ISPs within this block. Each ISP is stored in $\Theta(\lg \lg n)$ bits; all ISPs leaving a subblock are stored consecutively in B .
3. A DA C of pointers to the representations (see (4)) of the $\Theta(\lg^2 n)$ subblocks of the block. These pointers are ordered so that the roots of the subblocks to which they point appear in a *pre-order consistent* manner, i.e. in the same order that they would in a pre-order traversal of the block
4. A set of $\sqrt{\lg n}$ EAs, with the i -th EA having record size $i\sqrt{\lg n}$, that store the representations of the subblocks (detailed in (8)-(11)) along with backpointers to B and C . The representation of each sub-block is padded out to the next multiple of $\sqrt{\lg n}$.
5. The prefix sums of the subtree sizes of all the child blocks of the block, where the subtree sizes appear in pre-order consistent manner.
6. The prefix sums of the sizes of all the subblocks within the block ordered in a pre-order consistent manner.
7. A constant number of $\lg n$ -bit pointers/data such as: a pointer to the parent block, position in its parent's list of IBPs (used for constant time navigation and also as backpointers), subtree size of the root node of the block, etc.

We now calculate the total nominal size of all the EAs. (1) As there are $O(n/(\lg n)^3)$ IBPs, this adds up to $O(n/\lg^2 n)$ bits over all blocks. (2, 3 or 6) $\Theta(\lg^2 n \lg \lg n)$ bits per block, or $O(n \lg \lg n / \lg n)$ bits overall. (4) We add up the sizes of the sub-block representations below. Here we only note that padding wastes $O(n/\sqrt{\lg n})$ bits overall. (5) Each prefix sum takes $O(\lg n)$ bits, but there are $O(n/(\lg n)^3)$ blocks, so this is $O(n/\lg^2 n)$ bits overall. (7) $O(\lg n)$ bits per block, or $O(n/(\lg n)^2)$ bits overall.

A sub-block is represented by concatenating the following bit-strings (we let ν be the number of nodes in this sub-block):

8. An implicit representation of the tree structure of the subblock.
9. A pointer to its list of IBPs stored in A , taking $\Theta(\lg \lg n)$ bits.
10. A pointer to its list of ISPs stored in B , taking $\Theta(\lg \lg n)$ bits. (The lengths of these lists can be obtained from the implicit representation of the subblock.)
11. The number of ISPs (within the block) before the sub-block's root in the pre-order numbering of the subblocks.
12. Number all the $\nu + 1$ edges leaving the sub-block $0, \dots, \nu$ in pre-order. Let $S, S' \subseteq [\nu + 1]$ be the set of ISPs and IBPs leaving the sub-block, respectively. A bit-string that contains $|S|, |S'|$ (using $\Theta(\lg \lg n)$ bits), followed by a bit-string of $\mathcal{B}(\nu + 1, |S|)$ bits that represents S implicitly (and likewise for S').

The size of this bitstring depends on ν , $|S|$ and $|S'|$. (8) takes 2ν bits, adding up to $2n$ bits overall and (9, 10 or 11) all take $O(\lg \lg n)$ bits, adding up to $O(n \lg \lg n / \lg n)$ bits overall. (12) takes $O(\lg \lg n) + \mathcal{B}(\nu + 1, |S|) + \mathcal{B}(\nu + 1, |S'|)$.

Using [2, Lemma 3.2] one can verify that (12) adds up to $O(n \lg \lg n / \lg n)$ bits overall. Note that (12) always takes less than 3ν bits, so we can choose the sub-block sizes small enough that the concatenation of (8)-(12) never exceeds $(\lg n)/2$ bits, allowing table lookup.

Any satellite data is stored in pre-order consistent manner in one of two DAs associated with each blocks, one each for internal and external node data. The total nominal sizes of these DAs are bn and $b(n+1)$ bits respectively.

Operations. Recall that we assume that all traversals start from the root and end at the root. We start a traversal at the root block, find its root subblock and navigate within it until we reach an edge leading out of the subblock. We then find out whether the edge is an IBP or ISP (if not, it points to an external node) and find its rank using the implicit representation of the subblock, follow the appropriate pointer to the corresponding block/subblock and traverse that.

To find the size of the subtree rooted at the current node, we find the sum of the subtree sizes of (the roots of) all the blocks leaving the current block that are within the subtree using the partial sum structure for the subtree sizes of the blocks and their pre-order numbers. We then find the sum of the sizes of all subblocks within the current block that are within the subtree using the prefix sums of the sizes of the subblocks. To these two numbers, we add the number of nodes in the current subblock that are within the subtree (which can be found in $O(1)$ time using a lookup table) to get the required subtree size.

To find the satellite datum of the current node, we first find the number of nodes before the root of the subblock (in the pre-order of the subblocks) using the prefix sums and then find the position of the node within the subblock, using a lookup table. The sum of these numbers gives the pre-order number of the node in the block, which is used to index into the satellite data block in $O(1)$ time. To find the pre-order number of the current node, we accumulate the sum of the sizes of all the blocks that are before the current block in pre-order as we traverse the tree down to the current node. To this we add the pre-order number of the node in the current block to find its pre-order number in the tree.

Many aspects of updates are easy. For example, subblocks are updated by table lookup in $O(1)$ time; the actions that need to be taken when a subblock or block gets too big are costly but infrequent and the amortised cost is easily seen to be $O(1)$. Apart from handling updates to the DAs, the only place where care is required is in updating the prefix sums. For this we use ideas from [19]. In summary:

Theorem 4. *There is a $2n+o(n)$ -bit binary tree representation that supports the following operations in $O(1)$ time: given a node, finding its parent, left child and right child, and in the course of traversing the tree, the size of the subtree rooted at the current node, the satellite datum associated with the current node (if any) and its pre-order number. We assume that traversals start and end at the root. Inserting a leaf or a node along an edge or requires $O((\lg \lg n)^{1+\epsilon})$ time. The representation can associate $b = O(w)$ -bit satellite data with external or internal nodes, or both, at an additional space cost of $bn + o(n)$ bits, or $2bn + o(n)$ bits,*

and can access the satellite datum associated with a node in $O(1)$ time. The space bounds are valid in both $\mathcal{M}_A$ and $\mathcal{M}_B$.

Acknowledgements. We thank Torben Hagerup and Venkatesh Raman for enlightening discussions.

References

1. A. Brodnik, S. Carlsson, E. D. Demaine, J. I. Munro and R. Sedgewick. Resizable arrays in optimal time and space. In *Proc. WADS '99*, LNCS 1663, 37–48.
2. A. Brodnik and J. I. Munro. Membership in constant time and almost minimum space. *SIAM J. Computing* **28** (1999), 1628–1640.
3. J. G. Cleary. Compact hash tables using bidirectional linear probing. *IEEE Trans. Comput.* **9** (1984), 828–834.
4. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 1990.
5. M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert and R. E. Tarjan. Dynamic perfect hashing: upper and lower bounds. *SIAM J. Computing*, 23 (1994), 738–761.
6. D. Fotakis, R. Pagh, P. Sanders and P. Spirakis. Space efficient hash tables with worst case constant access time. In *Proc. 20th STACS*, LNCS 2607, 271–282, 2003.
7. M. L. Fredman, J. Komlós and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31 (1984), 538–544.
8. M. L. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proc. 21st ACM STOC*, 345–354, 1989.
9. R. Grossi and J. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *Proc. 32nd ACM STOC*, 397–406, 2000.
10. T. Hagerup. Sorting and searching on the word RAM. In *Proc. 15th STACS*, LNCS 1373, 366–398, 1998.
11. T. Hagerup, K. Mehlhorn and J. I. Munro. Maintaining discrete probability distributions optimally. In *Proc. 20th ICALP*, LNCS 700, 253–264, 1993.
12. T. Hagerup and R. Raman. An efficient quasidictionary. In *Proc. 8th SWAT*, LNCS 2368, 1–18, 2002.
13. G. Jacobson. Space efficient static trees and graphs. In *Proc. 30th IEEE Symp. FOCS*, 549–554, 1989.
14. J. I. Munro and V. Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Proc. 37th IEEE Symp. FOCS*, 118–126, 1997.
15. J. I. Munro, V. Raman and A. Storm. Representing dynamic binary trees succinctly. In *Proc. 12th ACM-SIAM SODA*, 529–536, 2001.
16. R. Pagh. Low redundancy in static dictionaries with constant query time. *SIAM J. Comput.*, **31** (2001), 353–363.
17. R. Pagh and F. F. Rodler. Cuckoo Hashing. *Proc. 9th ESA*, LNCS 2161, 121–133, 2001.
18. R. Raman, V. Raman and S. S. Rao. Succinct indexable dictionaries, with applications to representing k -ary trees and multisets. In *Proc. 13th ACM-SIAM SODA*, 233–242, 2002.
19. R. Raman, V. Raman and S. S. Rao. Succinct dynamic data structures. In *Proc. 7th WADS*, LNCS 2125, 426–437, 2001.
20. A. Silberschatz, P. B. Galvin and G. Gagne. *Operating System Concepts*, 6th ed. John Wiley & Sons, 2001.

Labeling Schemes for Weighted Dynamic Trees

(Extended Abstract)

Amos Korman* and David Peleg*

The Weizmann Institute of Science,
Rehovot, 76100 Israel
{pandit,peleg}@wisdom.weizmann.ac.il

Abstract. This paper studies β -approximate distance labeling schemes, which are composed of a *marker* algorithm for labeling the vertices of a graph with short labels, coupled with a *decoder* algorithm allowing one to compute a β -approximation of the distance between any two vertices directly from their labels (without using any additional information). As most applications for informative labeling schemes in general, and distance labeling schemes in particular, concern large and dynamically changing networks, it is of interest to focus on *distributed dynamic* labeling schemes. The paper considers the problem on dynamic weighted trees and cycles where the vertices of the tree (or the cycle) are fixed but the (positive integral) weights of the edges may change. The two models considered are the *fully dynamic* model, where from time to time some edge changes its weight by a fixed quanta, and the *increasing dynamic* model in which edge weights can only grow. The paper presents distributed β -approximate distance labeling schemes for the two models, for $\beta > 1$, and establishes upper and lower bounds on the required label size and the communication complexity involved in updating the labels following a weight change.

1 Introduction

In order for a network representation method to be effective in the context of a large and distributed communication network, it must allow users to efficiently retrieve useful information about the network. Recently, a number of studies focused on a *localized* network representation method based on assigning a (hopefully short) *label* to each vertex, allowing one to infer information about any two vertices *directly* from their labels, without using *any* additional information sources. Labeling schemes have been developed for a variety of information types, including vertex adjacency [6,5,13], distance [19,17,12,11,9,14,21,7], tree routing [8,22], flow and connectivity [16], tree ancestry [1,3,4,15], and various other tree functions, such as center, least common ancestor, separation level, and Steiner weight of a given subset of vertices [20]. See [10] for a survey.

* Supported in part by a grant from the Israel Science Foundation.

By now, the basic properties of localized labeling schemes for *static* (fixed topology) networks are reasonably well-understood. However, when considering applications such as distributed systems and communication networks, the typical setting is dynamic, namely, the network topology undergoes repeated changes. Therefore, for a representation scheme to be useful in practice, it should be capable of reflecting up-to-date information in a dynamic setting, which may require occasional updates to the labels. Moreover, the algorithm for generating and revising the labels must be *distributed*, in contrast with the sequential and centralized label assignment algorithms described in the above cited papers.

The study of distributed labeling schemes for the dynamic setting was initiated in [18], which concentrates on the setting of an unweighted tree where at each step a leaf can be added to or removed from the tree. The labeling scheme presented therein for distances in this "leaf-dynamic" tree model has amortized message complexity $O(\log^2 n)$ per operation, where n is the size of the tree when the operation takes place. The protocol maintains $O(\log^2 n)$ bit labels, when n is the current tree size. This label size is known to be optimal even in the static scenario. A second result of [18] introduces a more general labeling scheme for the leaf-dynamic tree model, based on extending an existing *static* tree labeling scheme to a dynamic setting. The approach fits a number of natural tree functions, such as distance, separation level and flow. The main resulting scheme incurs an overhead of $O(\log n)$ multiplicative factor in both the label size and amortized message complexity in the case of dynamically growing trees (with no deletions). If an upper bound on n is known in advance, this method can yield a different tradeoff, with $O(\log^2 n / \log \log n)$ multiplicative overhead on the label size but only an $O(\log n / \log \log n)$ overhead on the amortized message complexity. In the non-restricted leaf-dynamic tree model (where both additions and deletions are allowed) the scheme incurs also an increased *additive* overhead in amortized communication, of $O(\log^2 n)$ messages per operation.

One key limitation of the setting studied in [18] is that the links are assumed to be unweighted. In reality, network distances are often based on link weights, and operator-initiated or traffic-dictated changes in these weights affect the resulting distances and subsequently the derived routing and circuit establishing decisions. In fact, whereas physical topology changes are relatively rare and are usually viewed as a disruption in the normal operation of the network, link weight changes are significantly more frequent and may be considered as part of the normal network operation. Subsequently, while it may be conceivable to approach physical topology changes by an offline label reorganization algorithm, this is an unreasonable approach when it comes to link weight changes, and a distributed update mechanism is desirable.

The current paper makes a step towards overcoming this limitation by investigating distance labeling schemes in dynamic settings involving changing link weights. The first model studied is the *fully dynamic model*. This model considers an underlying topology network with positive integer edge weights, where the vertices and edges of the network are fixed but at each time an edge weight can increase or decrease by a fixed quanta (which for notational convenience is

set to be 1), as long as the weight remains positive. (Our algorithms and bounds apply also for larger weight changes, as clearly, a weight change of $\Delta > 1$ can be handled, albeit naively, by simulating it as Δ individual weight changes of 1.)

The second model considered is the *increasing dynamic model* which is the fully dynamic model restricted to events where an edge weight can only increase by one at each step.

The underlying network topologies considered for the first model are trees and cycles. The second model considers only trees.

As shown in Sect. 2, any *exact* distance labeling scheme for either of the models cannot avoid a linear message complexity per operation in some worst-case scenarios. We therefore weaken the demands from the labeling scheme, and only require it to maintain a β -approximation of the distances (for $\beta > 1$) rather than exact distances. Such a scheme is referred to as a β -*approximate distance labeling scheme*.

Our main results are as follows. Throughout the paper, denote by n the number of vertices in the network, $G = (V, E)$. For a tree network, we present a β -approximate distance labeling scheme in each of the two models described above. Let W be the maximum weight assigned to an edge in the tree. Then, in both schemes, the maximum size of a label given to any vertex is bounded by $O(\log^2 n + \log n \log W)$ which as shown in [12] is optimal for (exact) distance labeling schemes even in the static scenario.

For an edge e , let $B(e, d)$ be the number of vertices at distance d or less from an endpoint of the edge e and let $\Lambda = \max\{B(e, d)/d \mid d \geq 1, e \in E\}$. Denote by m the number of edge changes occurring in the tree. We show that for $\beta > 1$ bounded away from 1, the message and bit complexities of the protocol for the fully dynamic model are $O(m\Lambda \log^2 n)$ and $O(m\Lambda \log^2 n \cdot \log \log n)$ respectively. The message and bit complexities for the increasing dynamic model are $O(m \log^2 n + n \log n \log m)$ and $O(m \log^2 n \cdot \log \log n + n \log n \log m \cdot \log \log n)$ respectively.

For the fully dynamic model, if the underlying network topology is a path we describe a different β -approximate distance labeling scheme yielding a different tradeoff between the size of the labels and the communication complexity. The scheme uses maximum label size $O(\log n \log m)$ and its message complexity is $O(m \log^2 n)$.

Similarly, if the underlying topology is a cycle then we get two schemes with the same asymptotic complexities as for the path.

2 Preliminaries

Our network model is restricted to either tree or cycle topologies. We assume that the vertices of the network are fixed and that the edges of the network are assigned positive integer weights. The network is assumed to dynamically change via weight changes of the edges. For two vertices u and v in some graph, denote by $d^\omega(u, v)$ the weighted distance between u and v . In the *fully dynamic* (tree or cycle) model the following events may occur:

1. An edge (u, v) increases its weight by one.
2. An edge (u, v) with weight at least 2 decreases its weight by one.

Subsequent to an event on an edge $e = (u, v)$, its end points u and v are informed of this event. In the *increasing dynamic* model the only event that may occur is that an edge (u, v) increases its weight by one and subsequently u and v are informed of this event.

For $\beta \geq 1$, a *static β -approximate distance labeling scheme* $\pi = \langle \mathcal{M}(\beta), \mathcal{D} \rangle$ for a family of graphs $\mathcal{F}$ is composed of the following components:

1. A *marker* algorithm $\mathcal{M}(\beta)$ that given a graph in $\mathcal{F}$, assigns labels to its vertices.
2. A polynomial time *decoder* algorithm $\mathcal{D}$ that given the labels $Label(u)$ and $Label(v)$ of two vertices u and v in some graph $G \in \mathcal{F}$, outputs a *distance estimate* $\tilde{d}^\omega(u, v)$ satisfying $\tilde{d}^\omega(u, v)/\beta \leq d^\omega(u, v) \leq \beta \cdot \tilde{d}^\omega(u, v)$.

A *static distance labeling scheme* is a static 1-approximate distance labeling scheme. For examples of static distance labeling schemes see [18,12,19].

In this paper we are interested in distributed networks where each vertex in the graph represents a processor. This does not affect the definition of the decoder algorithm of the labeling scheme, since it is performed locally, but the marker algorithm must be implemented as a *distributed marker protocol*.

The approximate labeling schemes for the fully dynamic and the increasing dynamic models involve a marker protocol $\mathcal{M}$ which is activated after every change in the network topology. The protocol $\mathcal{M}$ maintains the labels of all vertices in the underlying graph so that the corresponding decoder algorithm will work correctly. We assume that the topological changes occur sequentially and are sufficiently spaced so that the protocol has enough time to complete its operation in response to a given topological change before the occurrence of the next change. We refer to scenario where m weight changes have occurred as an *m-change scenario*.

For a dynamic β -approximate labeling scheme $\pi = \langle \mathcal{M}(\beta), \mathcal{D} \rangle$, for either one of the models, we are interested in the following complexity measures.

- *Label Size*, $\mathcal{LS}(\mathcal{M}(\beta), m)$: the maximum size of a label assigned by $\mathcal{M}(\beta)$ to a vertex in the worst case n -vertex underlying graph and the worst vase m -change scenario. (the graph classes considered are trees and cycles).
- *Message Complexity*, $\mathcal{MC}(\mathcal{M}(\beta), m)$: the maximum number of messages sent by $\mathcal{M}(\beta)$ in the worst case n -vertex underlying graph and the worst vase m -change scenario.
- *Bit Complexity*, $\mathcal{BC}(\mathcal{M}(\beta), m)$: the maximum number of bits sent by $\mathcal{M}(\beta)$ in the worst case n -vertex underlying graph and the worst vase m -change scenario.

Next we establish some lower bounds for the message complexity. Throughout, we omit some proofs due to lack of space. See the full paper for detailed proofs.

Lemma 1. *Any exact distance labeling scheme $\pi = (\mathcal{M}, \mathcal{D})$ for the class of trees or for the class of cycles in either of the above dynamic models incurs a message complexity of $\mathcal{MC}(\mathcal{M}, m) = \Omega(mn)$.*

Next we establish a lower bound on the message complexity of β -approximate distance labeling schemes in the fully dynamic model. Let T be a rooted tree and let $\pi(\beta)$ be a β -approximate distance labeling scheme in the fully dynamic model on some graph family containing T . Let $B(e, d)$ be the number of vertices at distance d or less from an endpoint of the edge e . Depicting the tree with the root at the top, let $B_{\text{down}}(e, d)$ be the number of vertices at distance d or less below an endpoint of the edge e . Let $B_{\text{up}}(e, d) = B(e, d) - B_{\text{down}}(e, d)$. Let $\tilde{B}(e, d) = \min\{B_{\text{up}}(e, d), B_{\text{down}}(e, d)\}$ and $\tilde{A} = \max\{\frac{\tilde{B}(e, d)}{d} \mid d \geq 1, e \in E\}$. $\tilde{A}$ is an attempt at capturing the graph-theoretic parameter governing the complexity of the problem. We use the parameter $\tilde{A}$ in our lower bound and a slightly different parameter A in our upper bounds.

Lemma 2. *For constant β , $\mathcal{MC}(\pi(\beta), m) = \Omega(m\tilde{A})$.*

Proof Let e and d be the parameters maximizing $\tilde{A}$. Consider the following scenario. Initially all the edge-weights of T are set to 1. At the first stage, e 's weight, $\omega(e)$, is raised to be $(2d + 1) \cdot \beta^2$. At the second stage, $\omega(e)$ is reduced from $(2d + 1) \cdot \beta^2$ back to 1. These two stages are now executed repeatedly. We claim that at each stage of each two-stage cycle, at least $\tilde{B}(e, d)$ messages must be sent by the marker protocol of the scheme π . This is because otherwise there must exist a pair of vertices, u and v , on different sides of e and both at distance at most d from an endpoint of e , that did not receive any message during that stage. Therefore their labels have not changed from the previous stage, contradicting the fact that these labels must maintain a β -approximate to $d^\omega(u, v)$ at all times. This establishes the lemma. $\square$

3 Dynamic Labeling Schemes for Distance

3.1 Estimating the Distance to the Root

A key ingredient in our schemes concerns a mechanism for allowing each vertex v to estimate its distance from the root of the tree at any given moment. Throughout, we denote the path from a node v to the root by $\mathcal{P}_v$. We introduce two *root-distance* protocols in which each node v keeps a β -approximate of $d^\omega(v)$, v 's weighted distance to the root. The complexity bounds of these protocols are expressed in terms of the following quantities. Let $B(e, d)$ be the number of vertices at distance d or less from an endpoint of the edge e . Let $A = \max\left\{\frac{B(e, d)}{d} \mid d \geq 1, e \in E\right\}$ and set $\alpha = \frac{\beta}{\beta-1}$ and $\gamma = \frac{\sqrt{\beta}}{\sqrt{\beta}-1}$. The first root-distance protocol, $\mathcal{R}_{\text{dyn}}(\beta)$, is applied to the fully dynamic model and satisfies $\mathcal{MC}(\mathcal{R}_{\text{dyn}}(\beta), m) = O(m\alpha A \log^2 n)$. The second root-distance protocol, $\mathcal{R}_{\text{inc}}(\beta)$, is applied to the increasing dynamic model and satisfies $\mathcal{MC}(\mathcal{R}_{\text{inc}}(\beta), m) = O(m\gamma \log^2 n + n \log_\beta m \log n)$.

The root-distance protocol $\mathcal{R}_{dyn}(\beta)$ for the fully dynamic model. Inspired by [2], protocol $\mathcal{R}_{dyn}(\beta)$ is designed so that in the fully dynamic model each node v has a β -approximate to $d^\omega(v)$, v 's weighted distance to the root. The message complexity of the protocol on m weight changes is $\mathcal{MC}(\mathcal{R}_{dyn}(\beta), m) = O(m\alpha\lambda \log^2 n)$.

Each node v maintains two bins, a “local” bin $b_l(v)$ and a “global” bin $b_g(v)$, storing a varying number of tokens throughout the execution. Let $H(v)$ denote the height of v in the tree, namely, its unweighted (hop) distance from the root. The bins of each non-root node v at height $H(v)$, are assigned a *level*, defined as $Level(b_g(v)) = \max\{i \mid 2^i \text{ divides } H(v)\}$ and $Level(b_l(v)) = -1$.

Note that the level of the bin determines whether it is of type b_l or b_g . Therefore, in the following discussion, we omit the subscripts g and l unless it might cause confusion. For each bin b at node v , on any path from v to a leaf, the closest bin b' such that $Level(b') = Level(b) + 1$ is set to be a *supervisor* of b . If for some path (from v to a leaf) there is no such bin then the leaf is set to be a supervisor of b . Note that the supervisors of a local bin are either the global bin of the same node or the global bins of its children. This defines a bin hierarchy.

1. The depth of the bin hierarchy is at most $\log n + 1$.
2. If $Level(b(v)) = l$ then any path from v to a node that holds one of $b(v)$'s supervisors has at most $O(2^l)$ nodes.
3. On any path of length p , the number of level l bins that are supervisors is at most $1 + \frac{p}{2^{l-1}}$.

Let $\sigma = 2^{\lfloor \log_{\alpha(\log n + 1)} \frac{1}{\alpha} \rfloor}$. The number of tokens stored at each bin b at a given time is denoted $\tau(b)$. The tokens can be of either positive or negative type so by saying $\tau(b) = -x$ where x is a positive integer, we mean that b holds x negative tokens. The *capacity* of each bin depends on its level. Specifically, a bin b on $Level(b) = l$ may store $-C(l) \leq \tau(b) \leq C(l)$ tokens, where $C(l) = \max\{\sigma \cdot 2^l, 1\}$. Intuitively a level l bin can be thought of as storing at most $C(l)$ “positive” tokens and at most $C(l)$ “negative” ones. Positive and negative tokens cancel each other out, so at any given moment a bin is either empty or it stores tokens of at most one type. In fact, it will follow from the algorithm description that at any given moment a nonempty bin is half-full, namely, it stores either $C(l)/2$ positive tokens or $C(l)/2$ negative ones, or formally, $\tau(b) \in \{-C(l)/2, 0, C(l)/2\}$.

The protocol $\mathcal{R}_{dyn}(\beta)$.

1. Initially all bins are empty.
2. Each time a node learns that the edge to its parent has increased (respectively, decreased) its weight by one, it adds a $+1$ (resp., -1) token to fill its local bin.
3. Whenever a bin b with $\tau(b) = x$ (positive or negative) tokens gets a signal to add y tokens (where again, y is either positive or negative) it will now be considered having $\tau(b) = x + y$ tokens.

4. Whenever a bin b on level l at a node v gets filled with tokens (i.e., it either has $C(l)$ positive tokens or $C(l)$ negative tokens), v immediately empties the bin and broadcasts a signal to all its supervisor bins to add $C(l)$ (positive or negative) tokens to their bins. This signal message can consist of just l along with the appropriate sign.

In addition, each node v monitors the signals passing through it and estimates $d^\omega(v)$, v 's weighted distance to the root, in the following way. Each node v keeps a counter $\tilde{d}(v)$, initially set to v 's distance to the root in the original tree. When a signal of level l with positive (resp. negative) sign reaches v or passes through it, v adds (resp. subtracts) $C(l)$ to $\tilde{d}(v)$.

For a node v , consider the path $\mathcal{P}_v$ from the root to v . Define $\phi(\mathcal{P}_v)$, the *amount of wasted tokens* on $\mathcal{P}_v$, as the sum of tokens left in the non-empty bins on $\mathcal{P}_v$ counted with their appropriate signs (i.e., with positive and negative tokens canceling each other out). Define $\mu(\mathcal{P}_v)$, the *number of wasted tokens* on $\mathcal{P}_v$, as the total number of (either positive or negative) tokens left in the non-empty bins on $\mathcal{P}_v$. More formally, $\phi(\mathcal{P}_v) = \sum_{b \text{ on } \mathcal{P}_v} \tau(b)$ and $\mu(\mathcal{P}_v) = \sum_{b \text{ on } \mathcal{P}_v} |\tau(b)|$.

Lemma 3. *At any given moment, $\phi(\mathcal{P}_v) = d^\omega(v) - \tilde{d}(v)$.*

Proof Initially the lemma is trivially correct. We use induction on the following events.

1. If an edge on $\mathcal{P}_v$ increases (respectively, decreases) its weight by one and as a result a local bin on $\mathcal{P}_v$ gets full with 1 (resp., -1) token then both $\phi(\mathcal{P}_v)$ and $d^\omega(v)$ increase by 1 (resp., -1). Therefore the equation remains valid.
2. If a message of level l with positive sign (resp., negative) is sent from bin b to bin b' then consider the following.
 - If b and b' are both either in or out of $\mathcal{P}_v$ then none of the parameters of the equation change.
 - If b is in $\mathcal{P}_v$ and b' is out of $\mathcal{P}_v$ then v is on the path from b to b' and the message must pass through v . Therefore $\phi(\mathcal{P}_v)$ decreases (resp., increases) by $C(l)$ and $\tilde{d}(v)$ increases (resp., decreases) by $C(l)$ and the equation remains valid. $\square$

Lemma 4. *For each node v , $\tilde{d}(v)/\beta \leq d^\omega(v) \leq \beta \cdot \tilde{d}(v)$*

Proof Initially all bins are empty. If the capacity of a bin equals 1, the bin always remains empty since it serves only as a relay between the node it supervises and its supervisors (i.e., when a token is added to the bin it gets full and immediately gets empty while sending a message to its supervisor bins). The only bins that might not be empty are global bins $b_g(v)$ that are supervisors with capacity larger than 1. A bin which is not empty is necessarily half-full.

Fix v , denote the length of the path $\mathcal{P}_v$ by p . On $\mathcal{P}_v$, for each level l , there are at most $\frac{p}{2^{l-1}}$ bins at level l that are supervisors. Even if all of them are half-full we get that the total number of wasted tokens in bins of level l is at

most $\frac{p}{2^{l-1}} \cdot \frac{1}{2} \cdot \frac{2^l}{\alpha(\log n+1)} = \frac{p}{\alpha(\log n+1)}$. Therefore, the number of wasted tokens on $\mathcal{P}_v$ (on all levels) satisfies $\mu(\mathcal{P}_v) \leq \frac{p}{\alpha}$. The proof follows since $|d^\omega(v) - \tilde{d}(v)| = |\phi(\mathcal{P}_v)| \leq \mu(\mathcal{P}_v) \leq \frac{p}{\alpha}$ and since both $\tilde{d}(v)$ and $d^\omega(v)$ are always at least p . $\square$

Lemma 5. 1. $\mathcal{MC}(\mathcal{R}_{dyn}(\beta), m) = O(m\alpha\lambda \log^2 n)$
 2. $\mathcal{BC}(\mathcal{R}_{dyn}(\beta), m) = O(m\alpha\lambda \log^2 n \log \log n)$

Proof We say that a bin b' *affects* bin b if by filling b' sufficiently many times, b gets full at least once. Define the *local fillers* of b to be the local bins that affect b . The local fillers of $b(v)$ lie on $\mathcal{P}_v$, the path from the root to v . Moreover, if b is of level l , then all the local fillers of b are at distance $O(2^l)$ from b . Define the *lowest local filler* of b , denoted $LLF(b)$, to be the local filler of b at the lowest height. For fixed l , we now use the LLF function to define a partition of the l -level bins. Let $b_{low} = LLF(b)$ for some level l bin b . Define the *local tree* of b , denoted $LT(b)$, as the smallest subtree of T rooted at b_{low} that contains any level l bin b' such that $b_{low} = LLF(b')$. The following properties are easy to show.

1. All the local fillers of b belong to $LT(b)$.
2. If b' is a level l bin such that $LT(b) \cap LT(b') \neq \emptyset$ then $LT(b') = LT(b)$.
3. The depth of $LT(b)$ is $O(2^l)$.

We now bound the communication caused by the level l bins. Enumerate the local trees $T_1, T_2, \dots$ of the level l bins. Let e_i be an edge adjacent to the root of T_i . Let m_i be the number of weight changes occurring in T_i . Then none of the level l bins that belong to T_i got filled more than $\frac{m_i}{C(l)}$ times. Each time such a bin gets filled it sends a message to all its supervisors. These supervisors are at distance at most $O(2^l)$ below it, which is also at distance $O(2^l)$ from e_i . Hence even if all the level l bins in T_i get filled exactly $\frac{m_i}{C(l)}$ times, the message complexity incurred by them is bounded by $B(e_i, O(2^l)) \cdot \frac{m_i}{C(l)} = B(e_i, O(2^l)) \cdot \frac{\alpha(\log n+1)}{2^l} \cdot m_i = O(\alpha\lambda m_i \log n)$. Therefore, the message complexity caused by level l bins during weight changes is bounded by $O(\alpha\lambda m \log n)$ and the first part of the lemma follows as there are at most $\log n + 1$ levels. The second part of the lemma follows from the first part and from the fact that all messages are of $O(\log \log n)$ bits. $\square$

The root-distance protocol, $\mathcal{R}_{inc}(\beta)$, for the increasing dynamic model. As $\mathcal{R}_{dyn}(\beta)$, protocol $\mathcal{R}_{inc}(\beta)$ is designed so that each node v keeps a β -approximate to $d^\omega(v)$. However, the $\mathcal{R}_{inc}(\beta)$ protocol is designed to work in the increasing dynamic model where at each step an edge weight can only increase by one and indeed achieves better performance. We show that $\mathcal{MC}(\mathcal{R}_{inc}(\beta), m) = O(m\gamma \log^2 n + n \log n \log_\beta m)$.

For a node v , let $T(v)$ be the subtree rooted at v and let t_v be the number of vertices in $T(v)$. We say a child u of v is a *heavy child* of v if $t_u \geq t_w$ for any child w of v . For each non-leaf node v we choose a heavy-child u (breaking ties

arbitrarily) and mark the edge connecting u and v . The non-heavy children of v are referred to as v 's *light children*. The trees hanging down from v 's light children are referred to as v 's *light-subtrees*. The marked edges induce a decomposition of the graph into a collection $\mathcal{S}$ of edge-disjoint paths in the following stages. At the first stage, starting at the root, take into $\mathcal{S}$ the longest path that starts at the root and is composed of only marked edges. At the i 'th stage, take into $\mathcal{S}$ all the longest paths that start with an unmarked edge emanating from some node on a path taken at the $i - 1$ 'st stage and continue over marked edges.

We say a non-root node v *belongs* to a path P if the edge from v 's parent to v belongs to P . Therefore each non-root node v belongs to exactly one path in $\mathcal{S}$; we denote this path by $P(v)$. We denote by $P'(v)$ the subpath of $P(v)$ truncated at v (namely, $P'(v)$ doesn't include any descendent of v). For each path P we denote by $|P|$ the number of edges in P and by $|P|^w$ we denote the weighted length of P .

The decomposition $\mathcal{S}$ has the following property. Each path $\mathcal{P}_v$ from the root to v is decomposed into $k \leq \log n$ edge-disjoint paths $P_1 \dots P_{k-1}$ (where each P_i is prefix of a path in $\mathcal{S}$ and $P_k = P'(v)$). Moreover, all edges in $\mathcal{P}_v$ are marked except maybe the first edges in $P_1, \dots, P_k$. Denote the number of unmarked edges along $\mathcal{P}_v$ by $\eta(v)$. The following claim is used to show that the protocol $\mathcal{R}_{inc}(\beta)$ has only one-side error.

Claim. When applying $\mathcal{R}_{dyn}(\beta)$ in the increasing model we get $\tilde{d}(v) \leq d^\omega(v)$ for each node v .

Proof The claim follows from Lemma 3 and from the fact that in the increasing model $\phi(\mathcal{P}_v) \geq 0$ at any given time. $\square$

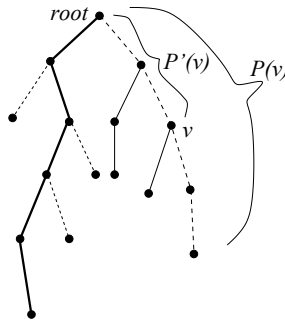


Fig. 1. The thick path, the dashed paths and the regular paths are the paths taken to $\mathcal{S}$ at the first, second and third stages of the decomposition, respectively.

The protocol $\mathcal{R}_{inc}(\beta)$. Each node v simultaneously invokes two protocols. The first, $\mathcal{R}_1$, is the protocol $\mathcal{R}_{dyn}(\sqrt{\beta})$ restricted to the path $P(v)$. The second

protocol, $\mathcal{R}_2$, monitors the behavior of $\tilde{d}(v)$ where $\tilde{d}(v)$ is the approximated weighted distance from v to the root of $P(v)$ maintained by $\mathcal{R}_1$. Each time $\tilde{d}(v)$ increases by a multiplicative factor of $\sqrt{\beta}$, v broadcasts a signal to all the vertices in its light subtrees, containing the number of unmarked edges on the path $\mathcal{P}_v$ from the root to v .

Each node v monitors the $\mathcal{R}_1$ and $\mathcal{R}_2$ signals passing through it and estimates $d^\omega(v)$ in the following way. Decompose the path from the root to v into $P_1, \dots, P_k$ as before. The node v keeps counters $d_1, \dots, d_k$ for approximating $|P_1|^w, \dots, |P_k|^w$ respectively, as follows. For each $1 \leq i \leq k-1$, denote by u_i the bottom node of P_i .

1. Initially $d_i = |P_i|$ for each i .
2. The counter d_k is maintained by $\mathcal{R}_1$.
3. Each time v gets an $\mathcal{R}_2$ signal from u_i , v raises d_i by a multiplicative factor of $\sqrt{\beta}$.
4. At all times, $\tilde{d}(v) = \sum d_i$.

Lemma 6. *At all times, $\tilde{d}(v) \leq d^\omega(v) \leq \beta \cdot \tilde{d}(v)$ and therefore $\mathcal{R}_{inc}$ guarantees that each node v maintains a β -approximation to $d^\omega(v)$.*

Proof It suffices to show that $d_i \leq |P_i^w| \leq \beta \cdot d_i$ for each i . Initially the condition is satisfied since $d_i = |P_i| = |P_i|^w$. Then $d_{u_i} \leq |P_i|^w \leq \sqrt{\beta} \cdot d_{u_i}$ since by Lemma 4, $|P_i|^w \leq \sqrt{\beta} \cdot d_{u_i}$, and by Claim 3.1, $d_{u_i} \leq |P_i|^w$ in the increasing model. Each time this approximation increases by a multiplicative factor of $\sqrt{\beta}$, v will know about it since v belongs to u_i 's light subtrees. Therefore if v gets t $\mathcal{R}_2$ signals from u_i then $d_i = |P_i| \sqrt{\beta}^t \leq |P_i|^w \leq |P_i| \sqrt{\beta}^t \cdot \beta = \beta \cdot d_i$. Together with the fact that d_k is a $\sqrt{\beta}$ -approximate to $|P_k|^w$ as guaranteed by $\mathcal{R}_1$, the lemma follows. $\square$

Lemma 7. 1. $\mathcal{MC}(\mathcal{R}_{inc}(\beta), m) = O(m\gamma \log^2 n + n \log n \log_\beta m)$
 2. $\mathcal{BC}(\mathcal{R}_{inc}(\beta), m) = O(m\gamma \log^2 n \cdot \log \log n + n \log n \log_\beta m \cdot \log \log n)$

Proof Protocol $\mathcal{R}_1$ is applied on disjoint paths and therefore by Lemma 5 applied to each one of the paths, and as $\mathcal{R}_1$ invokes $\mathcal{R}_{dyn}$ with parameter $\sqrt{\beta}$, we get $\mathcal{MC}(\mathcal{R}_1, m) = O(m\gamma \log^2 n)$. We now show that $\mathcal{MC}(\mathcal{R}_2, m) = O(n \log n \log_\beta m)$. For an integer $0 \leq i \leq \lceil \log n \rceil$, let $V_i = \{v \mid \eta(v) = i\}$. For $v \in V_i$, denote by $T_l(v)$ the subtree of $T(v)$ that contains precisely v and its light subtrees. The following observations are trivial.

1. The $\mathcal{R}_2$ communication incurred by v flows only on $T_l(v)$.
2. The number of times v broadcasts a message on $T_l(v)$ is $O(\log_\beta m)$. The reason is that v broadcasts a message on $T_l(v)$ when $\tilde{d}(v)$ increases by a multiplicative factor of $\sqrt{\beta}$ and this can only happen $O(\log_\beta m)$ times in the increasing model.
3. For every v and u in V_i , the trees $T_l(v)$ and $T_l(u)$ are disjoint.

The $\mathcal{R}_2$ message complexity incurred by the nodes of V_i is therefore bounded by $O(n \log_\beta m)$ and as there are at most $\lceil \log n \rceil + 1$ such sets, $\mathcal{MC}(\mathcal{R}_2, m) = O(n \log n \log_\beta m)$ which proves the first part of the lemma. The second part of the lemma follows from the fact that all messages are of $O(\log \log n)$ bits. $\square$

3.2 Dynamic Labeling Schemes for Distance in Trees

throughout this subsection, the underlying topology network is restricted to trees. Given the fully dynamic or the increasing dynamic model, we show how to use a root-distance protocol (specifically, $\mathcal{R}_{dyn}(\beta)$ for the former or $\mathcal{R}_{inc}(\beta)$ for the latter) to give a dynamic labeling scheme for distances.

Definitions. Given a tree T , a *separator* is a vertex v whose removal breaks T into disconnected subtrees of at most $n/2$ vertices each. It is a well known fact that every n -vertex tree T has a separator. As described in [19] one can recursively partition the tree by separators. For convenience, whenever a subtree T' on some level of this recursive partition is split into subtrees $T_1, \dots, T_q$ by removing a separator node v from T' , we formally define each of these subtrees T_i to include v as its root. In the resulting recursive partitioning, each vertex v belongs to a unique subtree $T_l(v)$ on each level l of the hierarchy, up to the level in which v itself is selected as the separator. For a vertex v and a level l of this recursive partitioning, denote by $r_l(v)$ the root of $T_l(v)$, which as explained above is the level l separator that defined $T_l(v)$.

The Marker Protocol $\mathcal{M}(\beta)$. The following discussion applies to both models with $\mathcal{R}$, their appropriate root-distance protocol ($\mathcal{R}_{dyn}(\beta)$ for the fully dynamic model and $\mathcal{R}_{inc}(\beta)$ for the increasing model). Simultaneously, for each level l , the marker protocol $\mathcal{M}(\beta)$ invokes $\mathcal{R}$ separately on each level l subtree. These subtrees are all disjoint and the root of such a tree is the appropriate level l separator. Thus, for each level l , vertex v keeps a β -approximation $\tilde{d}_l(v)$ for $d^\omega(v, r_l(v))$. Fix a vertex v and let $l(v)$ be the level on which v was selected as a separator. The label of v will consist of $l(v)$ pairs, $Label(v) = (\Psi_1(v), \dots, \Psi_{l(v)}(v))$, where the first field of the pair $\Psi_j(v)$ gives the index of $r_l(v)$ and the second field gives $\tilde{d}_l(v)$.

Lemma 8. *Let W be the maximum weight given to an edge. Then $\mathcal{LS}(\mathcal{M}(\beta), m) = O(\log^2 n + \log n \log W)$.*

Proof For each v , $l(v) \leq \log n$ and therefore the number of pairs is at most $\log n$. The first field of each pair is bounded by $\log n$ and the second field is bounded by $\log(nW) = \log n + \log W$. $\square$

As shown in [12], the above label size is optimal for (exact) distance labeling schemes even in the static model.

Lemma 9. 1. *For the fully dynamic model, $\mathcal{MC}(\mathcal{M}(\beta), m) = O(m\alpha\Delta \log^3 n)$ and $\mathcal{BC}(\mathcal{M}(\beta), m) = O(m\alpha\Delta \log^3 n \cdot \log \log n)$*
 2. *For the increasing dynamic model, $\mathcal{MC}(\mathcal{M}(\beta), m) = O(m\gamma \log^3 n + n \log^2 n \log_\beta m)$ and $\mathcal{BC}(\mathcal{M}(\beta), m) = O((m\gamma \log^3 n + n \log^2 n \log_\beta m) \cdot \log \log n)$.*

Proof Given one of the above models, let $\mathcal{R}$ be the root-distance protocol for that model. Fix l and let $T_1, T_2, \dots$ be the level l trees. These trees are disjoint

and $\mathcal{R}$ is performed on each of them separately. Therefore, for fixed l , the message complexity of P on all the level l trees together is bounded by $O(m\alpha\lambda\log^2 n)$ in the fully dynamic model and by $O(m\gamma\log^2 n + n\log n\log_\beta m)$ in the increasing dynamic model. Since there are at most $\log n$ levels and since all messages are of $O(\log \log n)$ bits, the lemma follows. $\square$

The Decoder algorithm $\mathcal{D}(\beta)$. Algorithm $\mathcal{D}(\beta)$ gets as input the labels $Label(u)$ and $Label(v)$ of two vertices u and v . $\mathcal{D}(\beta)$ scans the pairs of $Label(u)$ and $Label(v)$ from right to left looking for the first pairs Ψ_u and Ψ_v with common first field, r . Denote by $\tilde{d}(u, r)$ and $\tilde{d}(v, r)$ the second fields of Ψ_u and Ψ_v respectively. Return $\mathcal{D}(\beta)(Label(u), Label(v)) = \tilde{d}(u, r) + \tilde{d}(v, r)$.

Lemma 10. *The decoder $\mathcal{D}$ yields a β -approximation to the weighted distance.*

3.3 A Different Labeling Scheme $\mathcal{L}_{path-dyn}(\beta)$ for Paths

Consider the fully dynamic model restricted to paths. We show a β -approximation labeling scheme for this model using label size $O(\log n \log m)$ and only $O(m\alpha \log^2 n)$ message complexity. Consider the root-distance protocol $\mathcal{R}_{dyn}(\beta)$ for the fully dynamic model on the path. In this model each node v monitors the messages passing through it and uses them to estimate its distance to the root. This is applied to all of v 's separators and thus causes an overhead of $\log n$ on the message and bit complexities of $\mathcal{M}_{dyn}(\beta)$. The improved labeling scheme, $\mathcal{L}_{path-dyn}(\beta)$, uses protocol $\mathcal{M}_{path-dyn}(\beta)$ which is very similar to protocol $\mathcal{R}_{dyn}(\beta)$, however, it is applied only on the path (and not separately for all the separators). Note that in the protocol $\mathcal{M}_{dyn}(\beta)$ each supervisor bin b supervisors at most two bins b_0 and b_1 . Assume without loss of generality that b_0 is higher than b_1 . Apart from the way the vertices monitor the messages, the only difference between $\mathcal{M}_{path-dyn}(\beta)$ and $\mathcal{R}_{dyn}(\beta)$ is that the messages include an additional bit indicating whether the message was originated at b_0 or b_1 . Since the message and bit complexities of $\mathcal{L}_{path-dyn}(\beta)$ is the same as the complexities of $\mathcal{R}_{dyn}(\beta)$ we get the following.

Lemma 11. $\mathcal{MC}(\mathcal{L}_{path-dyn}(\beta), m) = O(m\alpha \log^2 n)$ and $\mathcal{BC}(\mathcal{L}_{path-dyn}(\beta), m) = O(m\alpha \log^2 n \cdot \log \log n)$

The label structure. Each node v keeps counters $c_i^0(v)$ and $c_i^1(v)$, where $c_i^0(v)$ (respectively, $c_i^1(v)$) is the number of messages passing through v whose destiny is a level i bin and whose origin bin was b_0 (resp., b_1). Let $h(v)$ be the height of v in T and let r be the number of levels.

Let $Label(v) = (h(v), c_1^0(v), c_1^1(v), \dots, c_r^0(v), c_r^1(v))$. Since $r = O(\log n)$ and each c_i^k is of size at most $\log m$ we get the following.

Lemma 12. $\mathcal{LS}(\mathcal{L}_{path-dyn}(\beta), m) = O(\log n \log m)$.

The decoder. Given $Label(v)$ and $Label(u)$ the decoder algorithm estimates the weighted distance between u and v in the following manner. Without loss of generality assume $h(v) > h(u)$. Let $j = \lceil \log(h(v) - h(u)) \rceil$. The decoder now checks whether b_{j+1} , the closest level $j+1$ bin below u , is strictly below v or not. The case where b_{j+1} is below v is denoted by *case 1* and the other case is *case 2*. In case 1, the decoder checks whether there is a bin of level j in the interval $(u, v]$ (there can be at most one). The subcase where there is a bin of level j in the interval $(u, v]$ is denoted by *case 1.1* and the other subcase by *case 1.2*.

For a vertex w and level l let $C_l(w) = \sum_{k=1}^l (c_k^0(w) + c_k^1(w)) \cdot C(k)$. $C_l(w)$ denotes the amount of tokens passing through w destined to a level at most l . Let $h = h(v) - h(u)$.

- For case 2, $\mathcal{D}(Label(u), Label(v)) = C_{j+1}(v) - C_j(u) + h$.
- For case 1.1, $\mathcal{D}(Label(u), Label(v)) = C_j(v) + c_{j+1}^1(v) - C_j(u) + h$.
- For case 1.2, $\mathcal{D}(Label(u), Label(v)) = C_j(v) - C_j(u) + h$.

Lemma 13. *The decoder $\mathcal{D}$ yields a β -approximation for the distance.*

Proof Consider two vertices u and v . Denote by W the number of weight changes that occurred in the subpath between v and u . Our goal is to estimate $W + h$. Denote by s the number of tokens that are stuck at bins in the interval $(u, v]$ and by s^i the number of tokens that are stuck at bins of level at most i in the interval $(u, v]$. It is easy to show that the equation $C_v + s = C_u + W$ holds at all times.

Claim. For $j = \lceil \log(h(v) - h(u)) \rceil$, the equation $\mathcal{D}(Label(u), Label(v)) + s^j = W + h$ holds at all times.

Proof Initially the claim is trivially correct. We use induction on the following events.

1. Assume that the value W changes by one because an edge (w, z) in $(u, v]$ has changed its weight by one. Assume without loss of generality that w is above z . Then the local bin at z gets full and therefore s^j and W get even.
2. Assume that some bin b of level i gets full and a message from that bin travels from b to b' , b' 's supervisor. Consider the following cases.
 - Assume $i \geq j+1$. Then this event doesn't influence any of the equation's parameters.
 - Assume b' is above u or b is below v . Again this event doesn't influence any of the equation's parameters.
 - Assume b is above u and b' is in $(u, v]$. If $i = j$ then again this event doesn't influence any of the equation's parameters. If $i < j$ then s^j and $C_j(u)$ get even.
 - Assume b is above u and b' is below v then we must be in case 1. If $i = j$ then none of the parameters change. If $i < j$ then $C_j(v)$ and $C_j(u)$ get even.

- Assume b and b' are in $(u, v]$. In this case $i < j$ because any message from a level i bin must travel at least a distance of 2^i . Therefore none of the parameters change.
- Assume $i \leq j$, b is in $(u, v]$ and b' is below v then we are not in subcase 1.2. If we are in case 2 then s^j and $C_{j+1}(v)$ get even. If we are in subcase 1.1 then s^j and $C_{j+1}^1(v)$ get even. $\square$

Since $s^j \leq h/\beta$, the lemma follows. $\square$

4 Dynamic Labeling Schemes for the Fully Dynamic Model on Cycles

We modify an approximate labeling scheme for the class of paths (namely, $\mathcal{L}_{path-dyn}$ or $\mathcal{L}_{dyn}$) to get a β -approximate labeling scheme for the fully dynamic model with the underlying topology restricted to cycles using the same asymptotic complexities as for paths. We give an intuitive review of the modified scheme, a detailed description will appear in the full paper.

Two vertices are pivots if they divide the weighted cycle into two arcs so that up to some small constant factor, the shortest way between two vertices on the same arc is on that arc (and not by going through the other arc). Our marker protocol constantly holds two vertices as pivots. The pivots simultaneously invoke the desired path-scheme for the clockwise paths from each pivot to itself. The label of a vertex is a concatenation of its labels in these two schemes. The decoder protocol uses the decoder protocol of the two schemes (on the appropriate labels) and outputs the smaller value.

We therefore get the following lemma for constant β .

Lemma 14. *By using $\mathcal{L}_{dyn}$ (resp., $\mathcal{L}_{path-dyn}$) we get a β -approximate distance labeling scheme for the fully dynamic model on cycles with the same asymptotic complexities of $\mathcal{L}_{dyn}$ (resp., $\mathcal{L}_{path-dyn}$) for paths.*

References

1. S. Abiteboul, H. Kaplan and T. Milo. Compact labeling schemes for ancestor queries. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2001.
2. Y. Afek, B. Awerbuch, S.A. Plotkin and M. Saks. Local management of a global resource in a communication. *J. of the ACM*, pages 1–19, 1989.
3. S. Alstrup, C. Gavoille, H. Kaplan and T. Rauhe. Identifying nearest common ancestors in a distributed environment. IT-C Technical Report 2001-6, The IT University, Copenhagen, Denmark, Aug. 2001.
4. S. Alstrup and T. Rauhe. Improved Labeling Scheme for Ancestor Queries. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.
5. M.A. Breuer and J. Folkman. An unexpected result on coding the vertices of a graph. *J. of Mathematical Analysis and Applications*, 20:583–600, 1967.
6. M.A. Breuer. Coding the vertexes of a graph. *IEEE Trans. on Information Theory*, IT-12:148–153, 1966.

7. E. Cohen, E. Halperin, H. Kaplan and U. Zwick. Reachability and Distance Queries via 2-hop Labels. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.
8. P. Fraigniaud and C. Gavoille. Routing in trees. In *Proc. 28th Int. Colloq. on Automata, Languages & Prog.*, LNCS 2076, pages 757–772, July 2001.
9. C. Gavoille and C. Paul. Split decomposition and distance labelling: an optimal scheme for distance hereditary graphs. In *Proc. European Conf. on Combinatorics, Graph Theory and Applications*, Sept. 2001.
10. C. Gavoille and D. Peleg. Compact and Localized Distributed Data Structures. Research Report RR-1261-01, LaBRI, Univ. of Bordeaux, France, Aug. 2001.
11. C. Gavoille, M. Katz, N.A. Katz, C. Paul and D. Peleg. Approximate Distance Labeling Schemes. In *9th European Symp. on Algorithms*, Aug. 2001, Aarhus, Denmark, SV-LNCS 2161, 476–488.
12. C. Gavoille, D. Peleg, S. Pérennes and R. Raz. Distance labeling in graphs. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, pages 210–219, Jan. 2001.
13. S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 334–343, May 1988.
14. H. Kaplan and T. Milo. Short and simple labels for small distances and other functions. In *Workshop on Algorithms and Data Structures*, Aug. 2001.
15. H. Kaplan, T. Milo and R. Shabo. A Comparison of Labeling Schemes for Ancestor Queries. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.
16. M. Katz, N.A. Katz, A. Korman and D. Peleg. Labeling schemes for flow and connectivity. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.
17. M. Katz, N.A. Katz, and D. Peleg. Distance labeling schemes for well-separated graph classes. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science*, pages 516–528, February 2000.
18. A. Korman, D. Peleg, and Y. Rodeh. Labeling schemes for dynamic tree networks. In *Proc. 19th STACS Symp. on Theoretical Aspects of Computer Science*, March. 2002.
19. D. Peleg. Proximity-preserving labeling schemes and their applications. In *Proc. 25th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 30–41, June 1999.
20. D. Peleg. Informative labeling schemes for graphs. In *Proc. 25th Symp. on Mathematical Foundations of Computer Science*, volume LNCS-1893, pages 579–588. Springer-Verlag, Aug. 2000.
21. M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science*, Oct. 2001.
22. M. Thorup and U. Zwick. Compact routing schemes. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architecture*, pages 1–10, Hersonissos, Crete, Greece, July 2001.

A Simple Linear Time Algorithm for Computing a $(2k - 1)$ -Spanner of $O(n^{1+1/k})$ Size in Weighted Graphs

Surender Baswana* and Sandeep Sen**

Department of Computer Science and Engineering,
I.I.T. Delhi, Hauz Khas, New Delhi-110016, India.
{sbaswana, ssen}@cse.iitd.ernet.in

Abstract. Let $G(V, E)$ be an undirected *weighted* graph with $|V| = n$, and $|E| = m$. A t -spanner of the graph $G(V, E)$ is a sub-graph $G(V, E_S)$ such that the distance between any pair of vertices in the spanner is at most t times the distance between the two in the given graph. A 1963 girth conjecture of Erdős implies that $\Omega(n^{1+1/k})$ edges are required in the worst case for any $(2k - 1)$ -spanner, which has been proved for $k = 1, 2, 3, 5$. There exist polynomial time algorithms that can construct spanners with the size that matches this conjectured lower bound, and the best known algorithm takes $O(mn^{1/k})$ expected running time. In this paper, we present an extremely simple linear time randomized algorithm that constructs $(2k - 1)$ -spanner of size matching the conjectured lower bound.

Our algorithm requires local information for computing a spanner, and thus can be adapted suitably to obtain efficient distributed and parallel algorithms.

Keywords: Graph algorithms, Randomized algorithms, Shortest path

1 Introduction

A spanner is a (sparse) sub-graph of a given graph that preserves approximate distance between all-pairs of vertices. In precise words, a sub-graph $G(V, E_S)$ is said to be a t -spanner of the graph $G(V, E)$ if, between any pair of vertices, the distance in the spanner is at-most t times the distance in the original graph. The value t is the *stretch factor* associated with the spanner.

The concept of a sparse spanner is motivated by numerous applications that involve computation of distances in a graph. Since the running time is proportional to the number of edges, therefore to achieve efficiency in computation time it is desired to have a sub-graph (of a given dense graph) that is sparse, but at the same time, preserves all-pairs distances approximately.

Spanners are used as underlying graph structures in various areas of distributed computing; e.g. the design of synchronizers [2] and design of succinct

* Work was supported in part by a fellowship from Infosys Technologies, Bangalore.

** Work was supported in part by an IBM Faculty Partnership award

routing tables [9] implicitly generate spanners. Spanners are also used in computational biology [3] in the process of reconstructing phylogeny trees from matrices whose entries represent genetic distances among contemporary living species. For other numerous applications, please refer to the papers [1,2,9,10].

1.1 Previous Work

Previously a number of papers [1,6,11] had addressed the problem of computing sparse spanners of graphs efficiently. In addition, a lot of work [9,11] had also been done to establish a lower bound on the size of spanner in terms of stretch factor. To establish these bounds on the size of the spanner, these results use the following relationship between the stretch of a spanner and girth of the graph.

A graph has girth at-least $t + 2$ if and only if it does not have a t -spanner other than the graph itself.

A classical result from graph theory shows that every graph with $\Omega(n^{1+1/k})$ edges must have a cycle of size at most $2k$. It has been conjectured by Erdős [7], Bondy and Simonovits [5], and Bollobás [4] that this bound is indeed tight. Namely, for any $k \geq 1$, there are graphs with $\Omega(n^{1+1/k})$ edges that have girth greater than $2k$. However, the proof exists only for the cases $k = 1, 2, 3$ and 5 . Since any graph contains a bipartite sub-graph with at least half the edges, the conjecture implies the existence of graphs with $\Omega(n^{1+1/k})$ edges and girth at-least $2k + 2$. This bound and the relation between the stretch of a spanner and girth of given graph (mentioned above) imply a lower bound of $O(n^{1+1/k})$ on the size of $(2k - 1)$ -spanner.

For unweighted graphs, Halperin and Zwick [8,12] gave an $O(m)$ time algorithm to construct a $(2k - 1)$ -spanner of size $O(n^{1+1/k})$. However, their algorithm does not seem to be extensible to weighted graphs. For weighted graphs, the first algorithm for constructing $(2k - 1)$ -spanners of $O(n^{1+1/k})$ size was given by Althöfer et al. [1]. However, the best known implementation of their algorithm has a running time of $O(mn^{1+1/k})$. Thorup and Zwick [11], improving a result of Cohen [6], gave a randomized algorithm for computing $(2k - 1)$ -spanner of optimal size in $O(kmn^{1/k})$ expected time. All the existing algorithms require computation of shortest distance information between many pairs of vertices [1], or computing shortest path trees from a set of $(n^{1/k})$ vertices [11]. Since there is a bound of $\tilde{O}(m)$ on the best known algorithm for shortest path trees, the running times of the earlier algorithms may not be able to achieve a bound of $O(m)$.

1.2 Our Contribution

We present a randomized algorithm that takes expected linear time for computing $(2k - 1)$ -spanner of optimal size. More specifically we show that

Given a weighted graph $G(V, E)$, and integer $k > 1$, a spanner of stretch $(2k - 1)$ and $O(kn^{1+1/k})$ size can be computed in expected time $O(km)$.

Unlike previous algorithms that require global information (computing full shortest path tree from few vertices or determining the pair-wise distance), our algorithm requires only local information, viz. in the neighborhood of each vertex or a group of vertices. In addition to achieving a linear time sequential static algorithm for computing spanner, the local approach of our algorithm leads to near optimal algorithms for computing a $(2k-1)$ -spanner in distributed/parallel environment and in external memory as well. The algorithm can be suitably adapted to provide efficient partial dynamic algorithms for maintenance of spanners for small k as well.

We have organized the paper as follows. As a warm-up, we first present an $O(m)$ expected time algorithm for 3-spanner, and mention some of the key ideas (clustering of vertices) which we formalize and extend in order to compute a $(2k-1)$ -spanner. We present an overview of the algorithm followed by the details and proofs of correctness of the algorithm. The details of the other applications are not given in this extended abstract.

2 Computing a 3-Spanner

In order to build 3-spanner of a graph (with potentially $\theta(n^2)$ edges), the objective is to minimize the number of edges (at most $O(n^{3/2})$) to be included in the spanner, and still ensure that the distance between any pair of vertices is not stretched beyond a factor of 3.

The algorithm selects edges to be included in the spanner in two phases. Without loss of generality, we can assume that the edge weights are distinct.

1. *Forming the clusters* :

We form a sample $\mathcal{R} \subset V$ by picking each vertex independently with probability $n^{-\frac{1}{2}}$. The expected size of the sample set is $O(\sqrt{n})$. We group the set of vertices neighboring to these sampled vertices into clusters. Initially the clusters are $\{\{u\} | u \in \mathcal{R}\}$. Each $u \in \mathcal{R}$ will be referred to as the *center* of its cluster. We process a vertex $v \in V - \mathcal{R}$ as follows.

- If v is not adjacent to any sampled vertex, we add all its edges to the spanner.
- If v is adjacent to one or more sampled vertices, let $\mathcal{N}(v, \mathcal{R})$ be the sampled neighbor that is nearest to v . We add the edge $e(v, \mathcal{N}(v, \mathcal{R}))$ to the spanner, and every other edge incident on v with weight less than the weight of the edge $e(v, \mathcal{N}(v, \mathcal{R}))$ to the spanner. The vertex v is added to the cluster centered at $\mathcal{N}(v, \mathcal{R})$.

Finally, all the intra-cluster edges, i.e., the edges between the vertices belonging to the same cluster are removed.

2. *Joining the Clusters*:

For each vertex v , we group all its neighbors into their respective clusters. There will be at-most $|\mathcal{R}|$ neighboring clusters of v . For each cluster adjacent to v , we add the least-weight edge among all the edge between (the vertices of) the cluster and v to the spanner.

It is easy to see that the above algorithm merely requires exploring the adjacency list of each vertex at-most twice (once in each of the two phases) in

addition to picking a random sample of vertices. Thus the running time of the algorithm is $O(m)$. Let E_S^1 and E_S^2 be the sets of edges added to the spanner in the first phase and the second phase respectively.

From the description of the first phase of the algorithm, the following Lemma holds true.

Lemma 1. *For each edge $e(u, v) \in E - E_S^1$, the edge from u to $\mathcal{N}(u, \mathcal{R})$ (the center of the cluster to which u belongs) has weight no more than the weight of the edge $e(u, v)$.*

We shall use the following Lemma to show that the set $E_S^1 \cup E_S^2$ is a 3-spanner.

Lemma 2. *For an edge $e(u, v) \in E$ that is not present in the spanner $G(V, (E_S^1 \cup E_S^2))$ constructed above, the following assertion holds true:*

There is a path in the spanner with weight at-most three times the weight of the edge $e(u, v)$.

Proof. It follows from the first phase of the algorithm described above that both u and v must be adjacent to vertices of the sample $\mathcal{R}$. There are two cases now.

Case 1 : Both the vertices belong to same cluster, say the cluster centered at $w \in \mathcal{R}$. It follows from Lemma 1 that there is a 2-edge long path $u - w - v$ in the spanner with weight no more than twice the weight of the edge (u, v) . (This provides a justification for deleting any intra-cluster edge from the set E at the end of first phase)

Case 2 : The vertices u and v belong to different clusters, and let u belongs to the cluster centered at $x \in \mathcal{R}$. Let $e(u', v)$ be the least weight edge $\in E - E_S^1$ among all the edges incident on v from the vertices of the cluster centered at x . It follows from the second phase of our algorithm, that the edge $e(u', v)$ was added to the spanner. Hence there is a path $u - x - u' - v$ from u to v in the spanner and its weight $\mathbf{w}_S$ can be bounded as follows.

$$\begin{aligned} \mathbf{w}_S(u, v) &= \mathbf{w}(u, x) + \mathbf{w}(x, u') + \mathbf{w}(u', v) \\ &\leq \mathbf{w}(u, v) + \mathbf{w}(x, u') + \mathbf{w}(u', v) \quad \{\text{using Lemma 1}\} \\ &\leq \mathbf{w}(u, v) + 2\mathbf{w}(u', v) \quad \{\text{using Lemma 1}\} \\ &\leq 3\mathbf{w}(u, v) \quad \{\text{follows from the second phase of the algorithm}\} \end{aligned}$$

Using the above Lemma, we can show that the spanner $G(V, E_S^1 \cup E_S^2)$ has stretch 3 as follows.

Consider any pair of vertices $u, v \in V$, and the shortest path p_{uv} between the two in the graph $G(V, E)$. For each edge e of this path that is missing in the spanner $G(V, E_S^1 \cup E_S^2)$, there is a path in the spanner with weight at-most thrice the weight of the edge e (using Lemma 2). Applying this argument for each missing edge, we can conclude that there is a path between the vertex u and the vertex v in the spanner with weight at-most thrice the weight of the shortest path between the two in the graph $G(V, E)$.

Now we shall show that the size of the 3-spanner $E_S^1 \cup E_S^2$ as computed by our algorithm given above is bounded by $O(n^{3/2})$.

Note that the sample set $\mathcal{R}$ is formed by picking each vertex randomly independently with probability $\frac{1}{\sqrt{n}}$. It thus follows from elementary probability

that for each vertex $v \in V$, the expected number of incident edges with weight less than that of $e(v, \mathcal{N}(v, \mathcal{R}))$ is $\sqrt{n}$. Thus the expected number of edges contributed to the spanner by each vertex in the first phase of the algorithm is $\sqrt{n}$. So the expected size of the set E_S^1 is $O(n^{3/2})$.

Remark Since we can verify the number of edges chosen in the first phase, we will repeat the sampling if the number of edges exceeds $O(n^{3/2})$ - the expected number of repetitions will be $O(1)$.

The number of the edges added to the spanner in the second phase (*Joining the Clusters*) is $O(n|\mathcal{R}|)$. Since the expected size of the sample $\mathcal{R}$ is $\sqrt{n}$, therefore, the expected size of E_S^2 is also $O(n^{3/2})$. Hence the expected size of the spanner $E_S^1 \cup E_S^2$ is $O(n^{3/2})$.

Thus we can conclude that 3-spanner of a weighted undirected graph can be computed in $O(m)$ expected time.

3 Key Ideas Underlying the Algorithm

The algorithm for computing a $(2k - 1)$ -spanner selects a set E_S of $O(kn^{1+1/k})$ edges from the given graph $G(V, E)$ ensuring that the following proposition holds true for each edge $e \notin E_S$.

$\mathcal{P}_k(e)$: there is a path between the end-points of the edge e in the graph $G(V, E_S)$ consisting of at-most $(2k - 1)$ edges, and the weight of each edge on this path is no more than that of the edge e

We say that $\mathcal{P}_k(v)$ holds for a vertex if $\mathcal{P}_k(e)$ holds true for each edge e incident on the vertex v . It follows from the discussion at the end of the previous section that the spanner formed in this way will be of stretch $(2k - 1)$.

In order to pick a (small) set E_S of $O(n^{1+1/k})$ edges (from potentially $O(n^2)$ edges in a graph), that would ensure the proposition $\mathcal{P}_k(e)$ for each edge $e \in E - E_S$, the key idea underlying our algorithm is the partitioning of set of vertices into *clusters*. Recall from the previous section how the clustering of the vertices (by sampling a random set of vertices, and grouping each vertex with its nearest sampled neighbor) proves to be crucial in the computation of a 3-spanner with $O(n^{3/2})$ edges. It is the smaller number of these clusters (only $\sqrt{n}$) compared to the number of vertices (n) that enables us to get a bound on the size of the 3-spanner, and it is the closeness of the vertices within a cluster that ensured a bound on the stretch of the spanner. Note that the latter property (closeness of vertices of same cluster) is achieved by associating each vertex to its nearest sampled vertex in the clustering.

In order to design an algorithm for computing $(2k - 1)$ -spanner, we formally define the *clustering* of vertices, and associate a parameter called *radius of a cluster* that captures the closeness of the vertices of the same cluster compared to the vertices outside the cluster.

3.1 Definitions and Notations

The following definitions and notations are in context of a given weighted graph $G(V, E)$.

Definition 1. A **cluster** is a subset of vertices. A partition of a set $V' \subset V$ into clusters is called **clustering**.

Definition 2. A set $S \subset E$ is a **spanning set** for a cluster $c \subset V$ if each pair of vertices of c is connected by a path that is internal to the cluster (i.e., the intermediate vertices of the path belong to the cluster c) and consists of edges from the set E_S only.

Definition 3. A cluster $c \subset V'$ with a spanning set $E_S \subset E'$ is a cluster of **radius** $\leq i$ in the graph $G(V', E')$, if there is a vertex $u \in c$ called the center of the cluster c such that the following holds true :

For each edge $e(x, y) \in E' - E_S, x \in c$, there is a path from x to u internal to the cluster and consisting of at-most i edges each from the set E_S and having weight no more than the weight of the edge $e(x, y)$.

Intuitively the vertices of a cluster are close together compared to the vertices lying outside the cluster.

Definition 4. A clustering $\mathcal{C}$ with a spanning set $E_S \subset E$ is a clustering of radius $\leq i$ if each of its cluster with spanning set E_S is a cluster of radius $\leq i$.

We shall use the following notations in the rest of our paper.

- $\mathcal{E}(x, c_1)$: the edges from the set $\mathcal{E}$ that are incident from the vertices of cluster c_1 to the vertex x .
- $\mathcal{E}(c_1, c_2)$: the set of edges between vertices of cluster c_1 and vertices of cluster c_2 that belong to the set $\mathcal{E}$.
- $\mathcal{E}(\mathcal{S})$: the set of edges from set $\mathcal{E}$ between the vertices of the clusters of the set $\mathcal{S}$.
- $|\mathcal{C}|$: The number of clusters in the clustering $\mathcal{C}$ (also referred as the *size* of the clustering).

Our algorithm exploits the properties of a clustering of bounded radius as mentioned in the following two Lemmas.

Lemma 3. For a given graph $G(V', E' \cup E_S)$, let $\mathcal{C}$ be a clustering with E_S as its spanning set, and let $c \in \mathcal{C}$ be a cluster having radius at-most i . For a vertex $u \notin c$, adding the least weight edge from the set $E'(u, c)$ to the spanning set E_S will ensure that the proposition $\mathcal{P}_{i+1}(e)$ holds for the entire set $E'(u, c)$.

Proof. Let the edge $e(u, y)$ of weight α be the least-weight edge from the set $\mathcal{E}(u, c)$. Let (u, x) be any other edge of weight $\beta \geq \alpha$ from the set $\mathcal{E}(u, c)$ (see the Figure 1). Since the radius of the cluster c is at-most i , therefore, there is a path p_{xv} from x to the center v (of the cluster c) consisting of edges from the set E_S only, and its weight is at-most i times β . Using the same argument, we deduce that there is a path p_{yv} from vertex y to v consisting of edges from the set E_S , and with weight at-most $i\alpha$. Thus there is path p_{ux} from vertex u to vertex x formed by concatenating the edge (u, y) and the paths p_{yv}, p_{vx} in this order; and its weight can be bounded as follows.

$$\begin{aligned}
 \mathbf{w}(p_{ux}) &= \mathbf{w}(u, y) + \mathbf{w}(p_{yv}) + \mathbf{w}(p_{vx}) \\
 &\leq \alpha + i\alpha + i\beta \leq \beta + i\beta + i\beta \quad \{ \text{since } \alpha \leq \beta \} \\
 &= (2i + 1)\beta = (2(i + 1) - 1)\beta
 \end{aligned}$$

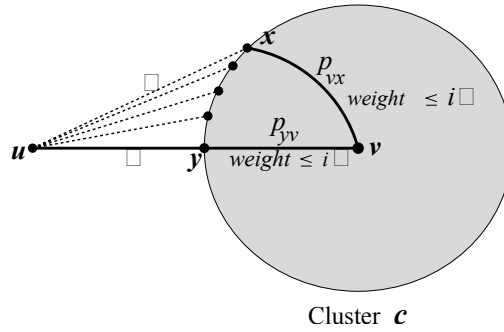


Fig. 1. Ensuring that the proposition $\mathcal{P}_{i+1}$ holds true for the set $E'(u, c)$.

Therefore, we can conclude that adding the edge $e(u, y)$ to the spanning set makes the proposition $\mathcal{P}_{i+1}(e)$ hold true for each edge $e \in \mathcal{E}'(u, c)$.

Along similar lines we can prove the following Lemma.

Lemma 4. *For a given graph $G(V', E' \cup E_S)$, let $\mathcal{C}$ be clustering with E_S as a spanning set, and let $c_1, c_2 \in \mathcal{C}$ be two clusters having radius i and j respectively. Adding the least weight edge of the set $E'(c_1, c_2)$ to the spanning set E_S will ensure that the proposition $\mathcal{P}_k, k = i + j + 1$ holds true for the entire set $E'(c_1, c_2)$.*

4 Algorithm for Computing a $(2k - 1)$ -Spanner

4.1 An Overview

Based on the key observations of a clustering of finite radius mentioned in the Lemmas 3 and 4, the algorithm for computing a $(2k - 1)$ -spanner of a weighted graph $G(V, E)$ selects edges to be included in the spanner in two phases.

The first phase *Forming the Clusters* starts with a clustering $\{\{v\} | v \in V\}$ (of n clusters but zero radius), and executes $\lfloor \frac{k}{2} \rfloor$ iterations. In i th iteration, a set of edges are selected to be added to the spanner that makes the proposition $\mathcal{P}_{i+1}$ true for a (possibly large) set of edge and vertices (using Lemma 3). All these edges (and vertices) are removed from the graph. A clustering is obtained again for the remaining vertices. In successive iterations, the size of the clustering (the number of clusters) reduces geometrically while the radius of clusters increase by just one unit (see Figure 2).

At the end of $\lfloor \frac{k}{2} \rfloor$ iterations, we obtain a clustering of the rest of the vertices (for whom $\mathcal{P}_{\frac{k}{2}+1}$ does not hold) that consists of only $n^{1-\frac{1}{k}\lfloor \frac{k}{2} \rfloor}$ clusters. Moreover, a subset of spanner-edges added till $\lfloor \frac{k}{2} \rfloor$ th iterations spans this clustering, and ensures that the radius of each cluster is no more than $\lfloor \frac{k}{2} \rfloor$. This clustering consisting of very few clusters and *not-so-large* radius is passed onto the second phase of the algorithm.

In the second phase *Joining the Clusters*, the clusters are joined together by adding the least weight edge between each pair of neighboring clusters. This

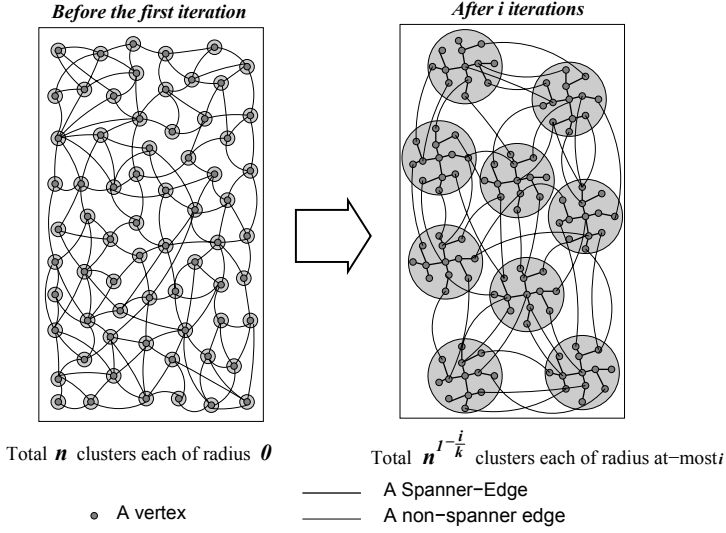


Fig. 2.

phase employs Lemma 4 to ensure that $\mathcal{P}_k$ holds true for all the vertices left in the graph after the first phase.

4.2 Details of the Algorithm

We describe the details of the two phases of our algorithm for computing a $(2k - 1)$ -spanner of a weighted graph $G(V, E)$ as follows.

Phase 1 : Forming the clusters

This phase executes in $\lfloor \frac{k}{2} \rfloor$ iterations. At each stage in this phase, E_S^1 denotes the set (initially $\emptyset$) of edges that are added to the spanner, and E' denotes the set of edges remaining in the graph.

i th iteration begins with a clustering $\mathcal{C}_i$ of the set V'_i of vertices defined by the endpoints of E' after $(i - 1)$ th iteration.

For the first iteration, the sets are $E_S^1 = \emptyset$, $E' = E$, $V'_1 = V$, and the clustering is $\mathcal{C}_1 = \{\{v\} | v \in V\}$.

We describe the processing done in an iteration as follows.

In i th iteration, a sample $\mathcal{R}_i$ of clusters is formed by picking each cluster from the clustering $\mathcal{C}_i$ randomly independently with probability $n^{-\frac{1}{k}}$. The vertices belonging to the sampled clusters are added to the set V'_{i+1} , and passed onto the next iteration as it is. Each of the remaining vertices of the set V'_i are processed according to the following two cases.

- If v is not adjacent to any sampled cluster, for each cluster $c \in \mathcal{C}_i$ adjacent to v , we add the least weight edge from the set $E'(v, c)$ to the spanner. We remove all the edges incident on v from the graph.

- If vertex v is adjacent to one or more sampled cluster, let $c \in \mathcal{R}_i$ be the cluster that is adjacent to v with edge of least weight (say w) among all the clusters from the set $\mathcal{R}_i$. We add the least weight edge from the set $E'(v, c)$ to the spanner, and remove the entire set $E'(v, c)$ from the graph. In addition, we do the following. For each cluster $c' \in \mathcal{C}_i$ adjacent to vertex v with an edge of weight $< w$, we add the least weight edge from the set $E'(v, c')$ to the spanner, and remove the entire set $E'(v, c')$ of edges from the graph. (It can be seen that all the edges from set E' that remain incident on the vertex v with weight $< w$ are removed in i th iteration). For the $(i + 1)$ th iteration, the vertex v is added to the set V_{i+1} .

For the $(i + 1)$ th iteration, the set V'_{i+1} thus consists of only those vertices from the set V'_i which either belong to or are adjacent to some cluster from the sample $\mathcal{R}_i$. The clustering of $\mathcal{C}_{i+1}$ for these vertices is obtained as follows. After initializing $\mathcal{C}_{i+1}$ to $\mathcal{R}_i$, each vertex $v \in V_{i+1}$ not belonging to any cluster from sample $\mathcal{R}_i$ is added to the cluster from $\mathcal{R}_i$ that is incident with the least weight edge to v . Thus a cluster $c \in \mathcal{C}_{i+1}$ can be viewed as a union of a cluster from the sample $\mathcal{R}_i$ with the set of all those vertices from V'_i for whom c was the sampled neighboring cluster incident with the least weight edge. As a last step of i th iteration, we eliminate all the intra-cluster edges (whose both end-points belong to the same cluster) of the clustering $\mathcal{C}_{i+1}$ from the graph (i.e., the set E').

Theorem 1. *The following assertion holds for each $j \geq 1$.*

$\mathcal{A}(j)$: *For each cluster $c \in \mathcal{C}_j$, there exist a subset $\mathcal{E}$ of edges added to the spanner by the end of $(j - 1)$ th iteration such that c with $\mathcal{E}$ as its spanning set is a cluster of radius $\leq (j - 1)$ in the graph $G(V'_j, E' \cup \mathcal{E})$.*

Proof. We shall prove the theorem by induction $j \geq 1$.

Base Case : $j = 1$: In the beginning of the algorithm, the clustering is $\mathcal{C}_1 = \{\{v\} | v \in V\}$, $E_S = \emptyset$, $V'_1 = V$, $E' = E$. It is easy to observe that each cluster of $\mathcal{C}_1$ is a cluster of radius 0 in the graph $G(V, E)$. Therefore, the assertion $\mathcal{A}(1)$ holds.

Induction Hypothesis : $j \leq i$: Let the assertion $\mathcal{A}(i)$ holds.

Proof of assertion $\mathcal{A}(i + 1)$:

As mentioned in the first phase of the algorithm, a cluster from the clustering $\mathcal{C}_{i+1}$ is a union $R \cup \mathcal{N}_R$ of a cluster $R \in \mathcal{R}_i$ with the set $\mathcal{N}_R$ of vertices $v \in V_i$ for whom the cluster R is the sampled cluster incident with the least weight edge among all adjacent clusters that belong to the sample $\mathcal{R}_i$. For each vertex $v \in \mathcal{N}_R$, we add to the spanner the edge (say, $e(v, u)$, $u \in R$) of least weight from the set $E'(v, R)$ in the i th iteration.

It follows from the induction hypothesis that there exists a subset $\mathcal{E}$ of edges added to the spanner by the end of $(i - 1)$ th iteration such that, with $\mathcal{E}$ as its spanning set, R is a cluster of radius $(i - 1)$ in the graph $G(V'_i, E' \cup \mathcal{E})$. From the definition of cluster-radius, it implies that there is a vertex $r \in R$ called the center of the cluster R such that the following holds true :

For each edge $e(v, x) \in E'$, $x \in R$, there is a path from x to the vertex r , that is internal to the cluster R and consists of at-most $(i - 1)$ edges from the set $\mathcal{E}$ only; and each edge on the path has weight no more than that of $e(v, x)$.

Since the edge $e(v, u)$ belongs to E' at the end of $(i - 1)$ th iteration, and is added to the spanner in i th iteration, so there is a path from the vertex v to the vertex r consisting of at-most i edges, each belonging to the spanner and having weight no more than that of the edge $e(v, u)$. Also it follows from the processing of the vertex v in i th iteration that there is no edge incident on v from the set E' whose weight is less than that of $e(v, u)$. Thus we can conclude that for each edge $e(v, y) \in E', v \in R \cup \mathcal{N}_R$ at the end of i th iteration, there exists a subset $\mathcal{E}'$ of edges added to the spanner by the end of i th iteration so that the following holds true :

there is a path from v to the vertex $r \in R \cup \mathcal{N}_R$ (center of the cluster $R \cup \mathcal{N}_R$) that is internal to the cluster $R \cup \mathcal{N}_R$, and consists of at-most i edges, each from the set $\mathcal{E}'$, and weight of each edge being no more than that of $e(v, y)$.

From definition of the cluster-radius, it follows that the cluster $R \cup \mathcal{N}_R$ with spanning set $\mathcal{E}'$ is a cluster of radius $\leq i$ in the graph $G(V'_{i+1}, E' \cup \mathcal{E}')$.

Similar arguments can be given for any other cluster in the clustering $\mathcal{C}_{i+1}$, i.e., for each cluster $c \in \mathcal{C}_{i+1}$, there is a subset $\mathcal{E}_c$ of edges added to the spanner by the end of i th iteration such that c with $\mathcal{E}_c$ as its spanning set is a cluster of radius $\leq i$ in the graph $G(V'_{i+1}, E' \cup \mathcal{E}_c)$.

Thus the assertion $\mathcal{A}_{i+1}$ holds. Hence by the principle of mathematical induction, the assertion $\mathcal{A}_j$ holds for all $j \geq 1$.

Using Lemma 3 and the Theorem given above, we can state the following corollary.

Corollary 1. *For each edge $e \in E'$ eliminated from the graph in i th iteration, the proposition $\mathcal{P}_{i+1}$ holds true.*

Since there are $\lfloor \frac{k}{2} \rfloor$ iterations in the first phase, it follows from the corollary stated above that $\mathcal{P}_k$ holds for each edge eliminated from the graph in the first phase.

We shall now bound the expected number of edges added to the spanner in the first phase.

Lemma 5. *The expected number of edges added to the spanner in each iteration of the first phase of our algorithm is $n^{1+1/k}$.*

Proof. Consider i th iteration, and a vertex $v \in V'_i$. All the neighbors of the vertex v are grouped into their respective clusters of the clustering $\mathcal{C}_i$. Let $c_1, c_2, \dots, c_l$ be the clusters adjacent to v , and arranged in the increasing order of the weight of their least-weight edge incident on v , i.e., the least weight edge from a set $E'(v, c_j)$ is lighter (has less weight) than the least weight edge from the set $E'(v, c_{j+1})$ for all $j < l$.

It follows from the algorithm that for the cluster c_j adjacent to v , we add just one edge (the least weight edge) from the set $E'(v, c_j)$ to the spanner, if none of the clusters preceding it, i.e., $c_1, \dots, c_{j-1}$ are sampled. Since each cluster is sampled independently with probability $n^{-1/k}$, the probability that we add an edge from $E'(v, c_j)$ to the spanner is no more than $(1 - n^{-1/k})^{j-1}$. Thus the expected number of edges contributed to the spanner by a vertex $v \in V_i$ is given by

$$\sum_{j=1}^{j=l} \left(1 - n^{-1/k}\right)^{j-1} \leq \frac{1}{n^{-1/k}} = n^{1/k}$$

Thus the expected number of edges added to the spanner in i th iteration is bounded by $n^{1+1/k}$.

We repeat an iteration if the number of edges exceed $O(n^{1+1/k})$ - the expected number of repetitions is $O(1)$. There are total $\lfloor \frac{k}{2} \rfloor$ iterations, so the expected number of edges added to the spanner in the first phase is $O(kn^{1+1/k})$.

Let E' be the set of edges left in the graph after first phase, and let V' be the end-points of the edges E' . The first phase outputs a clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ for the vertices V' . Note that after each iteration in the first phase, the number of clusters reduces by a factor of $n^{1/k}$. Since there are $\lfloor \frac{k}{2} \rfloor$ number of iterations, the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ has only $n^{1-\frac{1}{k}\lfloor \frac{k}{2} \rfloor}$ clusters. Moreover, it follows from the Theorem 1 that a subset of edges added to the spanner in first phase of the algorithm spans the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ and ensures a bound of $\lfloor \frac{k}{2} \rfloor$ on the radius of each cluster $c \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$.

In order to ensure that the proposition $\mathcal{P}_k$ holds for the remaining vertices V' also, the algorithm makes use of Lemma 4 in the second phase of addition of edges.

Phase 2 : Joining the Clusters

In this phase, we perform the following two operations of adding the edges to the spanner depending on whether k is even or odd.

- If k is odd, then for each pair of clusters $c', c'' \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$, we add the least-weight edge between the two clusters (i.e., the least weight edge from the set $E'(C(u), C(v))$) to the spanner E_S . It follows from Lemma 4 that $\mathcal{P}_k$ holds for each edge $e \in E'$. Also note that the number of edges added is at-most square of the number of clusters, i.e., $|\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}|^2$, which is $n^{1+1/k}$.
- If k is even, then for each cluster $c \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$, we do the following. We group the vertices incident on the cluster c in their respective clusters of the clustering at the end of the $(\lfloor \frac{k}{2} \rfloor - 1)$ th iteration (the second last iteration). For each group of vertices (belonging to same cluster in the second last iteration) we pick the least-weight edge among the set of edges incident from these vertices to the cluster c , and add it to the spanner. It follows from Lemma 4 that $\mathcal{P}_k$ holds for each edge $e \in E'$.

The number of edges added is at-most the number of clusters in the last clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ times the number of clusters in the second last clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor - 1}$, which is $n^{1+1/k}$.

Figure 3 shows how the clusters are joined in the second phase of our algorithm for building the spanner of stretch 3, 5, 7 and 9 (i.e., $k = 2, 3, 4, 5$).

Let E_S^2 be the set of edges added to the spanner in the second phase of our algorithm. From the Theorem 1, and the arguments given above, we can state the following theorem.

Theorem 2. *If E_S^1 , and E_S^2 are the sets of edges selected to form the spanner in the first and the second phase of our algorithm for the graph $G(V, E)$ and parameter k , then the sub-graph $G(V, E_S^1 \cup E_S^2)$ is a $(2k - 1)$ -spanner for the graph $G(V, E)$ and consists of $O(kn^{1+1/k})$ edges.*

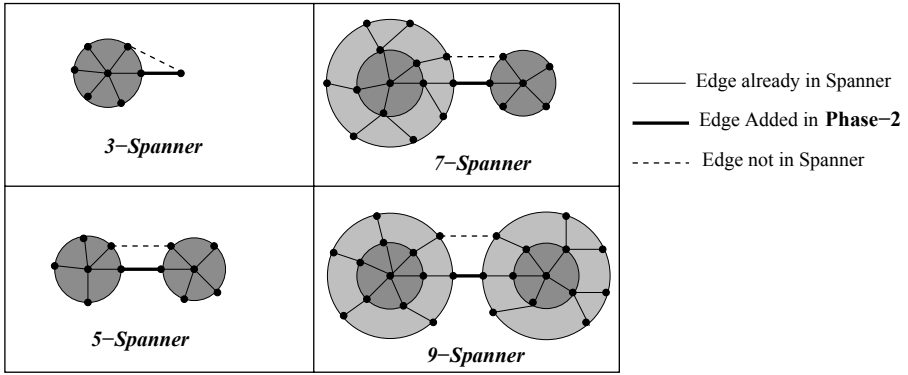


Fig. 3. Joining the clusters in the second phase, and ensuring the stretch-bound

Running time of the Algorithm :

We shall now show that both the phases of our algorithm run in $O(km)$ time. For sake of brevity, we sketch the analysis of running time of the first phase only (the analysis for the second phase is analogous).

It is easy to observe that having sampled a subset $\mathcal{R}_i$ of clusters from $\mathcal{C}_i$, it will take a total of $O(|E'|)$ time to find out the neighboring cluster (if exists) for each vertex $v \in V'_i$ that is incident with the least weight edge among all the sampled clusters adjacent to v .

In order to perform the remaining task of the i th iteration, we need an efficient way to pick the least weight edge from a set $E'(v, c)$, $c \in \mathcal{C}_i$ (to be added to the spanner), followed by removing the entire set $E'(v, c)$ from E' . This entire task of i th iteration can be accomplished in $O(|E'|)$ time if adjacency list of each $v \in V'_i$ is ordered in such a way that the edges incident on v from the vertices belonging to same cluster appear contiguous in the adjacency list of v . Such an order of edges in the adjacency lists can be achieved in $O(|E'|)$ time using a radix sort on the end-points of the edges as follows.

A clustering $\mathcal{C}$ on a set of vertices V' can be expressed by a labeling function $f_C : V' \rightarrow V'$ where $f_C(u) = f_C(v)$ iff both u and v belong to the same clusters in the clustering $\mathcal{C}$. Given a clustering $\mathcal{C}$, such a labeling f_C can be defined by labeling all the vertices of a cluster by a vertex (arbitrary one) picked from the same cluster. Given a graph $G(V, E)$ with a clustering $\mathcal{C}$ and its associated function f_C , we do the following :

We concatenate all the adjacency lists of the vertices to form a list $\mathcal{L}$. Note that an edge between a vertex u and a vertex v appears twice in this list - once as $e(u, v)$ (from the adjacency list of u), and once as $e(v, u)$ (from the adjacency list of v). First we sort $\mathcal{L}$ on the label of the second end-point of edges as defined by f_C . This will bring together all the edges whose second end-points belongs to the same cluster. Let $\mathcal{L}'$ be the new list after the sorting. Now we sort $\mathcal{L}'$ on the first end-point of edges. This will arrange all the edges of $\mathcal{L}'$ in such a way that edges belonging to adjacency list of a vertex appear together, and within each adjacency list also the edges incident from same cluster appear together. The

entire process of getting this desired ordering as explained above takes $O(|E'|)$ time since the radix sort runs in linear time.

Thus each iteration of the first phase of our algorithm runs in $O(|E'|) = O(m)$ time. Total number of iterations being $\lfloor \frac{k}{2} \rfloor$, it can be concluded that the running time of the first phase of our algorithm is $O(km)$. We repeat an iteration of the first phase if the total number of edges exceed $O(n^{1+1/k})$, and the expected number of repetitions is $O(1)$. Along similar lines, it can be shown that the second phase of our algorithm runs in $O(m)$ time. Thus the expected running time of our algorithm is $O(km)$. Combining with Theorem 2, we state the following Theorem.

Theorem 3. *Given a weighted graph $G(V, E)$, and integer $k > 1$, a spanner of stretch $(2k - 1)$ and $O(n^{1+1/k})$ size can be computed in expected time $O(km)$.*

References

1. I. Althofer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete and Computational Geometry*, 9:81–100, 1993.
2. B. Awerbuch. Complexity of network synchronization. *Journal of Ass. Compt. Mach.*, pages 804–823, 1985.
3. H. J. Bandelt and A. W. M. Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Journal of Advances of Applied Mathematics*, 7:309–343, 1986.
4. B. Bollobas. In *Extremal Graph Theory*, page 164. Academic Press, 1978.
5. J.A. Bondy and M. Simonovits. Cycle of even length in graphs. *Journal of Combinatorial Theory, Series B*, 16:97–105, 1974.
6. Edith Cohen. Fast algorithms for constructing t -spanners and paths with stretch t . *SIAM J. Comput.*, 28:210–236, 1998.
7. P. Erdos. Extremal problems in graph theory. In *In Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963)*, pages 29–36. Publ. House Czechoslovak Acad. Sci., Prague, 1964, 1963.
8. Shay Halperin and Uri Zwick. Unpublished result. 1996.
9. David Peleg and A. A. Schaffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.
10. David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of Assoc. Comp. Mach.*, 36(3):510–530, 1989.
11. Mikkel Thorup and Uri Zwick. Approximate distance oracle. In *Proceedings of 33rd ACM Symposium on Theory of Computing (STOC)*, pages 183–192, 2001.
12. Uri Zwick. Personal communication.

Multicommodity Flows over Time: Efficient Algorithms and Complexity

Alex Hall^{1,*}, Steffen Hippler², and Martin Skutella^{3,**}

¹ Computer Engineering and Networks Laboratory (TIK)
Gloriastrasse 35, ETH Zentrum, 8092 Zurich, Switzerland
hall@tik.ee.ethz.ch

² Institut für Mathematik, Technische Universität Berlin
Straße des 17. Juni 136, 10623 Berlin, Germany
hippler@math.tu-berlin.de

³ Algorithms and Complexity Group, Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
skutella@mpi-sb.mpg.de

Abstract. Flow variation over time is an important feature in network flow problems arising in various applications such as road or air traffic control, production systems, communication networks (e.g., the Internet), and financial flows. The common characteristic are networks with capacities and transit times on the arcs which specify the amount of time it takes for flow to travel through a particular arc. Moreover, in contrast to static flow problems, flow values on arcs may change with time in these networks.

While the ‘maximum s - t -flow over time’ problem can be solved efficiently and ‘min-cost flows over time’ are known to be NP-hard, the complexity of (fractional) ‘multicommodity flows over time’ has been open for many years. We prove that this problem is NP-hard, even for series-parallel networks, and present new and efficient algorithms under certain assumptions on the transit times or on the network topology. As a result, we can draw a complete picture of the complexity landscape for flow over time problems.

Keywords: network flow, routing, flow over time, dynamic flow, complexity, efficient algorithm

1 Introduction

A crucial characteristic of network flows occurring in real-world applications is flow variation over time. This characteristic is not captured by classical ‘static’

* Supported by the joint Berlin/Zurich graduate program Combinatorics, Geometry, and Computation (CGC), financed by ETH Zurich and the German Science Foundation (DFG)

** Supported in part by the EU Thematic Networks APPOL I & II, Approximation and Online Algorithms, IST-1999-14084, IST-2001-30012.

network flow models known from the literature. Moreover, apart from the effect that flow values on arcs may change over time, there is a second temporal dimension in many applications: Usually, flow does not travel instantaneously through a network but requires a certain amount of time to travel through each arc. Thus, not only the amount of flow to be transmitted but also the time needed for the transmission plays an essential role. Various interesting examples can be found in the survey articles of Aronson [1] and Powell, Jaillet, and Odoni [13].

The Model. Ford and Fulkerson [7,8] introduce the notion of *flows over time* (or ‘dynamic flows’) which comprises both temporal features mentioned above. They consider networks (directed graphs) $G = (V, E)$ with *capacities* u_e and *transit times* τ_e on the arcs $e \in E$. The transit time τ_e of an arc specifies the amount of time it takes for flow to travel from the tail to the head of that arc. In contrast to the classical case of static flows, a flow over time in such a network specifies a *flow rate* entering an arc for each point in time¹. In this setting, the capacity u_e of an arc limits the rate of flow into the arc at each point in time. In order to get an intuitive understanding of flows over time, one can associate arcs of the network with pipes in a pipeline system for transporting some kind of fluid. The length of each pipeline determines the transit time of the corresponding arc while the width determines its capacity. A precise definition of flows over time is given in Section 2.

Results from the Literature. Ford and Fulkerson [7,8] observe that a flow-over-time problem in a given network with transit times on the arcs can be transformed into an equivalent static flow problem in the corresponding *time-expanded network*. The time-expanded network contains one copy of the node set of the underlying network for each discrete time step θ (building a *time layer*). Moreover, for each arc e with transit time τ_e in the given network, there is a copy between each pair of time layers of distance τ_e in the time-expanded network. Thus, a discrete flow over time in the given network can be interpreted as a static flow in the corresponding time-expanded network. Since this interrelation works in both directions, the concept of time-expanded networks allows to solve a variety of flow over time problems by applying algorithmic techniques developed for static network flows; see, e.g., [6]. Notice, however, that one has to pay for this simplification of the considered flow problem in terms of an enormous increase in the size of the network. In particular, the size of the time-expanded network is only pseudo-polynomial in the input size and thus does not directly lead to efficient algorithms for computing flows over time.

Ford and Fulkerson [7,8] give an efficient algorithm for the problem of sending the maximum possible amount of flow from one source s to one sink t within a given time horizon T . The problem can be solved by essentially one ‘static’ min-cost flow computation on the given network. Ford and Fulkerson show that

¹ In fact, the discrete flow model considered by Ford and Fulkerson is slightly different from the model we consider in this paper. However, Fleischer and Tardos [6] point out that the two models are essentially equivalent; see also [4].

an optimal solution to this min-cost flow problem can be turned into a maximal flow over time by first decomposing it into flows on s - t -paths. The corresponding flow over time starts to send flow on each path at time zero, and repeats each so long as there is enough time left in the T time units for the flow along the path to arrive at the sink. A flow over time featuring this structure is called *temporally repeated*.

A problem closely related to the one considered by Ford and Fulkerson is the *quickest s - t -flow problem*. Here, instead of fixing the time horizon T and asking for a flow over time of maximal value, the value of the flow (demand) is fixed and T is to be minimized. This problem can be solved in polynomial time by incorporating the algorithm of Ford and Fulkerson into a binary search framework. Using Megiddo's method of parametric search [12], Burkard, Dlaske, and Klinz [2] present a faster algorithm which solves the quickest s - t -flow problem in strongly polynomial time.

Hoppe and Tardos [10,9] study the *quickest transshipment problem* which asks for a flow over time satisfying given supplies and demands at the nodes of a network within minimum time. Surprisingly, this problem turns out to be much harder than the special case with a single source and sink. Hoppe and Tardos give a polynomial time algorithm for the problem, but this algorithm relies on submodular function minimization and is thus much less efficient than for example the algorithm of Ford and Fulkerson for maximum s - t -flows over time.

Even more surprising, Klinz and Woeginger [11] show that the problem of computing a minimum cost s - t -flow over time with prespecified value and time horizon is NP-hard. On the other hand, this problem can be solved in pseudo-polynomial time by a static min-cost flow computation in the time-expanded network. Klinz and Woeginger also point out that the class of temporally repeated flows does in general not contain a min-cost s - t -flow over time. In fact, it is even strongly NP-hard to compute a temporally repeated solution of minimum cost [11].

Fleischer and Skutella [4,5] introduce a 'condensed' variant of time-expanded networks which is based on a rougher discretization of time and therefore leads to networks whose size is polynomially bounded in the input size. This approach yields fully polynomial time approximation schemes (FPTASes) for various variants of the weakly NP-hard quickest flow problem with costs. The best known result for the strongly NP-hard problem of computing a quickest temporally repeated flow of minimum cost is a $(2 + \epsilon)$ -approximation algorithm [4], which is based on a length-bounded static flow computation.

Contribution of this Paper. The results in [4,5] also hold for the more general setting with multiple commodities. Multicommodity flows model the transportation of several distinct types of flow through a single network. The resulting problems are typically much harder than their single-commodity counterparts. For example, the only known polynomial-time algorithms for static multicommodity flow computations require general linear programming techniques (e.g., the ellipsoid method or interior point methods). The complexity of the multicommodity flow

Table 1. The complexity landscape of flows over time in comparison to the corresponding static flow problems. The third column ‘transshipment’ refers to single-commodity flows with several source and sink nodes. The NP-hardness results marked with a ‘*’ are proved in this paper. The weak NP-hardness results even hold for series-parallel networks. On the other hand, we prove that these problems can efficiently be solved in tree networks and arbitrary networks with ‘uniform path-lengths’. The ‘pseudo-poly’ entries follow since the respective problems can be solved as static flow problems in the time-expanded network. The quoted approximation results hold for the corresponding quickest flow problems.

	s - t -flow	transshipment	min-cost	multicommodity
(static) flow	poly	poly ($\simeq s$ - t -flow)	poly	poly ($\simeq$ LP)
flow over time with storage	poly [7] ($\simeq$ min-cost flow)	poly [10] ($\simeq$ subm. func.)	pseudo-poly NP-hard [11] FPTAS [5]	pseudo-poly NP-hard* FPTAS [5]
flow over time without storage				strongly NP-hard* 2-approx. [4]

over time problem (without costs) has so far been open. In his excellent PhD thesis [9] on flows over time, Hoppe poses the problem of developing a polynomial time algorithm to solve fractional multicommodity flows over time. In Section 5 we prove that such an algorithm does not exist, unless $P=NP$. In fact, the multicommodity flow over time problem is NP-hard, even when restricted to series-parallel networks or to the case of only two commodities.

Flows over time raise issues that do not arise in standard network flows. One issue is storage of flow at intermediate nodes. In most applications (such as, e.g., traffic routing, evacuation planning, telecommunications), storage is limited, undesired, or even prohibited at intermediate nodes. For single commodity problems, storage is unnecessary, even in the NP-hard setting with costs [5]. However, for the quickest multicommodity flow problem, there exist instances where the time horizon of an optimal solution increases by a factor of $4/3$ when storage of flow at intermediate nodes is prohibited. In Section 5 we mention that the multicommodity flow over time problem with simple flow paths and without storage of flow at intermediate nodes is strongly NP-hard. Without the latter restriction the problem can be solved in pseudo-polynomial time as a static multicommodity flow problem in the time-expanded network. The best known result for the strongly NP-hard variant with simple flow paths and no intermediate storage is a 2-approximation algorithm for the quickest multicommodity flow problem [4]. An overview of the complexity landscape of flows over time is given in Table 1.

Motivated by the results on the hardness of multicommodity flows over time in Section 5, we study special conditions on the transit times of arcs and on the network topology under which multicommodity flows over time can be computed in polynomial time. In Section 3 we consider arbitrary network topologies with transit times on the arcs satisfying the following condition: All paths (in

the corresponding bidirected network) between every fixed pair of nodes have the same transit time. This condition is, for example, obviously satisfied for the important but non-trivial case of tree networks. We show that, under this assumption, many flow over time problems can be solved as static flow problems in a polynomial-size variant of time-expanded networks with $O(n)$ time layers ($n := |V|$). We believe that this result is also of interest for flow over time problems, like the quickest transshipment problem, which are known to be solvable in polynomial time for arbitrary transit times. While the algorithm of Hoppe and Tardos [10,9] relies on submodular function minimization, we can solve the special case of the problem as a static s - t -flow problem in a network with $O(n^2)$ nodes and $O(nm)$ arcs ($m := |E|$). The presented approach works for both settings, with and without storage of flow at intermediate nodes.

Finally, in Section 4 we consider networks with arbitrary transit times where every node has at most one outgoing arc. In particular, there is a unique source-sink path for every commodity in such networks. Under the assumption that storage of flow at intermediate nodes is allowed, we present a simple greedy algorithm for the quickest multicommodity flow problem: Whenever there is a conflict between several commodities using the same arc, the algorithm gives top priority to the commodity which is furthestmost from its sink node. We prove that this simple strategy yields an optimal solution in polynomial time. The proof uses a generalized notion of ‘earliest arrival flows’.

Due to space limitations, we omit most proofs in this extended abstract. For details we refer to the full version of the paper.

2 Preliminaries

We are considering network flow problems in a network (directed graph) $G = (V, E)$ with $n := |V|$ nodes and $m := |E|$ arcs. For an arc $e = (v, w)$ we write $\text{head}(e) := w$ and $\text{tail}(e) := v$. For a node $v \in V$, the terms $\delta^+(v)$ and $\delta^-(v)$ denote the set of arcs leaving node v ($\text{tail}(e) = v$) and entering node v ($\text{head}(e) = v$), respectively. Each arc $e \in E$ has associated with it a positive capacity u_e and a nonnegative transit time $\tau_e \in \mathbb{R}^+$. Moreover, in the setting with costs, each arc e has associated a non-negative cost coefficient c_e which determines the per unit cost for sending flow through the arc.

There is a set of commodities $K = \{1, \dots, k\}$, where every commodity $i \in K$ is defined by a set of terminals $S_i \subseteq V$ which can be partitioned into a subset of sources S_i^+ and sinks S_i^- . Every source node $v \in S_i^+$ has a supply $D_{v,i} \geq 0$ and every sink $v \in S_i^-$ has a demand $D_{v,i} \leq 0$ such that $\sum_{v \in S_i} D_{v,i} = 0$. In the special case of only one source $s_i \in V$ and one sink $t_i \in V$ we let $d_i := D_{s_i,i} = -D_{t_i,i}$ and refer to d_i as the *demand* of commodity i .

Static Flows. A *static (multicommodity) flow* x in G assigns every arc-commodity pair e, i a non-negative flow value $x_{e,i}$ such that *flow conservation* holds:

$$\sum_{e \in \delta^-(v)} x_{e,i} - \sum_{e \in \delta^+(v)} x_{e,i} = 0 \quad \text{for all } v \in V \setminus S_i \text{ and } i \in K. \quad (1)$$

The static flow x satisfies the supplies and demands if

$$\sum_{e \in \delta^+(v)} x_{e,i} - \sum_{e \in \delta^-(v)} x_{e,i} = D_{v,i} \quad \text{for all } v \in S_i \text{ and } i \in K.$$

Finally, x is called *feasible* if it obeys the capacity constraints $x_e := \sum_{i \in K} x_{e,i} \leq u_e$, for all $e \in E$. The cost of a static flow x is defined as $c(x) := \sum_{e \in E} c_e x_e$.

Flows Over Time. A (multicommodity) flow over time f in G with time horizon T is given by Lebesgue-measurable functions $f_{e,i} : [0, T) \rightarrow \mathbb{R}^+$ where $f_{e,i}(\theta)$ is the rate of flow (per time unit) of commodity i entering arc e at time θ . In order to simplify notation, we sometimes use $f_{e,i}(\theta)$ for $\theta \notin [0, T)$, implicitly assuming that $f_{e,i}(\theta) = 0$ in this case.

The flow $f_{e,i}(\theta)$ of commodity i entering arc e at time θ arrives at $\text{head}(e)$ at time $\theta + \tau_e$. All arcs must be empty from time T on, i.e., $f_{e,i}(\theta) = 0$ for $\theta \geq T - \tau_e$. To generalize the notion of *flow conservation* to flows over time, we integrate the flow conservation constraints (1) over time. Depending on the model, *storage of flow at intermediate nodes* might be allowed. That is, flow entering a node can be held back for some time before it is sent onward. To rule out deficit at any node, we require

$$\int_0^\theta \left(\sum_{e \in \delta^+(v)} f_{e,i}(\xi) - \sum_{e \in \delta^-(v)} f_{e,i}(\xi - \tau_e) \right) d\xi \leq 0 \quad (2)$$

for all $i \in K$, $\theta \in [0, T)$, $v \in V \setminus S_i^+$. Moreover, we require that equality holds in (2) for $i \in K$, $\theta = T$, and $v \in V \setminus S_i$, meaning that no flow should remain in the network after time T . In the model without storage of flow at intermediate nodes, we additionally require that equality holds in (2) for all $i \in K$, $\theta \in [0, T)$, and $v \in V \setminus S_i$.

The flow over time f satisfies the supplies and demands if by time T the net flow out of each terminal $v \in S_i$ of commodity i equals its supply $D_{v,i}$:

$$\int_0^T \left(\sum_{e \in \delta^+(v)} f_{e,i}(\xi) - \sum_{e \in \delta^-(v)} f_{e,i}(\xi - \tau_e) \right) d\xi = D_{v,i} \quad (3)$$

for all $i \in K$, $v \in S_i$. In the setting of flows over time, the capacity u_e is an upper bound on the *rate* of flow entering arc e at any moment of time. Thus, a flow over time f is feasible if $f_e(\theta) \leq u_e$ for all $\theta \in [0, T)$ and $e \in E$. Here, $f_e(\theta) := \sum_{i \in K} f_{e,i}(\theta)$ is the total rate at which flow is entering arc e at time θ . The cost of a flow over time f is defined as $c(f) := \sum_{e \in E} c_e \int_0^T f_e(\theta) d\theta$.

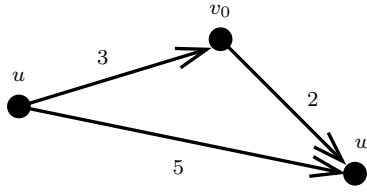


Fig. 1. Example of a network with uniform path-lengths. The numbers at the arcs indicate transit times.

Problem Definition. Given a network G with capacities and transit times on the arcs, a set of commodities with supplies and demands at their terminals, and a time horizon T , the *multicommodity flow over time problem* asks for a feasible flow over time with time horizon T , satisfying all supplies and demands. In the setting with costs, the *min-cost (multicommodity) flow over time problem* asks for such a flow over time with minimum cost. Another interesting objective for flows over time is to minimize the time horizon: The *quickest (multicommodity) flow problem (with costs)* is to find a flow over time in G that satisfies all supplies and demands within minimal time T (and whose cost is bounded by a given *budget* C). Finally, in the *maximum (multicommodity) flow over time problem (with costs)* we are given a time horizon T and instead of having supplies and demands at the terminals, the goal is to maximize the total amount of flow being sent from sources to sinks (under the condition that the costs are bounded by a given *budget* C). Of course, flow of each commodity can only be sent from its sources to its sinks.

3 Networks with Uniform Path-Lengths

In this section we present a polynomial-time algorithm for the min-cost multicommodity flow over time problem in a special class of networks which, in particular, comprises trees. Even on trees the multicommodity flow over time problem is far from being trivial. For example, it follows by a straightforward reduction from the *wavelength routing problem* [3] that finding an *integral* multicommodity flow over time is NP-hard in binary trees.

For a given network $G = (V, E)$ with transit times on the arcs, we let $\overleftrightarrow{G} = (V, \overleftrightarrow{E})$ denote the corresponding bidirected network with $\overleftrightarrow{E} := E \cup \overleftarrow{E}$ and $\overleftarrow{E} := \{(v, u) \mid (u, v) \in E\}$. Moreover, the ‘transit time’ of a backward arc $(v, u) \in \overleftarrow{E}$ is set to $\tau_{(v, u)} := -\tau_{(u, v)}$. In the following we assume that G is connected and that the transit time of every directed cycle in $\overleftrightarrow{G}$ is zero. The latter requirement is equivalent to the condition that, for all $u, v \in V$, the transit time of any two u - v -paths in $\overleftrightarrow{G}$ is equal. Therefore we refer to this class of networks with transit times as *networks with uniform path-lengths*. An example is given in Figure 1.

Let $v_0 \in V$ be an arbitrary but fixed node. For $v \in V$ let τ_v denote the transit time of a v - v_0 -path in $\overleftrightarrow{G}$. In the network depicted in Figure 1, these

values are $\tau_u = 3$, $\tau_{v_0} = 0$, and $\tau_w = -2$. For a given time horizon T , let $\mathcal{T} := \{\tau_v, T + \tau_v \mid v \in V\}$. If $T := 7$ for the network in Figure 1, then $\mathcal{T} = \{3, 10, 0, 7, -2, 5\}$. Notice that τ_v is the earliest point in time at which flow emerging from node v could possibly arrive at v_0 . Similarly, $T + \tau_v$ is the latest point in time at which flow can be sent from v_0 to v such that it still arrives in time. Hence, $\mathcal{T}$ contains all ‘essential’ points in time at which decisions have to be made. We show below that it is sufficient to change the outflow rate out of arcs arriving at v_0 and the inflow rate into arcs leaving v_0 at these points in time only. The same property holds for all other nodes $v \in V$ and incident arcs when $\mathcal{T}$ is replaced by $\mathcal{T} - \tau_v := \{\theta - \tau_v \mid \theta \in \mathcal{T}\}$. Since $|\mathcal{T}| \leq 2|V|$, this insight constitutes the backbone of the results presented in this section.

Before stating an exact version in Lemma 1, we first introduce some additional notation: Sort the elements in $\mathcal{T}$ such that $\theta_1 < \theta_2 < \dots < \theta_q$. Moreover, let $\theta_0 := -\infty$ and $\theta_{q+1} := \infty$, and define a corresponding partition of the time axis $[-\infty, \infty)$ by time intervals $I_j := [\theta_j, \theta_{j+1})$, $j = 0, \dots, q$. In Figure 1 with $T = 7$ we get $q = 6$ and the time intervals $I_0 = [-\infty, -2)$, $I_1 = [-2, 0)$, $I_2 = [0, 3)$, $I_3 = [3, 5)$, $I_4 = [5, 7)$, $I_5 = [7, 10)$, and $I_6 = [10, \infty)$.

Lemma 1. *If there exists a flow over time f with time horizon T satisfying all supplies and demands, then there exists a corresponding solution $\bar{f}$ with $c(\bar{f}) = c(f)$ and the following additional property: For every arc $e = (u, v) \in E$ and every time interval $I_j - \tau_u := [\theta_j - \tau_u, \theta_{j+1} - \tau_u)$, the flow rate $\bar{f}_{e,i}(\theta)$ is constant for $\theta \in I_j - \tau_u$, for every commodity $i \in K$.*

Proof. For $e = (u, v) \in E$ and $i \in K$, define

$$\bar{f}_{e,i}(\theta) := \frac{1}{|I_j|} \int_{I_j - \tau_u} f_{e,i}(\xi) d\xi \quad \text{for } \theta \in I_j - \tau_u, j = 0, \dots, q.$$

Here, $|I_j| := t_{j+1} - t_j$ denotes the length of the time intervals I_j and $I_j - \tau_u$. Hence, $\bar{f}$ arises from f by averaging flow on arcs $e = (u, v)$ within time intervals $I_j - \tau_u$, $j = 0, \dots, q$.

By definition of $\mathcal{T}$, each time interval $I_j - \tau_u$ is either contained in $[0, T - \tau_e)$ or disjoint from $[0, T - \tau_e)$, for all $e = (u, v) \in E$. This is clear since $\mathcal{T} - \tau_u$ contains both 0 and $T - \tau_e$. (For instance, for $T = 7$ and node u in Figure 1, the intervals $I_j - \tau_u$, $j = 0, \dots, q$, are $[-\infty, -5)$, $[-5, -3)$, $[-3, 0)$, $[0, 2)$, $[2, 4)$, $[4, 7)$, and $[7, \infty)$.) In particular, for all $e \in E$ and $i \in K$, we get $\bar{f}_{e,i}(\theta) = 0$ for $\theta \notin [0, T - \tau_e)$ since this property certainly holds for the given solution f .

Moreover, it follows from the definition of $\bar{f}$ that no flow is rerouted compared to f . Thus, $\bar{f}$ satisfies all supplies and demands (see (3)) and its cost is equal to the cost of f . It is easy to see that $\bar{f}$ satisfies the capacity constraints since f does. It therefore remains to show that $\bar{f}$ satisfies the flow conservation constraints (2). By definition of $\bar{f}$, the left hand side of (2) is equal for f and $\bar{f}$ if $\theta := \theta_j - \tau_v$ for some j . To see this, note that $\theta_j - \tau_v - \tau_e = \theta_j - \tau_u =: t$, for all arcs $e = (u, v) \in \delta^-(v)$ and thus t is on an interval boundary for u as well. Let $\delta_{v,i,j}$ denote the (non-positive) left hand side of (2) for $\bar{f}$ and $\theta := \theta_j - \tau_v$. Then, for $\theta_j - \tau_v < \theta < \theta_{j+1} - \tau_v$, the left hand side of (2) for $\bar{f}$ is a convex combination

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e \sum_{j=1}^{q-1} x_{e,i,j} \\ \text{s.t.} \quad & \sum_{j=1}^{q-1} \left(\sum_{e \in \delta^+(v)} x_{e,i,j} - \sum_{e \in \delta^-(v)} x_{e,i,j} \right) = D_{v,i} \quad \text{for all } i \in K, v \in S_i, \end{aligned} \quad (4)$$

$$\sum_{j=1}^p \left(\sum_{e \in \delta^+(v)} x_{e,i,j} - \sum_{e \in \delta^-(v)} x_{e,i,j} \right) \leq 0 \quad \begin{array}{l} \text{for all } i \in K, v \in V \setminus \\ S_i^+, \\ \text{and } p = 1, \dots, q-1, \end{array} \quad (5)$$

$$\sum_{i \in K} x_{e,i,j} \leq |I_j| u_e \quad \text{for all } e \in E, j = 1, \dots, q-1, \quad (6)$$

$$\begin{aligned} x_{e,i,j} &= 0 && \text{for all } e = (u, v) \in E, i \in K, \\ &&& \text{and } I_j - \tau_u \not\subseteq [0, T - \tau_e), \\ x_{e,i,j} &\geq 0 && \text{for all } e \in E, i \in K, \text{ and } j = 1, \dots, q-1. \end{aligned} \quad (7)$$

Fig. 2. A linear programming formulation of the min-cost multicommodity flow over time problem in networks with uniform path-lengths.

of $\delta_{v,i,j}$ and $\delta_{v,i,j+1}$ and therefore non-positive as well. This concludes the proof. $\square$

It follows from the last part of the proof of Lemma 1 that storage of flow at intermediate nodes does not occur in $\bar{f}$ if it does not occur in f .

Corollary 1. *Lemma 1 also holds if storage of flow at intermediate nodes is prohibited.*

The min-cost multicommodity flow over time problem can now be formulated as a linear program of polynomial size; see Figure 2. For $e \in E$, $i \in K$, and $j = 1, \dots, q-1$, the variable $x_{e,i,j}$ denotes the constant flow rate of commodity i into arc $e = (u, v)$ during the time interval $I_j - \tau_u$, multiplied by $|I_j|$. Constraints (4) correspond to (3) and enforce the satisfaction of all supplies and demands. Constraints (5) are a reformulation of the flow conservation constraints (2). In particular, replacing “ $\leq$ ” by “ $=$ ” in (5) yields a formulation for the model where storage of flow at intermediate nodes is prohibited. Constraints (6) correspond to the capacity constraints. Finally, constraints (7) ensure that flow can only occur within the time interval $[0, T)$.

Since linear programs can be solved efficiently (e.g., by interior point methods), we get the following main result of this section.

Theorem 1. *The min-cost multicommodity flow over time problem (with or without storage of flow at intermediate nodes) in networks with uniform path-lengths can be solved in polynomial time.*

While this result relies on general linear programming techniques, we can give more efficient algorithms for the special case of a single commodity. These

algorithms are based on the insight that the linear program stated above can be interpreted as a classical network flow problem in a network $G' = (V', E')$. We omit further details due to space restrictions.

Theorem 2. *A min-cost (multicommodity) flow over time problem in a network $G = (V, E)$ with uniform path-lengths can be solved by a static min-cost (multicommodity) flow computation in a network G' with $O(n^2)$ nodes and $O(nm)$ arcs.*

We finally turn to the quickest multicommodity flow problem (with bounded cost) in networks with uniform path-lengths. As a result of Theorem 2, this problem can be solved within a binary search framework by a series of static flow computations. There even exists a strongly polynomial algorithm with considerably improved running time.

Theorem 3. *The quickest (multicommodity) flow problem in a network $G = (V, E)$ with uniform path-lengths can be solved by $O(\log n)$ static (multicommodity) flow computations and one parametric static flow computation in networks with $O(n^2)$ nodes and $O(nm)$ arcs.*

4 Networks with Out-Degree at Most One

In this section we discuss a combinatorial greedy algorithm for the quickest multicommodity flow problem in networks whose nodes have either in- or out-degree at most one. (In the following we restrict without loss of generality to the latter case.) This class contains paths, cycles, in- and out-trees and also combinations such as a cycle with one or more of its nodes being roots of in-respectively out-trees. In the following we assume that every commodity $i \in K$ has exactly one source node s_i and one sink node t_i . An important feature of the networks under consideration is that there exists a unique s_i - t_i -path P_i for every commodity i .

The basic notion of our greedy algorithm is to schedule the flow according to priorities that are assigned to the individual commodities in such a way that the higher a commodity's priority, the longer (with respect to the number of arcs) its remaining flow path lying ahead. Intuitively, in a traffic network this approach corresponds to giving the right of way to those road users that are furthestmost from their destinations. Since this approach introduces waiting times at intermediate nodes, we do not restrict storage of flow at intermediate nodes in this section. We skip all further details in this extended abstract.

Theorem 4. *The greedy algorithm yields a quickest multicommodity flow and runs in time $O(mk^2)$, where $m := |E|$ is the number of arcs.*

5 Hardness Results

In this section we prove NP-hardness for the multicommodity flow over time problem. As in the last section, we consider instances with one source s_i , one sink t_i , and demand d_i for every commodity $i \in K$.

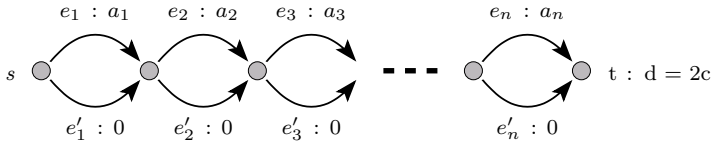


Fig. 3. Reduction from PARTITION: All arcs have unit capacity. The arc transit times are given in the picture. For the upper arcs e_j , $j = 1, \dots, n$, they are taken from the PARTITION instance: $a_1, \dots, a_n$. The transit times of the lower arcs e'_j are 0. At the sink node t a demand of $d = 2c$ is present.

Theorem 5. *The multicommodity flow over time problem with or without storage at intermediate nodes is NP-hard on series-parallel networks.*

We give a reduction from the NP-hard PARTITION problem: Given n integer numbers $a_1, \dots, a_n \in \mathbb{N}$ with $\sum_{i=1}^n a_i = 2L$ for some $L \in \mathbb{N}$, the task is to decide whether there is a subset $B \subseteq \{1, \dots, n\}$ such that $\sum_{i \in B} a_i = L$.

Given an instance of PARTITION, we construct a *chord* as shown in Figure 3 and introduce the first commodity with source s , sink t , and demand $d = 2c$. Here, $c < 1$ is some positive constant. Further commodities will be added later. The time horizon is set to $T := L + c$. This constitutes the main building block of the reduction, which is similar to other well known reductions of PARTITION to flow problems; see, e.g., [11].

Let $\mathcal{P}$ denote the set of all 2^n paths from s to t . For $P \in \mathcal{P}$, let $\tau(P) := \sum_{e \in P} \tau_e$ be the length of path P . For a given flow over time f (of the first commodity) and a path $P \in \mathcal{P}$, let x_P denote the total amount of flow of the first commodity routed along P , that is, $x_P = \int_0^T f_P(\theta) d\theta$. Furthermore, we define $x_e := \sum_{P \in \mathcal{P}: e \in P} x_P$ to be the amount of flow of the first commodity routed through arc e .

Lemma 2. *It is NP-hard to decide whether a flow amount of $2c$ can be routed from s to t within time $T = L + c$, such that $x_e \leq c$ holds for all arcs e .*

Proof. We will show that $2c$ can be routed from s to t if and only if there is a solution to the PARTITION instance, i.e., there is $B \subset \{1, \dots, n\}$ such that $\sum_{i \in B} a_i = L$. First the easy direction: If such a subset B exists, route c units of flow along the path P_B consisting of arcs $\{e_i \mid i \in B\} \cup \{e'_i \mid i \notin B\}$. The remaining c units are routed along the complementary path $P_{\bar{B}}$ with $\bar{B} := \{1, \dots, n\} \setminus B$. It is clear that $\tau(P_B) = \tau(P_{\bar{B}}) = L$. Therefore, $2c$ units can be routed in time T .

Now we prove the other direction: Assume that no such B exists but nevertheless $2c$ units can be routed in time $T = L + c$. Since $c < 1$ and all a_i are integer, we know that any path $P \in \mathcal{P}$ with non-zero flow amount $x_P > 0$ must have length $\tau(P) < L$. Note that length L is not possible. Since the flow is distributed over a finite number of paths (at most 2^n), there is at least one such path P_l with $x_{P_l} \geq 2c2^{-n}$. Consider the following sum of weighted path lengths

$$\sum_{P \in \mathcal{P}} x_P \tau(P) = \sum_{P \in \mathcal{P}} x_P \sum_{i \in \{1, \dots, n\}: e_i \in P} a_i = \sum_{i=1}^n a_i \sum_{P \in \mathcal{P}: e_i \in P} x_P = c \sum_{i=1}^n a_i = c2L.$$

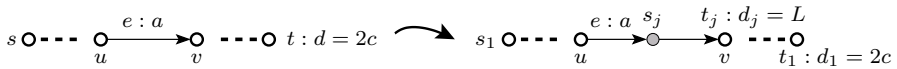


Fig. 4. To let only c units of flow pass through e in the given time horizon $T = L + c$, a commodity j is added with demand $d_j = L$. The new arc (s_j, v) has capacity 1 and length 0.

In the first two equations we use the definition of $\tau(P)$ and exchange the order of summation. Then in the last but one step, we apply $x_e \leq c$, whereby the equality stems from our assumption that $2c$ units are routed. We know that a positive amount $x_{P'}$ is routed along some path $P' \in \mathcal{P}$ with $\tau(P') < L$. But then, since the sum of weighted path lengths is equal to $2cL$, the assumption $\sum_{P \in \mathcal{P}} x_P = 2c$ yields the existence of at least one flow-carrying path P'' with $\tau(P'') > L$, which compensates for $\tau(P') < L$. Thus, the flow over time cannot finish within time $T = L + c$. $\square$

Our aim is now to enforce the bound $x_e \leq c$ for all arcs e by introducing one further commodity j per arc.

We split every arc $e = (u, v)$ into two consecutive arcs (u, s_j) and (s_j, v) with unit capacity; see Figure 4. The transit time of (s_j, v) is zero and the transit time of (u, s_j) equals the transit time of the original arc e . Notice that this modification has no impact on feasible flows over time for the first commodity. Now we introduce the additional commodity j with source s_j and sink $t_j = v$. The demand of this new commodity is set to L . Therefore, at most c additional units of flow of the first commodity can be sent through arc (s_j, v) within time $T = L + c$. This completes the proof of Theorem 5.

We conclude this section by mentioning several stronger hardness results. Details can be found in the full version of the paper.

Theorem 6. *The multicommodity flow over time problem is already NP-hard for the case of only two commodities. The same holds for the maximum multicommodity flow over time problem.*

Theorem 7. *The multicommodity flow over time problem with simple flow paths and without storage of flow at intermediate nodes is NP-hard in the strong sense. The same holds for the maximum multicommodity flow over time problem.*

Theorem 8. *There is no FPTAS for the quickest multicommodity flow problem with simple flow paths and without storage of flow at intermediate nodes, unless $P=NP$.*

Acknowledgments. We are much indebted to Lisa Fleischer, Ekkehard Köhler, Katharina Langkau, and Jim Orlin for helpful discussions on the topic of this paper.

References

1. J. E. Aronson. A survey of dynamic network flows. *Annals of Operations Research*, 20:1–66, 1989.
2. R. E. Burkard, K. Dlaska, and B. Klinz. The quickest flow problem. *ZOR — Methods and Models of Operations Research*, 37:31–58, 1993.
3. T. Erlebach and K. Jansen. Call scheduling in trees, rings and meshes. In *Proceedings of the 30th Hawaii International Conference on System Sciences*, pages 221–222. IEEE Computer Society Press, 1997.
4. L. Fleischer and M. Skutella. The quickest multicommodity flow problem. In W. J. Cook and A. S. Schulz, editors, *Integer Programming and Combinatorial Optimization*, volume 2337 of *Lecture Notes in Computer Science*, pages 36–53. Springer, Berlin, 2002.
5. L. Fleischer and M. Skutella. Minimum cost flows over time without intermediate storage. In *Proceedings of the 14th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 66–75, Baltimore, MD, 2003.
6. L. K. Fleischer and É. Tardos. Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters*, 23:71–80, 1998.
7. L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.
8. L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, 1962.
9. B. Hoppe. *Efficient dynamic network flow algorithms*. PhD thesis, Cornell University, 1995.
10. B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25:36–62, 2000.
11. B. Klinz and G. J. Woeginger. Minimum cost dynamic flows: The series-parallel case. In E. Balas and J. Clausen, editors, *Integer Programming and Combinatorial Optimization*, volume 920 of *Lecture Notes in Computer Science*, pages 329–343. Springer, Berlin, 1995.
12. N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.
13. W. B. Powell, P. Jaillet, and A. Odoni. Stochastic and dynamic networks and routing. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 3, pages 141–295. North–Holland, Amsterdam, The Netherlands, 1995.

Multicommodity Demand Flow in a Tree

(Extended Abstract)

Chandra Chekuri¹, Marcelo Mydlarz², and F. Bruce Shepherd¹

¹ Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974
{chekuri, bshep}@research.bell-labs.com

² Computer Science Dept., Rutgers University, Piscataway, NJ 08854-8019
marcem@cs.rutgers.edu

Abstract. We consider requests for capacity in a given tree network $T = (V, E)$ where each edge of the tree has some integer capacity u_e . Each request consists of an integer demand d_f and a profit w_f which is obtained if the request is satisfied. The objective is to find a set of demands that can be feasibly routed in the tree and which provide a maximum profit. This generalizes well-known problems including the knapsack and b -matching problems.

When all demands are 1, we have the integer multicommodity flow problem. Garg, Vazirani, and Yannakakis [5] had shown that this problem is NP-hard and gave a 2-approximation algorithm for the cardinality case (all profits are 1) via a primal-dual algorithm. In this paper we establish for the first time that the natural linear programming relaxation has a constant factor gap, a factor of 4, for the case of *arbitrary* profits.

We then discuss the situation for arbitrary demands. When the maximum demand $d_{\max}$ is at most the minimum edge capacity $u_{\min}$, we show that the integrality gap of the LP is at most 48. This result is obtained showing that the integrality gap for demand version of such a problem is at most 12 times that for the unit demand case. We use techniques of Kolliopoulos and Stein [8,9] to obtain this. We also obtain, via this method, improved algorithms for the line and ring networks. Applications and connections to other combinatorial problems are discussed.

Keywords: integer multicommodity flow, tree, integrality gap, packing integer program, approximation algorithm.

1 Introduction

Let $T = (V, E, u)$ be a capacitated tree network, where each edge capacity u_e is an integer. T is termed the *supply graph* and throughout we let n denote $|V|$. We are also given a collection of demands which is encoded as a multigraph $H = (V, F, d, w)$ where each *demand* edge $f \in F$ has an associated integer value d_f and a real valued *profit* w_f . The profit w_f is only obtained if the whole demand is satisfied. H is termed the *demand graph*.

A subset $S \subseteq F$ is *routable* (in T) if the demands can be simultaneously routed without violating any edge capacity of the tree. The DEMAND FLOW PROBLEM (DFP) is to find a routable subset S which maximizes $w(S)$. We use the term “demand flow” in reference to the all-or-nothing aspect of the optimization problem and not to unsplittability of the flows. For instance, demand flows in general graphs could in fact be fractional flows. We mention that we do not know the status (with respect to approximability) of the maximum demand flow problem in general graphs. We discuss this in some detail at the end of the introduction. Note, however, that on a tree, demand flows and unsplittable flows are the same since there is a unique path between every pair of vertices.

A natural linear programming (LP) relaxation for the demand flow problem on a tree is given below, where x_f denotes the percentage of demand f being satisfied, and $\text{Path}(f)$ denotes the unique path in T joining the endpoints of f .

$$\max \sum_{f \in F} w_f x_f \quad \text{s.t.} \quad (1)$$

$$\sum_{f: e \in \text{Path}(f)} d_f x_f \leq u_e \quad \forall e \in E \quad (2)$$

$$x_f \leq 1 \quad \forall f \in F \quad (3)$$

$$x_f \geq 0 \quad \forall f \in F. \quad (4)$$

Obviously, the demand flow problem is modelled by adding the constraint $x_f \in \{0, 1\}$ for each demand edge f .

Our main focus is to study the integrality gap for this linear program. We establish, for the first time, that it has an $O(1)$ (approximately 48) factor gap if we assume that each demand is bounded above by the minimum capacity of an edge in the tree. A trivial lower bound of 3 on the gap is obtained by considering a star with three edges, and a triangle of demand edges. Let the capacities of the supply edges be $2k$, and the demands be $k + 1$. Asymptotically, we may fractionally pack 3 demands worth, while integrally, we may only pack a single demand.

We now review some of the well-known combinatorial optimization problems that arise as restricted versions of the demand flow problem on the tree. We first discuss the case where all demands are 1, that is, the *integer multicommodity flow problem*.

Unit Demands:

(1) *Tree is a path; Unit capacities:* Suppose that the tree is just a path and each of its edges has unit capacity. We are then essentially looking for a maximum weight stable set in a path intersection graph. Namely, the graph contains a node for each demand d_f , and two nodes are adjacent if their corresponding paths on the line intersect. It is well known that such graphs are perfect. The maximum weight stable set is then equal to a minimum cost clique cover, and both objects may be found efficiently by means of dynamic programming.

(2) *Tree is a path; Arbitrary capacities:* Note that the LP formulation above is defined by a constraint matrix where each column has its ones appearing contiguously. Thus the matrix is totally unimodular and hence every basic solution is integral [11]. Thus, the problem can be solved in polynomial time by linear programming. It can also be solved combinatorially as a minimum cost circulation problem [2].

(3) *Tree is a star; Unit capacities:* Next suppose that the tree consists of a star. That is, there are nodes $v, v_1, \dots, v_{n-1}$ and each $vv_i \in E$. Consider the graph G obtained from H by replacing any edge $e = vv_i \in F$ by a leaf edge $v_i v_e$ where v_e is a new node. Then a set of demand edges is routable if and only if they form a matching in the graph G .

(4) *Tree is a star; Arbitrary capacities:* Similar to the above arguments, a set of edges is feasible if and only if they form a b -matching in G , where $b(v_i) = u_{vv_i}$. Thus, we may solve all problems on the star in polynomial time via matching algorithms.

(5) *Arbitrary tree:* This case is a maximum profit integer multicommodity flow problem. The special case where all profits are 1 was studied by Garg, Vazirani, and Yannakakis [5] where they gave a primal-dual algorithm which yields a factor 2 approximation. For the problem with general profit weights, there seems to have been no previous constant approximation known, although Cheriyan, Jordan, and Ravi [4] show that any half-integral solution to (LP) is at most a factor of $\frac{3}{2}$ times the optimal integral solution. In fact, they conjecture that the integrality gap for (LP) is $\frac{3}{2}$ for the unit demand case. We remark that the integrality gap for (LP) is lower bounded by $\frac{3}{2}$ for the unit profit case even when the tree is a star [5].

General Demands: We now consider what happens as we introduce demands to the problem, that is, when we shift from integer multicommodity flows to multicommodity demand flows.

(6) *Tree is an edge:* If the tree is itself a single edge, then the demand flow problem is precisely the knapsack problem. The relaxation (LP) is then well-known to have an integrality gap of 2. A fully polynomial time approximation scheme is also well-known for the knapsack problem. The knapsack problem is at the core of exact methods for solving integer programming since the feasible region is contained in the intersection of multiple knapsack polytopes. It is then important to establish variable forcing relationships and cutting planes for knapsack problems as part of these methodologies. These tasks have normally been carried out on individual knapsacks separately. The demand flow problem can be seen as a collection of knapsack problems (one for each edge of the tree) which share many variables. The study of demand flows in a tree is then a partial response to [14] where study of the interaction of several knapsacks simultaneously is proposed.

(7) *Tree is a star:* In this case, we have the demand matching problem where one is given a graph with capacities b_v for each node, and demands d_e for each edge. A set of edges M is a *demand matching* if for each node v : $\sum_{e \in M \cap \delta(v)} d_e \leq b_v$ ($\delta(v)$ denotes the edges incident to v). This is studied by Shepherd and Vetta

[13] where for instance, it is shown that this problem is MAX-SNP hard and that the integrality gap for (LP) is between 3 and $3\frac{13}{16}$ (2.5 and $2\frac{13}{16}$ for bipartite graphs).

(8) *Tree is a path*: In Chakrabarti et al. [3] an example is given where the supply graph is a path, but the gap between (LP) and the optimum demand flow is $\Omega(\log n)$. They are able to establish, however, that if we restrict to instances where $d_{\max} \leq u_{\min}$, then the integrality gap is $O(1)$. In Figure 1, it is shown that the integrality gap is at least 2.5.

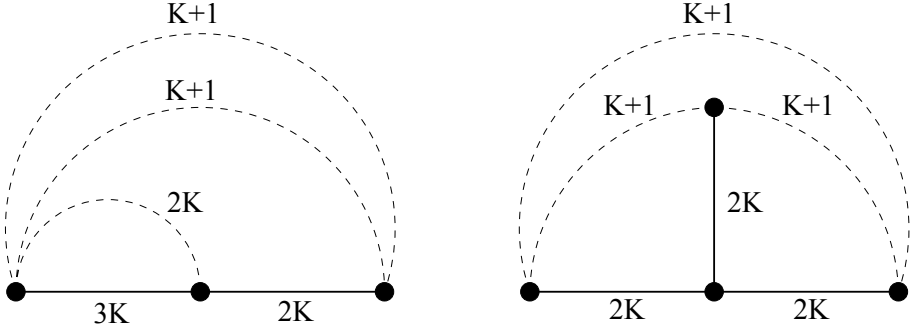


Fig. 1. Integrality gap for line (2.5) and tree (3). Demand graph is shown by dashed lines; all profits are 1.

Contribution of this paper: The present paper has several new contributions. The first result establishes that the natural LP formulation for integer multicommodity flow has a constant factor integrality gap in the case of a tree, that is, for the demand flow problem case where all demands are 1 (see the setting (5)). As mentioned above, a factor of two was already shown for the cardinality case in [5]. In Section 2 we prove the following:

Theorem 1. *Let $T = (V, E, u)$ and $H = (V, F, d, w)$ describe an instance of the maximum profit integer multicommodity flow problem on a tree. Then if (LP) has a feasible solution x of value O , then it has a feasible integral solution z of value at least $\frac{O}{4}$. Moreover, given such an x , we may compute such a z in polynomial time.*

In fact, we show the following stronger result. Let J be any multiset of demands, and k an integer, such that for any edge $e \in T$, at most ku_e of the paths $\{Path(f) : f \in J\}$ contain e . Then J can be partitioned into $4k$ routable demand sets. We remark that the case where all capacities are equal, finding such colourings is substantially simpler. For instance, in the case where all capacities are one, there is a $\frac{3}{2}k$ colouring, as is proved in [10]. Our result is inspired by ideas of Cheriyān, Jordan, and Ravi [4] for the half-integral case. Our stronger colouring result also implies a 4-approximation algorithm on tree networks for a wavelength assignment problem in optical networks. See [7] for more details

of this application of our result. We also discuss extensions of these results to a directed setting where each edge of T is replaced by a pair of oppositely directed arcs (with potentially different capacities).

Our second result is to extend and strengthen results of Chakrabarti et al. [3] on the path, to the general tree case (version (8)). In particular, we show that under the assumption that $d_{\max} \leq u_{\min}$, (LP) has a constant (the constant is about 48) factor integrality gap. This result is proved in Section 3. We obtain this result via a relation between the integrality gap of 0, 1 packing problems and that of their demand version, addressing a question raised in [13]. We use the ideas of Kolliopoulos and Stein [8,9] to explicitly obtain these relations, some of which were either implicit or only qualitatively hinted at in [8,9]. We also apply these ideas to the line and ring networks and obtain $(2 + \epsilon)$ -approximation algorithms, substantially improving the ratios provided in [3]. This connection between 0, 1 packing problems and their demand versions has been missed in recent work [1,2,3] and we believe that one significant contribution of this paper is to bring this to light in the context of natural combinatorial applications.

Demand Flows in General Graphs: Before we proceed with our results, we make some remarks about the problem DFP in general graphs. For our purposes, we focus only on the directed version of the problem. Here we are given a directed supply graph $D = (V, A)$ (we may assume all capacities are 1) as well as a directed demand graph $H = (V, F, d)$ where for each arc $f \in F$, d_f is an integer demand for flow from the tail of f to its head. We are seeking a maximum subset $F' \subseteq F$ such that the multicommodity flow problem on D for the demands of F' is feasible. We mention two variants of this problem: the first is the case where all d_f are 1, denoted by UNIT-DFP. The second is the directed arc-disjoint path problem EDP where the demands d_f are all one, and we require the flows to be integral (i.e., paths). Another variant is the integral splittable version, denoted by ISF, where we require each flow for a demand arc f , to be decomposable into integral flow paths. The problems ISF and EDP are considered in Guruswami et al. [6] and it is shown that for any $\epsilon > 0$ it is NP-hard to compute the optimum for these problems within a factor of $m^{\frac{1}{2}-\epsilon}$, where $m = |A|$. They also point out that the integrality gap for the natural LP may be $\Omega(m)$ if we do not require $d_{\max} \leq u_{\min}$; at the same time, their Proposition 2.2 implies a factor 2 gap (even for the unsplittable flow version) if $d_{\max} \leq u_{\min}$ and each edge determines a demand. It is easy to show that the reduction used for ISF is equally valid for DFP. Namely, they reduce stable set to such an instance. For a, say k -regular, graph G , we create an instance of DFP as follows. Each node v of G gives rise to a demand between s_v, t_v . The digraph D is then obtained as follows. Between each s_v and t_v , there are k node disjoint paths of length three. If v is adjacent to a node u in G , then exactly one of these paths has a middle arc which is also the middle arc for one of the paths from s_u to t_u . Each of these commodities is given a demand of k . Clearly then a routable subset corresponds to a stable set in G .

The case of UNIT-DFP, however, is not yet clear. We remark that the standard instances where there is a gap between EDP and the natural linear relaxation

essentially arise from a grid, where for any pair of commodity paths that cross, there is a unit capacity arc in their intersection [5,6]. Thus, the LP has a solution of value $k/2$ (where k is the number of commodities), while at most one demand can be satisfied integrally. However, for such instances, UNIT-DFP, has a routable subset of size $k/3$; thus the gap between EDP and UNIT-DFP may be large.

2 Unit Demands: Integer Multicommodity Flow

In this section we prove Theorem 1; that is, for any feasible x to (LP) there is an integral solution achieving at least $\frac{1}{4}$ of x 's profit. Coincidentally, we prove the following related approximate-convex combination result for multicommodity flows on a tree.

Theorem 2. *Let x be a feasible solution to (LP) where all d_e 's are 1 and suppose k is an integer such that kx is an integral vector. Then there exist feasible integral solutions $z^1, z^2, \dots, z^{4k}$ such that $kx \leq \sum_i z^i$.*

We now give a proof of Theorem 2. For the rest of the section, we assume without loss of generality that demand edges are incident only on the leaves of the tree. Let k be an integer and x a feasible solution to (LP) where all demands are 1. Furthermore, suppose kx is integral, and let J be a multiset which for each demand edge f contains kx_f copies of f . We give an algorithm which partitions J into $4k$ routable subsets $R_1, R_2, \dots, R_{4k}$. Note that we do not know a priori that k is polynomially bounded, and so this need not give rise to a polynomial time 4-approximate algorithm. However, we may adapt the arguments to find, in polynomial-time, a $\frac{1}{4}$ -optimal integral solution to (LP) for an arbitrary unit demand instance; we describe this at the end of the section. Our algorithm is based on a tree colouring problem that is described below.

An Instance of Binned Tree Colouring: An instance consists of an integer k , a capacitated tree (T, u) , rooted at a fixed leaf node v^* , and a multiset of undirected demand edges J . For each edge $e \in T$, the number of edges of J which lie in e 's fundamental cut is at most ku_e . (The *fundamental cut* of e consists of all edges with their endpoints in each of the two connected components of $T - e$.) In addition, each leaf $v \neq v^*$ has a partition of its incident edges, denoted by $\delta(v)$, into "bins" $B_1(v), \dots, B_{n_v}(v)$ such that $|B_i(v)| \in [k, 2k]$ for each i , and $n_v \leq u_{vp(v)}$ where $p(v)$ is the parent of v in the rooted tree. Our objective is to find a colouring of the edges of J such that each colour class J_i is a routable subset and for each leaf v , and each bin i , the edges of $B_i(v)$ all have different colours. We call such a colouring a *bin colouring* for the instance $T, u, J, \{B_i(v)\}_{i,v}$.

We now prove the following theorem which essentially implies Theorem 1 and 2 (we wrap up the loose details after its proof).

Theorem 3. *Each instance of the binned colouring problem is $4k$ -colourable.*

Proof. Consider the tree T as “hanging down” from the root v^* . We prove the result by induction on the size of T . If T consists of a single edge vv^* then we may colour J as follows. For each bin $B_i(v)$, with edges $e_1, e_2, \dots, e_s$ say, we colour each edge e_j with colour j . Clearly this uses at most $2k$ colours. Moreover, any colour j can occur at most once in each bin, and hence the number of edges of colour j is at most the number of bins n_v . Since each bin is of size at least k , and since $|J| \leq ku_{vv^*}$, we have that at most u_{vv^*} are coloured j , and we are done.

So suppose that T has a remote node v , that is, a node which is not a leaf node, and is adjacent to at most one non-leaf node. Let $v_1, v_2, \dots, v_l$ be the leaf nodes which are adjacent to v . We create a new instance as follows. Our new tree T' is obtained by contracting $\{v_1, v_2, \dots, v_l, v\}$ to a single node which we shall refer to by v' . The set J' is obtained by the same contraction and then dropping any loop edges incident to v' . Finally, we must create the bins for our new leaf v' . Let $B_1, B_2, \dots, B_T$ be the sub-partition of $\delta_J(v')$ (we let $\delta_J(v)$ denote the edges $\delta(v) \cap J$, the demand edges incident to a node v) obtained from the bins of the v_i 's by taking their intersections with the cut $\delta(v')$ in the shrunk graph (and throw out any empty sets). We create the bins for v' greedily as follows. First, make any of the bins of size at least k , new bins also for v' . Now for the remaining bins, start packing them together one by one until a bin of size at least k is obtained. At this point, it is designated a new bin for v (its size is clearly less than $2k$). Let x be the parent of v in T . Since we shrunk the children of v to create v' , x is the parent of v' in T' . Each new bin we create for v' , except perhaps the last one, has size at least k . Hence it follows that $n_{v'} \leq u_{vx} = u_{v'x}$.

Now by the induction hypothesis, we may find a bin colouring for the smaller instance obtained by the above process. We show that this induces a partial colouring of J which can be extended to a bin colouring for our original instance. First, consider some original leaf v_i and the edges $L_i = \delta_J(v_i) \cap \delta_J(v)$ (which are now all coloured). Recall that these edges were originally partitioned into at most $u_{v_i v}$ bins, and each of these bins, was included in some bin of v . Thus any colour could have been assigned to at most $u_{v_i v}$ of the edges in L_i . In particular, this shows that our partial colouring does not violate any of the edge capacity constraints $u_{v_i v}$.

It remains, to complete the colouring on the edges amongst the v_i 's. We now greedily extend the colouring. Suppose that e is an edge joining v_i and v_j which lies in a bin B_i for v_i , and bin B_j for v_j . Call a colour *used* at B_i if some edge in B_i has already been assigned this colour. There are at most $2k - 2$ such colours used since $|B_i| < 2k$. It follows that there are at least 4 colours which are not used in either B_i or B_j . Assign e one of these colours. After we complete this process we obtain a colour which satisfies all of the bin constraints. Each colour class is a routable set of demands, since by induction, the load on any non-leaf edge is at most its capacity. And for any leaf edge say v_i , the number of edges of one colour, is at most the number of bins it has. Since each bin is of size at least k , this is at most $u_{v_i v}$. $\square$

We may apply this theorem to obtain Theorem 2 as follows. The only minor point that we must address is to make sure that none of the $4k$ routable sets

contains any demand edge more than once (since (LP) has an upper bound constraint for each demand). One easily adapts the induction hypothesis to make sure this is the case. In particular, we restrict our leaf bins, so that for any demand edge $f = uv$, all copies of f occur in the same bin at u , and the same bin at v . Since there are at most k copies of any such f , one sees that this is always possible.

We mention that the Binned Tree Colouring Problem could also be defined for *bidirected trees*. These arise in directed multicommodity flow problems where the supply graph is obtained from a tree by replacing each edge e , by a pair of oppositely directed arcs, each with its own capacity. The (directed) demand graph then consists of arcs $f = (u, v)$ where there is a request for a single unit of flow to be sent along the unique directed path $P(f)$ from u to v . The *load* of a set of demands J on an arc a in a bidirected tree T , is $|\{f \in J : a \in P(f)\}|$. A set of demands J is *feasible* for a bidirected tree T with arc capacities u_a , if for each arc a , u_a is at least the load of J on a . Here again, one may use the same induction procedure from the proof of Theorem 3 to show the following.

Theorem 4. *Let T be a bidirected tree with capacities u_a on each arc. If J is a set of directed “demand” arcs which imposes a load of at most ku_a on each arc of T , then J may be partitioned into $4k$ subsets each of which is routable on T .*

We now return to the proof of the final claim in Theorem 1, to find a polynomial time 4-approximate algorithm. We give a comprehensive sketch of the argument below. To this end, suppose that x is a basic feasible solution for (LP). We process x in a manner similar to the proof of Theorem 2, in order to create a routable set with at least $\frac{1}{4}$ the profit of x . As in Theorem 3, we prove something stronger. Namely, we again root the tree at an internal node and think of the tree dangling downwards, with the leaves at the lower levels. In addition, each leaf has its incident demand edges partitioned into bins. Note that bins in this context do not contain multiple copies of a demand edge as in the proof of Theorem 3. Each bin B has the property that $x(B) = \sum_{e \in B} x_e < 2$; further, the number of bins at each leaf is at most the capacity of the edge incident on it. Clearly, we may always create such a “binning” at the leaves. We prove inductively that there exist integral feasible solutions r^i to (LP) such that

1. $r^i(B) \leq 1$ for each bin B , that is, $r_j^i = 1$ for at most one $e_j \in B$ (at most one edge in B is “in” r^i),
2. $x = \sum_i \lambda_i r^i$ for some choice of λ_i ’s with $\sum_i \lambda_i = 4$.

In fact, we show that if the size of the support of x is q , then we need at most $q + 1$ r^i ’s. In particular, if x is a basic solution, we need at most $n + 1$ to obtain our convex combination of the vector $\frac{x}{4}$. In this case, the combination is produced in polynomial time.

The base case is similar to before, we have a tree with a single edge vv^* . Suppose there are q demand edges $e_1, e_2, \dots, e_q$, and suppose that the bins for v are $B_1, B_2, \dots, B_t$ with $t \leq u_{vv^*}$. We construct the integral solutions greedily as follows. For $i = 1, 2, \dots$ perform the following to create a solution r^i . For

$j = 1, 2, \dots, t$ if B_j is nonempty, then add exactly one edge $e_h \in B_j$, chosen arbitrarily, to r^i , i.e., set $r_h^i = 1$. Once we have looked at all t bins, we set λ_i to be $\min_{g: r_g^i=1} x_{e_g}$. We then reduce the fractional value of each edge assigned to r^i by the amount λ_i . Any edge whose x value is reduced to 0, is deleted from consideration. We repeat this process until all bins are empty. We may obviously associate each r^i to a unique demand edge which was deleted upon construction of r^i . Thus the total number of r^i 's constructed in this process is at most q . Also note that $\sum_i \lambda_i \leq 2$, since at each stage i , the “size” of a maximum bin (i.e., $x(B)$ for a bin B) is reduced by λ_i . Finally, we may add a last solution r^{q+1} , say, corresponding to the empty routable set, and assign this a value $4 - \sum_i \lambda_i$.

The induction step is also similar to before. We consider contraction of a remote node which gives rise to a new leaf v' . The bins at v' are constructed from the partial bins from its descendant leaves. By induction, we obtain vectors r^i and multipliers λ_i in the smaller tree. A simple argument shows that each r^i is a feasible integral solution for the original instance as well.

We now extend this combination back to the original graph. In doing so, we must account for the missing demand edges in the original graph (i.e., demand edges joining two of the original leaves). That is, we must extend the combination to satisfy the bin constraints at these leaves. We can incorporate these demand one at a time, increasing the number of r^i 's by at most one each time. So let $e_\ell = \alpha\beta$ be a demand edge between two leaves α and β which were shrunk in the reduction. Let B_α, B_β be the two bins which contained the edge e_ℓ . We may scan the solutions $r^1, r^2, \dots$ in order. Each time, if adding e_ℓ to the solution r^i destroys the bin condition (that is $r^i(B) = 1$ already), we move on. Otherwise, e_ℓ can be added to r^i . So we set $r_\ell^i = 1$ (add e_ℓ to that solution). Now if $x_{e_\ell} > \lambda_i$, then set $x_{e_\ell} = x_{e_\ell} - \lambda_i$ (reduce e_ℓ 's demand in our “running” fractional solution). Otherwise, divide r^i into two solutions: one $r^{i,1}$ with $\lambda_{i,1} = x_{e_\ell}$, and the second copy with $r_\ell^{i,2} = 0$, and $\lambda_{i,2} = \lambda_i - x_{e_\ell}$. Note that this procedure can only increase the number of r^i 's once per each demand edge e_ℓ ; on the iteration where x_{e_ℓ} is finally reduced to 0. The last question is whether we could process all the r_i 's without fully covering an x_{e_ℓ} . Note that e_ℓ cannot be added to some r^i only if $r^i(B) = 1$ for $B = B_\alpha$ or $B = B_\beta$; in this case we call i bad for e_ℓ . But we have that $x(B_\alpha) + x(B_\beta) < 4$, by choice. Also, $\sum_{i \text{ bad}} \lambda_i \leq x(B_\alpha) + x(B_\beta) - x_{e_\ell} \leq 4 - x_{e_\ell}$. In other words, $\sum_{i \text{ not bad}} \lambda_i \geq x_{e_\ell}$, and hence we have enough room to add e_ℓ to the convex combination.

3 Arbitrary Demands

In this section we consider the case of multicommodity demand flows on a tree with arbitrary demands, with the assumption that $d_{\max} \leq u_{\min}$. We obtain this result via a more general one which shows that the integrality gap for a general “demand” problem is related by a constant to the unit demand version of the problem.

3.1 Column Restricted Packing Integer Programs

In this section we consider certain classes of packing problems that arise from 0, 1 matrices as follows. In the following, let A be a 0, 1 matrix with m rows and n columns. For an integer vector $d \in \mathbf{R}_+^n$, we denote by $A[d]$ the matrix obtained from A by multiplying each entry in column i by d_i . We restrict attention to the *column-restricted packing integer programs* (CPIP), introduced by Kolliopoulos and Stein [9]. Each such problem is of the form $\max\{wx : A[d]x \leq b, x \in \{0, 1\}^n\}$ for some choice of integer vectors w, d, b .

These column-restricted pips model the outcome of altering the original packing problem $\max\{wx : Ax \leq b, x \in \{0, 1\}^n\}$, by adding demand values d_i to the items (columns) being packed. In generalizing their own techniques from [8], Kolliopoulos and Stein [9] devise a grouping and scaling technique to show that the integrality gap for such CPIPs are “of similar quality to those” for the 0, 1 problems. Their main objective is to establish bounds for general column-restricted packing problems. In contrast, our thrust is to examine special classes of CPIPs. As such, we first use their ideas to explicitly relate the integrality gap of column-restricted PIPs as a function of the gap for the underlying 0, 1 PIP problems (also answering a question independently raised in [13]). We also indicate more general scenarios where these ideas hold. We later apply theorems to concrete packing problems: these applications have been missed by several papers in the recent past [1, 2, 3, 13].

We now formalize some of the concepts required. For a convex body P over $\mathbf{R}^n$ and objective vector $w \in \mathbf{R}^n$, the *integrality gap* for the optimization problem $\gamma = \max\{wx : x \in P\}$ is the ratio between the fractional optimum γ and the optimal value of an integral solution, that is, $\frac{\gamma}{\max\{wx : x \in P_I\}}$. Here P_I denotes the *integer hull* of P , that is the convex hull of all integer vectors in P .

We are interested in bounding the integrality gap for classes of integer programs. Each class $\mathcal{P}$ consists of problems induced by pairs P, w where P, w lie in some fixed space $\mathbf{R}^n$. The *integrality gap* for such a class is simply the supremum of the integrality gaps for individual problems in $\mathcal{P}$.

A collection of vectors $W \subseteq \mathbf{Z}^n$ is *closed* if for any vector $w \in W$, the vector w' obtained by setting some $w_j = 0$ is also in W . In the following, for a matrix A and closed collection W we denote by $\mathcal{P}(A, W)$ the class of problems of the form $\max\{wx : Ax \leq b\}$ for some $w \in W$ and vector $b \in \mathbf{Z}_+^m$. We then denote by $\mathcal{P}^{dem}(A, W)$ the class of problems of the form $\max\{wx : A[d]x \leq b\}$ for some $w \in W$, and vectors $b \in \mathbf{Z}^m, d \in \mathbf{Z}_+^n$ with $d_{max} \leq b_{min}$.

We then have the following result whose proof follows precisely the lines of analysis used by Kolliopoulos and Stein in their study of unsplittable flow [8]. We obtain a slightly better constant than they do because of a small error in their final calculation.

Theorem 5. *Let A be a 0, 1 matrix and W be a closed collection of vectors. If the integrality gap for the collection of problems $\mathcal{P}(A, W)$ is at most Γ , then the integrality gap for the collection of problems $\mathcal{P}^{dem}(A, W)$ is at most $11.542\Gamma \leq 12\Gamma$.*

To prove the above theorem we set up some notation. Let $\Pi(A, b, w)$ be a 0,1 packing problem of the form $\max\{wx : Ax \leq b\}$ from $\mathcal{P}(A, W)$ and let $\Pi(A, b, d, w)$ be a problem of the form $\max\{wx : A[d]x \leq b\}$ from $\mathcal{P}^{dem}(A, W)$. Given a subset S of $\{1, 2, \dots, n\}$, we denote by A^S the matrix A restricted to the columns in S . Given S , we have two naturally defined new problems $\Pi(A^S, b, w^S)$ and $\Pi(A^S, b, d^S, w^S)$. Given a fractional solution x for $\Pi(A, b, w)$, its restriction to S is denoted by x^S .

In the following we use two parameters α and β where $\beta < 1/2$ and $\alpha + \beta \leq 1$. We optimize these parameters at the end to obtain the best ratio. It is useful to have the setting $\alpha = \beta = 1/3$ in mind to follow the proof. We call a demand f *large* if $d_f \geq \beta d_{\max}$, otherwise a demand is called *small*. We show first how one may obtain at least $\frac{\beta}{2\Gamma}$ times the fractional profit (under x) of the large demands.

Lemma 1. *Let S be the set of large demands. Given a fractional solution x to $\Pi(A, b, d, w)$ there is an integral solution to $\Pi(A^S, b, d^S, w^S)$ of value at least $\frac{\beta}{2\Gamma} \sum_{f \in S} w_f x_f$.*

Proof. Given x , S and $\Pi(A^S, b, d^S, w^S)$ we create a new instance $\Pi(A^S, c, \hat{d}, w^S)$ and feasible fractional solution y^S as follows. First $\hat{d}$ denotes the vector in S -space with each component equal to $d_{\max}$; this has the effect of uniformly setting all the demands to be $d_{\max}$. We set $y^S = \frac{\beta}{2} x^S$. For $1 \leq j \leq m$, we set $c_j = \lfloor b_j / d_{\max} \rfloor d_{\max}$, in other words we set c_j to be the largest integer multiple of $d_{\max}$ not exceeding b_j . We observe two easy facts. First, the solution y^S is feasible for the instance $\Pi(A^S, c, \hat{d}, w^S)$. Second, any feasible integral solution to $\Pi(A^S, c, \hat{d}, w^S)$ translates into a feasible solution of the same value to $\Pi(A^S, b, d^S, w^S)$. These two observations combined with the fact that $\Pi(A^S, c, \hat{d}, w^S)$ is a uniform demand instance, and hence has an integrality gap of at most Γ , yields the lemma. $\square$

Now we address the small demands.

Lemma 2. *Let S be the set of small demands. Given a fractional solution x to $\Pi(A, b, d, w)$ there is an integral solution to $\Pi(A^S, b, d^S, w^S)$ of value at least $\alpha(1 - \frac{\beta}{1-\alpha}) \frac{1}{\Gamma} \sum_{f \in S} w_f x_f$.*

Proof. For $t \geq 0$, let S_t be the subset of small demands f such that $d_f \in (\alpha^{t+1}\beta d_{\max}, \alpha^t\beta d_{\max}]$. For each t we construct a new instance $\Pi(A^{S_t}, c^t, d^t, w^{S_t})$ and a feasible fractional solution y^t in S_t -space, as follows. For $f \in S_t$, we set $y_f^t = \alpha(1 - \frac{\beta}{1-\alpha}) x_f$ and we set $d_f^t = \alpha^t\beta d_{\max}$. We define the *load* on constraint i from demands in S_t in x , denoted by ℓ_i^t as $\sum_{f \in S_t} A_{if} d_f x_f$. We set c_i^t to be the smallest integer multiple of $\alpha^t\beta d_{\max}$ larger than $(1 - \frac{\beta}{1-\alpha})\ell_i^t$. Note that $c_i^t \leq (1 - \frac{\beta}{1-\alpha})\ell_i^t + \alpha^t\beta d_{\max}$.

By construction $\Pi(A^{S_t}, c^t, d^t, w^{S_t})$ is a uniform demand problem. It is easily verified that y^t is a feasible solution for this instance. Hence, by our assumption

on the integrality gap of the 0, 1 instances, there exists an integral solution z^t to $\Pi(A^{S_t}, c^t, d^t, w^{S_t})$ of value at least $\frac{1}{F} \sum_{f \in S_t} w_f y_f^t = \frac{1}{F} \alpha (1 - \frac{\beta}{1-\alpha}) \sum_{f \in S_t} w_f x_f$.

We now argue that combining the solutions z^t into one single solution z gives a feasible integral solution to $\Pi(A^S, b, d^S, w^S)$ of value at least $\frac{1}{F} \alpha (1 - \frac{\beta}{1-\alpha}) \sum_{f \in S} w_f x_f$. From the analysis in the previous paragraph, the value of z is at least as much as we claim. We show that that z is feasible. Consider an arbitrary constraint i : since z^t is feasible for $\Pi(A^{S_t}, c^t, d^t, w^{S_t})$ it follows that $\sum_{f \in S_t} A_{if} d_f^t z_f^t \leq c_i^t$. By construction, for $f \in S_t$, $d_f \leq d_f^t = \alpha^t \beta d_{\max}$, therefore we have that

$$\sum_{f \in S_t} A_{if} d_f z_f^t \leq c_i^t \leq (1 - \frac{\beta}{1-\alpha}) \ell_i^t + \alpha^t \beta d_{\max}.$$

Hence the load on constraint i in the combined solution z is at most $\sum_{t \geq 0} ((1 - \frac{\beta}{1-\alpha}) \ell_i^t + \alpha^t \beta d_{\max})$ which is at most $(1 - \frac{\beta}{1-\alpha}) \sum_t \ell_i^t + \frac{\beta}{1-\alpha} d_{\max}$. By the feasibility of x , $\sum_t \ell_i^t \leq b_i$. We also have that $d_{\max} \leq b_i$, therefore we get that load on constraint i by z is at most b_i . This shows that z is a feasible integral solution. $\square$

In the proof of the above lemma we scale the fractional solution x by $\alpha(1 - \frac{\beta}{1-\alpha})$. The factor α accounts for rounding up the demands in S_t to $\alpha^t \beta d_{\max}$. The factor $(1 - \frac{\beta}{1-\alpha})$ is to make room for the additional capacity to the tune of $\alpha^t \beta d_{\max}$ that we add to each instance $\Pi(A^{S_t}, c^t, d^t, w^{S_t})$ to make the capacities integral multiples of $\alpha^t \beta d_{\max}$.

We complete the proof of Theorem 5. Let L denote the set of large demands and S denote the set of small demands. From Lemma 1 we obtain an integer solution of value at least $\frac{\beta}{2F} \sum_{f \in L} w_f x_f$. From Lemma 2 we obtain an integer solution of value least $\alpha(1 - \frac{\beta}{1-\alpha}) \frac{1}{F} \sum_{f \in S} w_f x_f$. For a given β it is easy to verify that the expression $\alpha(1 - \frac{\beta}{1-\alpha})$ is maximized when $\alpha = 1 - \sqrt{\beta}$, hence for small demands we obtain an integer solution of value at least $(1 - 2\sqrt{\beta} + \beta) \frac{1}{F} \sum_{f \in S} w_f x_f$.

Let $\epsilon \in [0, 1]$ be defined by the equation $\sum_{f \in S} w_f x_f = \epsilon \sum_f w_f x_f$, in other words ϵ is the fraction of the total weight of small demands in the fractional solution. From the above analysis we are guaranteed an integral solution of value $\max\{\epsilon(1 - 2\sqrt{\beta} + \beta), (1 - \epsilon)\beta/2\} \frac{1}{F} \sum_f w_f x_f$. The algorithm can choose β to maximize this expression but has no control over the distribution of ϵ . Hence we can lower the bound the integrality gap by the expression $\max_{\beta < 1/2} \min_{0 \leq \epsilon \leq 1} \{\epsilon(1 - 2\sqrt{\beta} + \beta), (1 - \epsilon)\beta/2\} \frac{1}{F}$. Numerical computation shows that this expression is at least $\frac{1}{11.542F}$. Hence the integrality gap is at most $11.542F$. Setting $\alpha = \beta = 1/3$ yields a simple analysis that shows that the integrality gap is at most $12F$. $\square$

We now give useful corollaries of the above theorem. It is natural to expect that the integrality gap will tend to 1 as $d_{\max}/b_{\min} \rightarrow 0$. We denote by $\mathcal{P}^{edem}(A, W)$ the class of column restricted packing problems such that $d_{\max} \leq \epsilon b_{\min}$.

Corollary 1. *Let A be a $0, 1$ matrix and W be a closed collection of vectors. If the integrality gap for the collection of problems $\mathcal{P}(A, W)$ is at most Γ , then the integrality gap for the collection of problems $\mathcal{P}^{\text{edem}}(A, W)$ for $\epsilon < (3 - \sqrt{5})/2$, is at most $\frac{1+\sqrt{\epsilon}}{1-\sqrt{\epsilon}-\epsilon}\Gamma$ (upper bounded by $(1 + O(1)\sqrt{\epsilon})\Gamma$). This holds even under the weaker condition that $\max_j A_{ij}d_j \leq \epsilon b_i$, for each $i = 1, 2, \dots, m$.*

Proof. Follows from Lemma 2 with $\beta = \epsilon$ and $\alpha = \frac{1}{1+\sqrt{\epsilon}}$. An examination of the proof of Lemma 2 shows that the lemma holds even if we are only guaranteed that $\max_j A_{ij}d_j \leq \beta b_i$, for each $i = 1, 2, \dots, m$. $\square$

There are examples of packing problems (notably the ring [12]) for which relaxing the capacity constraints by an additive constant independent of the input parameters yields an improved integrality gap. For a $0, 1$ packing problem $\max\{wx : Ax \leq b, x \in \{0, 1\}^n\}$, constant c , the c -relaxed integrality gap is Γ if the value of the optimum solution to the relaxed problem $\max\{wx : Ax \leq b + \hat{c}, x \in \{0, 1\}^n\}$ is at least $1/\Gamma$ times the value of the (fractional) solution to $\max\{wx : Ax \leq b, x \in [0, 1]^n\}$, where $\hat{c}$ denotes the m -vector with all components equal to c .

Corollary 2. *Let A be a $0, 1$ matrix and W be a closed collection of vectors. If the c -relaxed integrality gap for the collection of problems $\mathcal{P}(A, W)$ is at most Γ , then the integrality gap for the collection of problems $\mathcal{P}^{\text{edem}}(A, W)$ for $\epsilon < \frac{1}{4(1+c)^2}$, is at most $\frac{1+\sqrt{\epsilon}}{1-(1+c)(\sqrt{\epsilon}+\epsilon)}\Gamma$ (upper bounded by $(1 + O(1)(1+c)\sqrt{\epsilon})\Gamma$). This holds even under the weaker condition that $\max_j A_{ij}d_j \leq \epsilon b_i$, for each $i = 1, 2, \dots, m$.*

Proof. The proof follows closely the proof of Lemma 2. In Lemma 2 the fractional solution is scaled down by a factor of $\alpha(1 - \frac{\beta}{1-\alpha})$. As we remarked, the factor $(1 - \frac{\beta}{1-\alpha})$ is to make room for the extra capacity we add to the sub problems generated by demands in S_t . Since we work with c -relaxed integrality gap, we need to add an additional capacity of $c\alpha^i\beta d_{\max}$ to the subproblem t . Hence we need a scaling factor of $\alpha(1 - \frac{(1+c)\beta}{1-\alpha})$ to accommodate this extra space. By choosing $\alpha = \frac{1}{1+\sqrt{\epsilon}}$ and $\beta = \epsilon$ we get the desired result. $\square$

Corollary 3. *Let A be a $0, 1$ matrix and W be a closed collection of vectors. If the integrality gap for the collection of problems $\mathcal{P}(A, W)$ is at most Γ , then for every problem $\mathcal{P}^{\text{dem}}(A, W)$ and for every $\beta > 1$ there is an integral solution to $\max\{wx : A[d]x \leq \beta b + \frac{\beta}{\beta-1}d_{\max}\}$ of value at least $1/\Gamma$ times the value of the optimum fractional solution to $\max\{wx : A[d]x \leq b\}$. This holds even under the condition that $\max_j A_{ij}d_j \leq b_i$, for each $i = 1, 2, \dots, m$.*

Proof. The proof follows along the lines of that for Lemma 2, however, we allow the capacities to be violated but do not scale down the solution x . Let $\alpha = 1/\beta$. For $t \geq 0$, let S_t be the set of demands f such that $d_f \in (\alpha^{t+1}d_{\max}, \alpha^t d_{\max}]$. We create a new instance $\Pi(A^{S_t}, c^t, d^t, w^{S_t})$ as follows. For $f \in S_t$, we set

$d_f^t = \alpha^t d_{\max}$. We define the *load* on constraint i from demands in S_t in x , denoted by ℓ_i^t as $\sum_{f \in S_t} A_{if} d_f x_f$. We set c_i^t to be the smallest integer multiple of $\alpha^t d_{\max}$ larger than ℓ_i^t / α . Note that $c_i^t \leq \ell_i^t / \alpha + \alpha^t d_{\max}$. We observe that the fractional solution x^{S_t} is feasible for $\Pi(A^{S_t}, c^t, d^t, w^{S_t})$.

As before we obtain integral solutions z^t for each of the above instances and combine them to obtain a solution z . Since we did not scale down the fractional solution and the integrality gap of each of the subproblems is at most Γ , the value of z is at least $1/\Gamma$ times the value of $\sum_f w_f x_f$. It remains to show that for $i \in 1, 2, \dots, m$, $\sum_f A_{if} d_f z_f \leq \beta b_i + \frac{\beta}{\beta-1} d_{\max}$.

The solution z^t satisfies the capacity constraints for $\Pi(A^{S_t}, c^t, d^t, w^{S_t})$. Hence it follows that z satisfies the capacity constraints defined by $\sum_t c^t$ which by construction is dominated by $\beta b + \frac{\beta}{\beta-1} d_{\max}$. $\square$

In applications to unsplittable flows, we need a minor refinement of the previous results. Let $\mathcal{V} = V_1 \cup V_2 \cup \dots \cup V_t$ be a partition of the column variables. A collection of vectors $W \subseteq \mathbf{Z}^n$ is *closed with respect to* $\mathcal{V}$ if for any vector $w \in W$ and any V_i , the vector w' obtained by setting $w_j = 0$ for all $j \in V_i$ is also in W . For such a partition and closed collection W , we denote by $\mathcal{P}^{dem}(A, W, \mathcal{V})$ the class of column restricted packing problems in $\mathcal{P}^{dem}(A, W)$ arising from demand vectors d with the property that $d_i = d_j$ for all $i, j \in V_q$ for some q . The previous proof immediately extends to yield:

Theorem 6. *Let A be a 0,1 matrix, $\mathcal{V}$ a partition of its columns and W be a closed collection of vectors over $\mathcal{V}$. If the integrality gap for the collection of problems $\mathcal{P}(A, W)$ is at most Γ , then the integrality gap for the collection of problems $\mathcal{P}^{dem}(A, W, \mathcal{V})$ is at most $(1 + \sqrt{6})^2 \Gamma \leq 12\Gamma$.*

Finally we remark that for *uniform* capacity problems, that is all entries in b are identical, improved results can be obtained. We defer the details.

3.2 Applications to Combinatorial Demand Problems

We now briefly discuss some applications of results in the previous section to combinatorial problems.

Tree: Our original task was to show that the natural LP formulation for the multicommodity demand flow problem has an $O(1)$ integrality gap for instances where $d_{\max} \leq u_{\min}$ and the supply graph is a tree. Theorems 1 and 5 now imply that the integrality gap is indeed at most 48. Moreover, we find in polynomial time, an integral solution delivering at least $\frac{1}{48}$ times the profit of the optimal fractional solution.

Line: When the supply graph is a line (path), the demand problem has been studied for its application to resource allocation [1,2,3]. In [2] a $2 + \epsilon$ -approximation is provided for the uniform capacity problem improving the 3-approximation in [1]. The main observation in [2] is that when $d_{\max} \leq \epsilon U$ where U is the common capacity of the edges the integrality gap of the LP is

$1/(1 - O(\sqrt{\epsilon \ln 1/\epsilon}))$: this is proved by an interesting use of randomized rounding with alteration. In [3] this approach is extended to the non-uniform capacity case and an $O(1)$ approximation is presented. Corollary 1 applies to both the uniform capacity and the non-uniform capacity problem in a rather simple way where the integrality gap of the underlying packing problem is 1. One immediate consequence is a $(2 + \epsilon)$ -approximation for the non-uniform capacity line, substantially improving the constant provided in [3]. We may view the problem of the line as a special case of packing directed paths, each path with its own profit, within some capacitated directed tree. This path packing problem can be solved via a totally unimodular matrix, and hence the demand version has a 12-approximation via Theorem 5.

Ring: In [3] the algorithm for the line is extended to case of the ring network. It is shown that an α -approximation for the line yields an $(\alpha + 1)$ -approximation for the ring. Here we indicate how to obtain a $(2 + \epsilon)$ -approximation algorithm. For the ring it is known [12] that the $1/2$ -relaxed integrality gap is 1. Using the version of Corollary 2 of Theorem 6 we can obtain a $(1 + \epsilon)$ -approximation for demands that are small (smaller than $O(\epsilon^2)u_{\min}$). For large demands a combination of enumeration and dynamic programming, using ideas similar to those in [3], yields an optimal algorithm. Combining these two algorithms yields a $(2 + \epsilon)$ -approximation.

Arborescences: Let D be a digraph with arc capacities u and weights w and specified node s . We may find a maximum w -weight packing into u of arborescences rooted at s via linear programming. Namely, if A is the $0, 1$ matrix with a column for each arborescence, and row for each arc, then $\max\{wx : Ax \leq u, x \geq 0\}$ has an integral optimum for each integer u . We then also obtain a factor 12 integrality gap for a version where each arborescence T has a demand $d(T)$. Only certain classes of these demand problems can be solved in polynomial time. Specifically, we may solve the problem if we restrict to demand assignments that are induced by link values d_a : $d(T) = \sum_{a \in T} d_a$, since in this case the separation problem for the dual LP is a shortest arborescence problem.

References

1. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, B. Schieber. A unified approach to approximating resource allocation and scheduling. *JACM*, 48(5), 1069–90, 2001. Preliminary version in *Proc. of STOC* 2000.
2. G. Calinescu, A. Chakrabarti, H. Karloff, Y. Rabani. Improved Approximation Algorithms for Resource Allocation. *Proc. of IPCO*, 2002.
3. A. Chakrabarti, C. Chekuri, A. Gupta, A. Kumar. Approximation Algorithms for the Unsplittable Flow Problem. *Proc. of APPROX*, 2002.
4. J. Cheriyan, T. Jordan, R. Ravi. On 2-coverings and 2-packings of laminar families. *Proc. of ESA*, 1999.
5. N. Garg, V. Vazirani, M. Yannakakis. Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees. *Algorithmica*, 18(1):3–20, 1997. Preliminary version appeared in *Proc. of ICALP*, 1993.

6. V. Guruswami, S. Khanna, R. Rajaraman, F. B. Shepherd, M. Yannakakis. Near-Optimal Hardness Results and Approximation Algorithms for Edge-Disjoint Paths and Related Problems, To appear in: *JCSS*. Preliminary version appeared in *Proc. of STOC*, 1999.
7. T. Erlebach, A. Pagourtzis, K. Potika, S. Stefanakos. Resource allocation problems in Multifiber WDM Tree Networks. *Manuscript*, March 2003.
8. S. G. Kolliopoulos, C. Stein, Approximation Algorithms for Single-Source Unsplittable Flow, *SIAM J. Computing* (31), 919–946, 2002; preliminary version in *Proc. of FOCS*, 1997.
9. S. G. Kolliopoulos C. Stein. Approximating Disjoint-Path Problems using Packing Integer Programs, *Proc. of IPCO*, 1998.
10. P. Raghavan, E. Upfal. Efficient routing in all-optical networks. *Proc. STOC*, 1994.
11. A. Schrijver. Theory of Linear and Integer Programming. John Wiley and Sons, 1986.
12. F.B. Shepherd, Lisa Zhang. An Augmentation Algorithm for Mincost Multicommodity Flow on a Ring, *Discrete Applied Mathematics* 110, 2001, 301–315.
13. F.B. Shepherd, A. Vetta. The Demand Matching Problem. *Proc. of IPCO*, 2002.
14. L. Wolsey, private communication, Oberwolfach, 2002.

Skew and Infinitary Formal Power Series

Manfred Droste and Dietrich Kuske*

Institut für Algebra, Technische Universität Dresden, D-01062 Dresden, Germany
{droste,kuske}@math.tu-dresden.de

Abstract. We investigate finite-state systems with costs. Departing from classical theory, in this paper the cost of an action does not only depend on the state of the system, but also on the time when it is executed. We first characterize the terminating behaviors of such systems in terms of rational formal power series. This generalizes a classical result of Schützenberger.

Using the previous results, we also deal with nonterminating behaviors and their costs. This includes an extension of the Büchi-acceptance condition from finite automata to weighted automata and provides a characterization of these nonterminating behaviors in terms of ω -rational formal power series. This generalizes a classical theorem of Büchi.

1 Introduction

In automata theory, Kleene's fundamental theorem [17] on the coincidence of regular and rational languages has been extended in several directions. Schützenberger [26] showed that the formal power series (cost functions) associated with weighted finite automata over words and an arbitrary semiring for the weights, are precisely the rational formal power series. Weighted automata have recently received much interest due to their applications in image compression (Culik II and Kari [6], Hafner [14], Katritzke [16], Jiang, Litow and de Vel [15]) and in speech-to-text processing (Mohri [20], [21], Buchsbaum, Giancarlo and Westbrook [4]).

On the other hand, Büchi [3] extended Kleene's result to languages of infinite words, showing that finite automata recognize precisely the ω -rational languages. This result stimulated a huge amount of more recent research on automata acting on various infinite structures, and Büchi-automata are used for formal verification of reactive systems with infinite processes. For theoretical background on formal power series, we refer the reader to [25,19,1,18], and for background on automata on infinite words to [27,23].

In this paper, we wish to extend Büchi's and Schützenberger's approaches to weighted automata on infinite words. Whereas Schützenberger's result for automata on finite words works for weights taken in an arbitrary semiring, it is clear that for weighted automata on infinite words questions of summability and convergence arise. Therefore we assume that the weights are taken in the

* This work was done while the second author worked at the University of Leicester.

non-negative real numbers, endowed with maximum and addition as operations. This max-plus semiring of real numbers is fundamental in max-plus algebra and algebraic optimization (Gaubert and Plus [13], Cuninghame-Green [7]), and related semirings also occurred in other investigations on formal power series (e.g., [18,8,9]). We note that a different approach of weighted automata acting on infinite words has been considered before in connection with digital image processing by Culik and Karhumäki [5].

We will introduce the concept of automata acting on infinite words and with weights in the real max-plus semiring. Their behaviour is described by the function associating to each word the cost the automaton needs for evaluating it. However, here the arising infinite sums of weights of the transitions in an infinite computation sequence usually diverge. In order to enforce their convergence, we introduce a deflation parameter $q \in [0, 1)$. That is, we assume that in a computation sequence, the cost of a later transition is decreased by multiplication with a power of q . This is a usual mathematical procedure in order to obtain convergence of series. It even enables one to compare their “rate of former divergence”. Moreover, here somewhat surprisingly it also reflects the usual human evaluation practices in which later events are considered less urgent and carry less weight than close events. Note that multiplication with a nonnegative real constitutes an endomorphism of the max-plus semiring.

Therefore we derive, as our first new result, a generalization of Schützenberger’s classical result on automata on finite words, where the weights are taken in an arbitrary semiring, but now changed along computation sequences by a given endomorphism. In fact, we show that also under this notion several different concepts of automata investigated before in the literature again coincide. If the endomorphism is the identity, we obtain Schützenberger’s theorem as a particular case. This result is of independent interest, since such skew multiplications have been considered in the area of Ore series in difference and differential algebra, cf. [22,12,11,2]. We prove analogues of classical preservation theorems for homomorphisms between different alphabets or semirings. We also show that when considering the max-plus semiring and multiplication with reals as endomorphisms, then different numbers yield indeed different collections of recognizable series.

Then we turn to automata on infinite words with weights in the real max-plus semiring as described above. The ω -recognizable series are those which can be obtained as the behaviour of a finite weighted automaton acting on infinite words. We define rational operations on series over infinite words like sum and skew product, Kleene iteration and ω -iteration. The ω -rational series then are those which can be obtained by these operations from the monomials.

Our second main result states that ω -recognizable and ω -rational formal power series over the real max-plus semiring with deflation parameter q coincide, for each $q \in [0, 1)$. We show that from this one can obtain Büchi’s classical result on the coincidence of ω -recognizable and ω -rational languages as a consequence. This is essentially due to the fact that the Boolean semiring can be naturally embedded into the (idempotent) max-plus semiring.

Due to length restrictions, many proofs had to be omitted from this extended abstract, the interested reader is referred to the technical report [10] for a complete version. For a recent study on Büchi-automata with weights in bounded distributive lattices, see [24].

2 Weighted Automata

First let us recall some background on semirings; see also [25,19,1,18].

A structure $(K, \oplus, \odot, 0, 1)$ is a *semiring* if $(K, \oplus, 0)$ is a commutative monoid, $(K, \odot, 1)$ is a monoid, $\odot$ is both left- and right-distributive over $\oplus$, and $0 \odot x = x \odot 0 = 0$ for any $x \in K$. If no confusion can arise, we will denote a semiring just by K . Important examples include

- the natural numbers $(\mathbb{N}, +, \cdot, 0, 1)$ with the usual addition and multiplication,
- the Boolean semiring $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$,
- the tropical semiring $\mathbb{R}_{\max} = (\mathbb{R}_{\geq 0} \cup \{-\infty\}, \max, +, -\infty, 0)$ (that is also known as max-plus semiring) with $\mathbb{R}_{\geq 0} = [0, \infty)$ and $-\infty + x = -\infty$ for each $x \in \mathbb{R}_{\max}$. Observe that in this semiring $-\infty$ acts as zero, i.e., neutrally with respect to $\max$, and 0 as one, i.e., neutral with respect to $+$.

A mapping $\varphi : K_1 \rightarrow K_2$ between two semirings K_1 and K_2 is called *homomorphism* if $\varphi(x \oplus y) = \varphi(x) \oplus \varphi(y)$ and $\varphi(x \odot y) = \varphi(x) \odot \varphi(y)$ for all $x, y \in K_1$, and $\varphi(0) = 0$ and $\varphi(1) = 1$. A homomorphism $\varphi : K \rightarrow K$ is an *endomorphism* of K .

In the following, $(K, \oplus, \odot, 0, 1)$ will always denote a semiring and $\varphi : K \rightarrow K$ an endomorphism of this semiring.

We next define weighted automata. The underlying idea is to provide the transitions of a finite automaton with costs in the semiring K . For later purposes, we include ε -transitions. In order that costs for words are well defined, we have to assume that the ε -transitions do not form any loop. So let A be an alphabet and $\mathcal{A} = (Q, T, \text{in}, \text{out})$ where

- Q is a finite set of *states*,
- $T \subseteq Q \times (A \cup \{\varepsilon\}) \times K \times Q$ is a finite set of *transitions*,
- $\text{in}, \text{out} : Q \rightarrow K$ are *cost functions* for entering and leaving the system.

A *path* is a word $P = t_1 t_2 \dots t_n \in T^*$ with $t_i = (q_i, a_i, x_i, q_{i+1})$. Its *label* is the word $w = a_1 a_2 \dots a_n \in A^*$. Then we write $P : q_1 \xrightarrow{w}_{\mathcal{A}} q_{n+1}$ to denote that P is a w -labeled path from q_1 to q_{n+1} . We call $\mathcal{A}$ a *weighted automaton with ε -transitions* provided there is no nonempty ε -labeled path $P : q \rightarrow q$ for any state $q \in Q$, and $\mathcal{A}$ is called a *weighted automaton* if $T \subseteq Q \times A \times K \times Q$.

The *running cost* $\text{rcost}(P)$ of the path $P = t_1 t_2 \dots t_n \in T^*$ is defined inductively:

$$\begin{aligned} \text{rcost}(\varepsilon) &= 1 \\ \text{rcost}((q_1, a, x, q_2)P) &= \begin{cases} x \odot \varphi(\text{rcost}(P)) & \text{if } a \neq \varepsilon \\ x \odot \text{rcost}(P) & \text{otherwise.} \end{cases} \end{aligned}$$

If the path P is labeled by w , its *cost* is given by

$$\text{cost}(P) = \text{in}(q_1) \odot \text{rcost}(P) \odot \varphi^{|w|}(\text{out}(q_{n+1}))$$

Let $w \in A^*$ be some word. Since our automata do not have ε -loops, there are only finitely many paths labeled by w . The *behavior* $\|\mathcal{A}\|$ of the weighted automaton with ε -transitions $\mathcal{A}$ is the mapping $\|\mathcal{A}\| : A^* \rightarrow K$ defined by

$$(\|\mathcal{A}\|, w) = \sum \{\text{cost}(P) \mid P \text{ is a path with label } w\}$$

for $w \in A^*$. Note that the sum on the right is finite. If it is empty, then $(\|\mathcal{A}\|, w) = 0$.

In the sequel, we will use the term *formal power series* (series or FPS for short) for mappings $S : A^* \rightarrow K$.

Definition 2.1. A series $S : A^* \rightarrow K$ is called φ -recognizable if there exists a weighted automaton $\mathcal{A}$ with $\|\mathcal{A}\| = S$. By $\text{Rec}_\varphi(A^*)$, we denote the set of all series that are φ -recognizable.

If the endomorphism φ is understood from the context, we will simply speak of recognizable functions. First we claim that weighted automata have the same computational power as weighted automata with ε -transitions.

Lemma 2.2. Let $\mathcal{A}$ be a weighted automaton with ε -transitions. Then there exists a weighted automaton $\mathcal{A}'$ such that $\|\mathcal{A}\| = \|\mathcal{A}'\|$ and, for any transitions (q, a, x, r) and (q, a, y, r) of $\mathcal{A}'$, one has $x = y$.

Next we show that φ -recognizability of a series can also be described algebraically by representations, similarly to the classical case (with $\varphi = \text{id}$), cf. [1].

Let $n \in \mathbb{N}$ and $(K^{n \times n}, \odot)$ be the monoid of $(n \times n)$ -matrices over the semiring K (with the usual matrix multiplication). We extend φ to an endomorphism $\bar{\varphi}$ of $K^{n \times n}$ by setting $(\bar{\varphi}(B))_{ij} = \varphi(b_{ij})$ for each matrix $B \in K^{n \times n}$. We call a mapping $\mu : A^* \rightarrow K^{n \times n}$ a φ -morphism if $\mu(\varepsilon) = E$ (the unit matrix) and for all words $u, v \in A^*$, we have $\mu(uv) = \mu(u) \odot \bar{\varphi}^{|u|}(\mu(v))$. We call a triple $(\text{in}, \mu, \text{out})$ a *representation* of the weighted automaton $\mathcal{A}$ and of the series $\|\mathcal{A}\|$ if $\mu : A^* \rightarrow K^{n \times n}$ is a φ -morphism, $\text{in} \in K^{1 \times n}$ a row and $\text{out} \in K^{n \times 1}$ a column vector of size n such that $(\|\mathcal{A}\|, w) = \text{in} \odot \mu(w) \odot \varphi^{|w|}(\text{out})$ for $w \in A^*$ where $\varphi(\text{out})$ is the vector defined by applying φ to each coordinate of out .

Now let $\mathcal{A} = (Q, T, \text{in}, \text{out})$ be a weighted automaton with $Q = \{1, 2, \dots, n\}$. We define a φ -morphism $\mu : A^* \rightarrow K^{n \times n}$ by letting (for any $w \in A^*$ and $i, j \in Q$)

$$\mu(w)_{ij} = \sum_{P: i \xrightarrow{w} j} \text{rcost}(P).$$

Considering in as a $(1 \times n)$ -row vector and out as $(n \times 1)$ -column vector, one can show that $(\text{in}, \mu, \text{out})$ is a representation of $\mathcal{A}$.

Conversely, let $\mu : A^* \rightarrow K^{n \times n}$ be a φ -morphism, $\text{in} \in K^{1 \times n}$, and $\text{out} \in K^{n \times 1}$. Let Q be the set $\{1, 2, \dots, n\}$ and define $T \subseteq Q \times A \times K \times Q$ by putting $(i, a, x, j) \in T$ iff $(\mu(a))_{ij} = x$. Then $\mathcal{A} = (Q, T, \text{in}, \text{out})$ is a weighted automaton and $(\text{in}, \mu, \text{out})$ is a representation of $\mathcal{A}$. Thus we obtain

Proposition 2.3. *Let $S : A^* \rightarrow K$. Then S is φ -recognizable iff there is a representation of S .*

3 Finitary Formal Power Series

Recall that $(K, \oplus, \odot, 0, 1)$ is a semiring and φ is an endomorphism of this semiring. On the set K^{A^*} of series $S : A^* \rightarrow K$, we define the operation $\oplus$ pointwise: $(S \oplus T, w) = (S, w) \oplus (T, w)$. The φ -skew product of formal power series is defined by

$$(S \odot_{\varphi} T, w) = \sum_{\substack{u, v \in A^* \\ uv = w}} (S, u) \odot \varphi^{|u|}(T, v).$$

Note that this is the well studied Cauchy product in case φ is the identity on K . The structure $(K^{A^*}, \oplus, \odot_{\varphi}, \mathbf{0}, \mathbf{1})$ is denoted by $K_{\varphi} \langle \langle A^* \rangle \rangle$ (here, $(\mathbf{0}, w) = 0$ for $w \in A^*$, $(\mathbf{1}, w) = 0$ for $w \in A^+$, and $(\mathbf{1}, \varepsilon) = 1$). With this definition, tedious but straightforward calculations show

Lemma 3.1. *The structure $K_{\varphi} \langle \langle A^* \rangle \rangle$ is a semiring, the semiring of skew formal power series.*

Since our definition of $\odot_{\varphi}$ involves the “skew parameter” φ , the semiring $K_{\varphi} \langle \langle A^* \rangle \rangle$ deviates strongly from the semiring of classical formal power series over any semiring: for $u \in A^*$ and $x \in K$, let xu denote the *monomial power series* with $(xu, w) = 0$ for $w \neq u$ and $(xu, u) = x$. Then, for $a \in A$ and $y \in K$, the Cauchy product satisfies $1a \odot y\varepsilon = ya = y\varepsilon \odot 1a$, but for the skew product, we have $1a \odot_{\varphi} y\varepsilon = \varphi(y)a$ and $y\varepsilon \odot_{\varphi} 1a = ya$.

For a series S , let $S^n = S \odot_{\varphi} S^{n-1}$ with $S^0 = \mathbf{1}$. Then, for $w \in A^*$,

$$(S^n, w) = \sum \left\{ \prod_{i=1 \dots n} \varphi^{|u_1 u_2 \dots u_{i-1}|}(S, u_i) \mid u_i \in A^*, w = u_1 u_2 \dots u_n \right\}.$$

The series S is *quasiregular* provided $(S, \varepsilon) = 0$. In this case, we define

$$(S^+, w) = \sum_{1 \leq n \leq |w|} (S^n, w)$$

for $w \in A^+$ and $(S^+, \varepsilon) = 0$. Furthermore, $S^* = S^+ \oplus \mathbf{1}$ for S quasiregular.

Definition 3.2. *Let $\text{Rat}_{\varphi}(A^*)$ denote the least class of formal power series that contains the monomials xu for $x \in K$ and $u \in A \cup \{\varepsilon\}$ and is closed under the operations $\oplus$, $\odot_{\varphi}$, and $^+$ (applied to quasiregular formal power series). The series in $\text{Rat}_{\varphi}(A^*)$ are called φ -rational.*

For φ the identity, the set $\text{Rat}_{\varphi}(A^*)$ consists of those formal power series that are classically termed “rational”. In this case, Schützenberger showed that

$\text{Rat}_\varphi(A^*) = \text{Rec}_\varphi(A^*)$. We will indicate how to prove this fact for arbitrary endomorphisms.

Let E be a term over the signature $(\oplus, \odot_\varphi, +)$ with constants of the form xa for $x \in K$ and $a \in A \cup \{\varepsilon\}$. The evaluation $\|E\|$ is defined canonically in the semiring $K_\varphi \langle\langle A^* \rangle\rangle$. The term E is a *rational expression* if the operation $+$ is only applied to subexpressions whose value is a quasiregular formal power series. Let Exp denote the set of all rational expressions. It is obvious that they give rise precisely to the rational formal power series.

Let Q be a finite set of states, $T \subseteq Q \times \text{Exp} \times Q$ a finite set of transitions, $\iota \in Q$ an initial state, and $F \subseteq Q$ a set of accepting states. The label $\|P\| \in K_\varphi \langle\langle A^* \rangle\rangle$ of a path P is defined inductively by $\|\varepsilon\| = 1$ and $\|(i, E, j)P\| = \|E\| \odot_\varphi \|P\|$. The quadruple $\mathcal{A} = (Q, T, \iota, F)$ is called a *generalized weighted automaton* provided the label of any nonempty path $P : q \rightarrow q$ is quasiregular for any $q \in Q$. The behavior of the generalized weighted automaton is the formal power series given by

$$(\|\mathcal{A}\|, w) = \sum \{(\|P\|, w) \neq 0 \mid P : \iota \rightarrow F \text{ is a path}\}$$

(here we write $P : \iota \rightarrow F$ to denote that the path P leads from the initial state ι to some accepting state in F .) Note that, due to our assumption on the label of loops, this is well defined since, for any $w \in A^*$, there are only finitely many paths P in $\mathcal{A}$ with $(\|P\|, w) \neq 0$. Such automata have been investigated for the case $\varphi = \text{id}$ before, e.g., by Kuich and Salomaa [19].

The *depth* of a rational expression is defined in the obvious way:

$$\begin{aligned} \text{depth}(xa) &= 0, & \text{depth}(E^+) &= 1 + \text{depth}(E), \text{ and} \\ \text{depth}(E \odot_\varphi E') &= \text{depth}(E \oplus E') = 1 + \max(\text{depth}(E), \text{depth}(E')) \end{aligned}$$

Let $\mathcal{A}$ be a generalized weighted automaton. Since T is finite, there is a rational expression occurring in a transition of $\mathcal{A}$ that has maximal depth; its depth is the *depth* of $\mathcal{A}$. Finally, the *breadth* of a generalized weighted automaton measures how often its depth is realised: $\text{breadth}(\mathcal{A}) = |\{(i, E, j) \in T \mid \text{depth}(E) = \text{depth}(\mathcal{A})\}|$. Since T is finite, this is always a finite number.

Lemma 3.3. *Let S be a φ -rational formal power series. Then S is φ -recognizable.*

Proof. Let $\mathcal{A}$ be a generalized weighted automaton and let (i, E, j) be an edge of maximal depth. If the depth of E is 0, then any edge in $\mathcal{A}$ is labeled by a constant (i.e., by a monomial over a letter or the empty word). We can easily define a weighted automaton with ε -transitions $\mathcal{A}'$ with the same behavior. By Lemma 2.2, we can dispense of the ε -transitions of this automaton, hence the formal power series $\|\mathcal{A}\|$ is φ -recognizable.

If the depth of E is positive, then E is of one of the forms $E_1 \oplus E_2$, $E_1 \odot_\varphi E_2$, or E_1^+ . In each of these cases, we can replace the edge (i, E, j) by some other edges whose labels are among E_1 , E_2 , and 1ε . Hence the breadth (if it was at least 2) or the depth (otherwise) has decreased which allows us to proceed by induction. $\square$

A weighted automaton $\mathcal{A} = (\{1, 2, \dots, n\}, T, \text{in}, \text{out})$ is called *normalized* provided

1. $\text{in}(i) = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{otherwise} \end{cases}$ and $\text{out}(i) = \begin{cases} 1 & \text{if } i = 2 \\ 0 & \text{otherwise} \end{cases}$.
2. Furthermore, in T , there are no transitions of the form $(i, a, x, 1)$ or $(2, a, x, i)$.

Lemma 3.4. *Let S be a φ -recognizable formal power series. Then there exists a normalized weighted automaton $\mathcal{A}$ with $(\|\mathcal{A}\|, w) = (S, w)$ for $w \in A^+$ and $(\|\mathcal{A}\|, \varepsilon) = 0$.*

Thus, in the proof of the following lemma, we can start from a normalized weighted automaton that we consider as a generalized weighted automaton. Inductively, the transitions of this generalized weighted automaton get collapsed until, finally, just one is left whose label is the desired rational expression.

Lemma 3.5. *Let S be a φ -recognizable formal power series. Then S is φ -rational.*

Altogether, we have obtained:

Theorem 3.6. *Let K be a semiring and φ an endomorphism of K . Let A be an alphabet and let $S : A^* \rightarrow K$ be a formal power series. Then the following are equivalent:*

1. S is φ -recognized by a weighted automaton with ε -transitions.
2. S is φ -recognized by a weighted automaton.
3. S is φ -recognized by a generalized weighted automaton.
4. S is φ -rational.
5. S has a representation.

4 Preservation Properties

In analogy to classical results on formal power series [25,1,19], here we show that also in our setting certain homomorphisms $h : A^* \rightarrow B^*$ and also homomorphisms between semirings define transformations of series which preserve rationality resp. recognizability of the series. Such a homomorphism h is called *length-preserving* if $|h(u)| = |u|$ for any $u \in A^*$, and h is *finite-to-one*, if $h^{-1}(w)$ is finite for any $w \in B^*$. An endomorphism φ of K is *idempotent* if $\varphi \circ \varphi = \varphi$.

Theorem 4.1. *Let $h : A^* \rightarrow B^*$ be a monoid homomorphism. Assume that either h is length-preserving or that h is finite-to-one and φ is idempotent. Then the mapping $\bar{h} : K_\varphi \langle\langle A^* \rangle\rangle \rightarrow K_\varphi \langle\langle B^* \rangle\rangle$ defined by*

$$(\bar{h}(S), w) = \sum_{\substack{v \in A^* \\ h(v)=w}} (S, v)$$

for $S \in K_\varphi \langle\langle A^* \rangle\rangle$ and $w \in B^*$ is a semiring homomorphism. Furthermore, for any φ -rational $S \in K_\varphi \langle\langle A^* \rangle\rangle$, the series $\bar{h}(S)$ is φ -rational.

We can also consider homomorphisms of the underlying semiring:

Theorem 4.2. *Let $\alpha : (K, \varphi) \rightarrow (K', \psi)$ be a homomorphism (i.e., α is a semiring homomorphism that commutes with the endomorphisms φ and ψ : $\alpha \circ \varphi = \psi \circ \alpha$). Then $\tilde{\alpha} : K_\varphi \langle\langle A^* \rangle\rangle \rightarrow K'_\psi \langle\langle A^* \rangle\rangle$ defined by $(\tilde{\alpha}(S), w) = \alpha(S, w)$ is a semiring homomorphism that preserves rationality of formal power series.*

As a consequence of Theorems 4.1, 4.2 and 3.6, $\bar{h}$ and $\tilde{\alpha}$ also preserve the recognizability of formal power series. For a formal power series S , let the *support* $\text{supp}(S)$ denote the set of words w with $(S, w) \neq 0$.

Corollary 4.3. *Let K be a semiring such that $x \odot y = 0$ or $x \oplus y = 0$ implies $x = 0$ or $y = 0$. Let φ be an endomorphism of K with $\varphi^{-1}(0) = \{0\}$. Let S be a φ -recognizable formal power series. Then $\text{supp}(S) \subseteq A^*$ is a regular word language.*

Proof. Our assumptions on the semiring K allow us to define a semiring homomorphism α from K onto the Boolean semiring with $\alpha(x) = 0$ iff $x = 0$. Then the previous theorem yields the result. $\square$

5 Weighted Automata over the Semiring $\mathbb{R}_{\max}$

In this section, we will consider the semiring $K = \mathbb{R}_{\max}$. It is our aim to compare the sets $\text{Rec}_\varphi(A^*)$ for different endomorphisms φ of $\mathbb{R}_{\max}$. For $q \in \mathbb{R}_{\geq 0}$, let $q \cdot (-\infty) = -\infty$. Then the mapping $x \mapsto q \cdot x$ is a semiring endomorphism of $\mathbb{R}_{\max}$. Conversely, all endomorphisms of $\mathbb{R}_{\max}$ are of this form. We will write $\text{Rec}_q(A^*)$ and $\odot_q$ whenever we refer to the endomorphism given by multiplication with q .

Lemma 5.1. *Let $S \in \text{Rec}_q(A^*)$, $x \in \mathbb{R}_{\max}$, $w \in A^*$, and $p, q > 0$. Then $xw \odot_p S \in \text{Rec}_q(A^*)$.*

Thus, the set $\text{Rec}_p(A^*) \cap \text{Rec}_q(A^*)$ is closed under skew multiplication by a monomial from the left. The following lemma prepares the proof that $\text{Rec}_p(A^*) \cap \text{Rec}_q(A^*)$ is not closed under skew multiplication with a monomial from the right. It also shows that the sets $\text{Rec}_p(A^*)$ and $\text{Rec}_q(A^*)$ are incomparable for distinct and positive p and q .

For a word language $L \subseteq A^*$, let $\mathbf{1}_L$ denote the characteristic function of L .

Lemma 5.2. *Let $p, q > 0$ be distinct. Let furthermore $\sigma \in A$.*

1. *If $q \neq 1$, then the series S with $(S, \sigma^n) = n$ and $\text{supp}(S) = \sigma^*$ is 1- but not q -recognizable.*
2. *If $p \neq 1$, then the series $T_p = \mathbf{1}_{\sigma^*} \odot_p \mathbf{1}_\varepsilon$ is p - but not q -recognizable.*

This lemma is shown using pumping arguments in weighted automata. For the second statement, we deal with the possible order relations between p , q , and 1 separately. Summarizing, we get the following

Theorem 5.3. *Let $p \neq q$ be positive real numbers. Then $\text{Rec}_p(A^*)$ and $\text{Rec}_q(A^*)$ are incomparable. Furthermore, the intersection $\text{Rec}_p(A^*) \cap \text{Rec}_q(A^*)$*

- *contains all monomials and characteristic series $\mathbf{1}_L$ for regular word languages L ,*
- *is closed under finite summation and contains $xw \odot_r S$ for xw a monomial, $S \in \text{Rec}_p(A^*) \cap \text{Rec}_q(A^*)$, and $r > 0$, and*
- *does not necessarily contain $S \odot_r xw$ for xw a monomial, $S \in \text{Rec}_p(A^*) \cap \text{Rec}_q(A^*)$ and $r \neq 1$ positive.*

Let $p \neq q$ be positive real numbers. Then $\text{Rec}_p(A^*) \cap \text{Rec}_q(A^*)$ contains all monomials, certain characteristic series, and satisfies the above closure properties. We conjecture that it is the least set of formal power series having these properties. If this is indeed the case, then $\text{Rec}_p(A^*) \cap \text{Rec}_q(A^*) = \bigcap_{r>0} \text{Rec}_r(A^*)$.

6 Weighted Büchi-Automata over $\mathbb{R}_{\max}$

In this section, we will consider non-terminating executions of a weighted automaton. For these considerations, we restrict the parameter q to values satisfying $0 \leq q < 1$.

However, first we recall the classical definition of a *Büchi-automaton*: it is a quadruple $\mathcal{A} = (Q, T, I, F, F_\infty)$ with Q a finite set, $T \subseteq Q \times A \times Q$ and $I, F, F_\infty \subseteq Q$. A finite word $w \in A^*$ is accepted by $\mathcal{A}$ if it is accepted in the usual way by the automaton (Q, T, I, F) . An infinite word $w \in A^\omega$ is accepted by $\mathcal{A}$ if there exists a w -labeled path P in $\mathcal{A}$ which starts in some state from I and passes infinitely often through F_∞ . The set of all words in A^ω accepted by $\mathcal{A}$ is denoted by $L^\infty(\mathcal{A})$. A language $L \subseteq A^\omega$ is *Büchi-recognizable* if there exists a Büchi-automaton $\mathcal{A}$ with $L = L^\infty(\mathcal{A})$.

Now we generalize this concept to weighted Büchi-automata.

Definition 6.1. *A weighted Büchi-automaton is a tuple $\mathcal{A} = (Q, T, \text{in}, \text{out}, \text{out}_\infty)$ such that $(Q, T, \text{in}, \text{out})$ and $(Q, T, \text{in}, \text{out}_\infty)$ are weighted automata with weights in $\mathbb{R}_{\max}$.*

For a finite word $w \in A^*$, we define $(\|\mathcal{A}\|, w) = (\|(Q, T, \text{in}, \text{out})\|, w)$. For an infinite path $P = (p_i, a_i, x_i, p_{i+1})_{i \in \mathbb{N}}$ let $P \upharpoonright_n$ denote the prefix of P of length n . Then the cost of P is defined by

$$\text{cost}(P) = \limsup \{ \text{in}(p_1) + \text{rcost}(P \upharpoonright_n) + q^n \text{out}_\infty(p_{n+1}) \mid n \in \mathbb{N} \}$$

and the behavior of $\mathcal{A}$ at infinite words w is given by

$$(\|\mathcal{A}\|, w) = \sup \{ \text{cost}(P) \mid P \text{ is a path labeled by } w \}$$

Definition 6.2. *A mapping $S : A^\omega \rightarrow \mathbb{R}_{\max}$ is q -Büchi-recognizable if there exists a weighted Büchi-automaton $\mathcal{A}$ with $\|\mathcal{A}\| = S$. By $\omega\text{-Rec}_q(A^*)$, we denote the set of all functions that are q -Büchi-recognizable.*

Culik II and Karhumäki [5] used another definition of the behavior of a weighted automaton on infinite words: for $T : A^* \rightarrow \mathbb{R} \cup \{-\infty, \infty\}$ define another function $\vec{T} : A^\infty \rightarrow \mathbb{R} \cup \{-\infty, \infty\}$ by $(\vec{T}, w) = \limsup_{n \rightarrow \infty} (T, w|_n)$ for w infinite and $(\vec{T}, w) = -\infty$ for w finite.¹ For a weighted automaton, they define the behavior at infinity $|\mathcal{A}|$ by $|\mathcal{A}| = \|\vec{\mathcal{A}}\|$. Therefore, the behavior according to their definition is $-\infty$ at finite words. Let $A = \{a, b\}$ and

$$(S, w) = \begin{cases} -\infty & \text{if } w \in A^* \text{ or } w \text{ contains infinitely many } bs \\ 0 & \text{if } w \in A^* a^\omega \end{cases}$$

Then one can construct a weighted Büchi-automaton $\mathcal{A}$ with $\|\mathcal{A}\| = S$. On the other hand, there is no function $T : A^* \rightarrow \mathbb{R}_{\max}$ whose limit $\vec{T}$ is S (the proof is analogous to the proof that $A^* a^\omega$ is not the limit $\vec{L}$ of any subset L of A^* , cf. [23]).

Let $\mathcal{A}$ be a deterministic automaton and let $L \subseteq A^+$ be the language accepted by $\mathcal{A}$. If we consider $\mathcal{A}$ as a Büchi-automaton, it accepts the language $\vec{L}$. A similar fact can be shown for weighted automata, where a weighted automaton is deterministic if $(i, a, x, j), (i, a, y, k) \in T$ imply $x = y$ and $j = k$. The following slightly more general lemma imposes restrictions on the number of paths with a given label. Furthermore, we have to assume the automaton to be complete: for any state i and any letter a , there is an edge (i, a, x, j) for some weight $x \geq 0$ and some state j .²

Lemma 6.3. *Let $\mathcal{A} = (Q, T, \text{in}, \text{out})$ be a complete weighted automaton and let $\mathcal{A}' = (Q, T, \text{in}, -\infty, \text{out})$ with $-\infty(i) = -\infty$ for each $i \in Q$. If for any infinite word w there are only finitely many w -labeled paths P in $\mathcal{A}$ with $\text{cost}(P) \geq 0$, then $|\mathcal{A}| = \|\mathcal{A}'\|$.*

Recall that the class of ω -rational languages in A^∞ is the smallest class of languages that contains all singletons and is closed under the operations union, product, Kleene-iteration and ω -iteration (the latter two applied to languages in A^*). Now we define the corresponding notions in our context.

A mapping $S : A^\infty \rightarrow \mathbb{R}_{\max} = K$ is an *infinitary formal power series*; the set of all infinitary formal power series is denoted by $K_q \langle\langle A^\infty \rangle\rangle$. Any (finitary) formal power series S can be considered as an infinitary formal power series by setting $(S, w) = -\infty$ for $w \in A^\omega$. The operation $\max$ can naturally be extended to infinitary formal power series. The sum $+_q$ of a *finitary* FPS S and an infinitary FPS T is defined by

$$(S +_q T, w) = \sup_{\substack{uv=w \\ u \in A^*}} ((S, u) + q^{|u|}(T, v)).$$

¹ Actually, Culik II and Karhumäki work in the semiring $(\mathbb{R}, +, \cdot, 0, 1)$, but the idea of their definition is captured by this formula.

² These conditions are required by our proof, we are not sure whether they can be relaxed.

If S and T are both finitary, then this is precisely the operation $\odot_q$ we considered so far. The formal difference in the definition is the replacement of $\max$ by $\sup$. This has no effect for w finite since in that case we consider only the supremum of a finite set. If w is infinite, the set $\{(S, u) + q^{|u|}(T, v) \mid u \in A^*, uv = w\}$ can be infinite; hence we consider its supremum.

Note that for a sequence of elements $x_i \in K$, the sum $\sum_{i \in \mathbb{N}} x_i$ equals $-\infty$ whenever there is $i \in \mathbb{N}$ with $x_i = -\infty$. If this is not the case, then the sum can well be $+\infty$, i.e., it need not be defined within the semiring K . Formally, we define for a quasiregular finitary formal power series S its ω -iteration by

$$(S^\omega, w) = \sup \left\{ \sum_{i \in \mathbb{N}} q^{|u_1 u_2 \dots u_{i-1}|} (S, u_i) \mid u_i \in A^*, w = u_1 u_2 \dots \right\}$$

for $w \in A^\omega$ and $(S^\omega, w) = -\infty$ for $w \in A^*$. In general, $S +_q T$ and S^ω can take the value $+\infty \notin \mathbb{R}_{\max}$, i.e., in general $S +_q T, S^\omega \notin K_q \langle\langle A^\infty \rangle\rangle$. Suppose S and T are bounded, i.e., there is some $b \in \mathbb{R}$ with $(S, w) \leq b$ for any $w \in A^*$ and similarly for T . Then $(S, u) + q^{|u|}(T, v) \leq 2b$ and $\sum_{i \in \mathbb{N}} q^{|u_1 u_2 \dots u_{i-1}|} (S, u_i) \leq \frac{b}{1-q}$ for any $u_i \in A^*$. Thus, for bounded finitary FPS, we have $S +_q T, S^\omega \in K_q \langle\langle A^\infty \rangle\rangle$ and rational finitary FPS are bounded. Hence the following definition makes sense:

Definition 6.4. Let $\omega\text{-Rat}_q(A^*)$ denote the least class of infinitary FPS that contains the monomials xu for $x \in K$ and $u \in A \cup \{\varepsilon\}$ and is closed under the operations $\max$, $+_q$, $^+$ and $^\omega$ (the latter two applied to quasiregular finitary formal power series).

Lemma 6.5. Let S be an ω -rational infinitary formal power series. Then S is ω -recognizable.

Proof. Recall that any ω -rational language can be written as a finite union of ω -languages of the form $U \cdot V^\omega$ with $U, V \subseteq A^*$ regular. One can prove this fact using the characterization of ω -rational languages by Büchi-automata or finite syntactic monoids. An alternative proof is by induction on the ω -rational construction of the ω -language; this second proof generalizes to our situation yielding $S = \max(T, \max\{T_i +_q U_i^\omega \mid 1 \leq i \leq n\})$ for some $n \in \mathbb{N}$ and $T, T_i, U_i \in \text{Rat}_q(A^*)$ such that T_i and U_i are quasiregular ($1 \leq i \leq n$). Then one shows that any infinitary formal power series of the form $T_i +_q U_i^\omega$ is ω -recognizable (which uses Lemma 3.3). Combining the Büchi-automata for these series yields an automaton for S . $\square$

The converse implication is provided by the following

Lemma 6.6. Let S be an ω -recognizable infinitary formal power series. Then S is ω -rational.

Proof. One first shows that S is the behavior of a weighted Büchi-automaton $\mathcal{A} = (Q, T, \text{in}, \text{out}, \text{out}_\infty)$ with $\text{out}_\infty(i) \in \{0, -\infty\}$ for $i \in Q$. The finitary part of

$\|\mathcal{A}\|$ is rational. To show the same for the infinitary part, one considers automata $\mathcal{A}^{st}$ that differ from $\mathcal{A}$ only in the costs for entering and leaving the system:

$$\text{in}^{st}(k) = \begin{cases} \text{in}(k) & \text{if } s = k \\ -\infty & \text{otherwise} \end{cases} \quad \text{and} \quad \text{out}_{\infty}^{st}(k) = \begin{cases} \text{out}_{\infty}(k) & \text{if } t = k \\ -\infty & \text{otherwise} \end{cases}$$

Then $\|\mathcal{A}\| = \max_{s,t \in Q} \|\mathcal{A}^{st}\|$. Changing the costs for entering and leaving the system appropriately once more, one defines two weighted automata $\mathcal{A}_1$ and $\mathcal{A}_2$ from $\mathcal{A}^{st}$ satisfying $\|\mathcal{A}^{st}\| = \|\mathcal{A}_1\| +_q \|\mathcal{A}_2\|^\omega$. $\square$

Thus, we obtain the following characterization of ω -recognizable formal power series:

Theorem 6.7. *Let $0 \leq q < 1$ and $U : A^\infty \rightarrow \mathbb{R}_{\max}$. Then U is ω -recognizable iff it is ω -rational.*

To formally derive the classical Büchi-result for ω -languages, one first shows that for any $L \subseteq A^\infty$, we have that L is Büchi-recognizable (ω -rational, resp.) iff $\mathbf{1}_L \in \omega\text{-Rec}_q(A^*)$ ($\mathbf{1}_L \in \omega\text{-Rat}_q(A^*)$, resp.). Together with Theorem 6.7, this implies

Corollary 6.8. *Let $L \subseteq A^\infty$. Then L is Büchi-recognizable iff L is ω -rational.*

References

1. J. Berstel and C. Reutenauer. *Rational Series and Their Languages*. EATCS Monographs. Springer Verlag, 1988.
2. M. Bronstein and M. Petkovsek. An introduction to pseudo-linear algebra. *Theoret. Comp. Science*, 157:3–33, 1996.
3. J.R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.
4. A. Buchsbaum, R. Giancarlo, and J. Westbrook. On the determinization of weighted finite automata. *Siam J. Comput.*, 30:1502–1531, 2000.
5. K. Culik II and J. Karhumäki. Finite automata computing real functions. *SIAM J. of Computing*, pages 789–814, 1994.
6. K. Culik II and J. Kari. Image compression using weighted finite automata. *Computer & Graphics*, 17:305–313, 1993.
7. R.A. Cuninghame-Green. Minimax algebra and applications. *Advances in Imaging and Electron Physics*, 90:1–121, 1995.
8. M. Droste and P. Gastin. The Kleene-Schützenberger theorem for formal power series in partially commuting variables. *Information and Computation*, 153:47–80, 1999.
9. M. Droste and P. Gastin. On aperiodic and star-free formal power series in partially commuting variables. In *Formal Power Series and Algebraic Combinatorics (Moscow, 2000)*, pages 158–169. Springer, 2000.
10. M. Droste and D. Kuske. Skew and infinitary formal power series. Technical Report 2001-38, Department of Mathematics and Computer Science, University of Leicester, 2002. www.math.tu-dresden.de/~kuske/.

11. A. Galligo. Some algorithmic questions on ideals of differential operators. In *Proc. EUROCAL '85, vol. 2*, Lecture Notes in Comp. Science vol. 204, pages 413–421. Springer, 1985.
12. S. Gaubert. Rational series over dioids and discrete event systems. In *Proceedings of the 11th Int. Conf. on Analysis and Optimization of Systems: Discrete Event Systems, Sophia Antipolis, 1994*, Lecture Notes in Control and Information Sciences vol. 199. Springer, 1994.
13. S. Gaubert and M. Plus. Methods and applications of $(\max, +)$ linear algebra. Technical Report 3088, INRIA, Rocquencourt, January 1997.
14. U. Hafner. *Low Bit-Rate Image and Video Coding with Weighted Finite Automata*. PhD thesis, Universität Würzburg, Germany, 1999.
15. Z. Jiang, B. Litow, and O. de Vel. Similarity enrichment in image compression through weighted finite automata. In *COCOON 2000*, Lecture Notes in Comp. Science vol. 1858, pages 447–456. Springer, 2000.
16. F. Katritzke. *Refinements of data compression using weighted finite automata*. PhD thesis, Universität Siegen, Germany, 2001.
17. S.E. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–42. Princeton University Press, Princeton, N.J., 1956.
18. W. Kuich. Semirings and formal power series: Their relevance to formal languages and automata. In *Handbook of Formal Languages Vol. 1*, chapter 9, pages 609–677. Springer, 1997.
19. W. Kuich and S. Salomaa. *Semirings, Automata, Languages*. Springer, 1986.
20. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311, 1997.
21. M. Mohri, F. Pereira, and M. Riley. The design principles of a weighted finite-state transducer library. *Theoretical Comp. Science*, 231:17–32, 2000.
22. O. Ore. Theory of non-commutative polynomials. *Annals Math.*, 34:480–508, 1933.
23. D. Perrin and J.-E. Pin. Infinite words. Technical report, 1999. Book in preparation.
24. U. Püschmann. Zu Kostenfunktionen von Büchi-Automaten. Diploma thesis, TU Dresden, 2003.
25. A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. EATCS Texts and Monographs in Computer Science. Springer, 1978.
26. M.P. Schützenberger. On the definition of a family of automata. *Inf. Control*, 4:245–270, 1961.
27. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 133–191. Elsevier Science Publ. B.V., 1990.

Nondeterminism versus Determinism for Two-Way Finite Automata: Generalizations of Sipser's Separation^{*}

Juraj Hromkovič¹ and Georg Schnitger²

¹ Lehrstuhl für Informatik I, Aachen University RWTH, Ahornstraße 55, 52074 Aachen, Germany. Fax: ++49-241-8888216. jh@I1.Informatik.RWTH-Aachen.de

² Fachbereich Informatik, Johann-Wolfgang-Goethe Universität, Robert-Mayer-Straße 11-15, 60054 Frankfurt am Main, Germany

Abstract. Whether there exists an exponential gap between the size of a minimal deterministic two-way automaton and the size of a minimal nondeterministic two-way automaton for a specific regular language is a long standing open problem and surely one of the most challenging problems in automata theory. Twenty four years ago, Sipser [M.Sipser: Lower bounds on the size of sweeping automata. ACM STOC '79, 360-364] showed an exponential gap between nondeterminism and determinism for the so-called sweeping automata which are automata whose head can reverse direction only at the endmarkers. Sweeping automata can be viewed as a special case of oblivious two-way automata with a number of reversals bounded by a constant.

Our first result extends the result of Sipser to general oblivious two-way automata with an unbounded number of reversals. Using this extension we show our second result, namely an exponential gap between determinism and nondeterminism for two-way automata with the degree of non-obliviousness bounded by $o(n)$ for inputs of length n . The degree of non-obliviousness of a two-way automaton is the number of distinct orders in which the tape cells are visited.

Keywords: Finite automata, nondeterminism, descriptional complexity of regular languages

1 Introduction

Finite automata are the simplest uniform computing model and hence a base for the study of fundamental questions concerning computation and complexity. One of the central topics of theoretical computer science and especially of complexity theory is devoted to the comparison of nondeterministic computation and of deterministic computation. But not only the famous $P \stackrel{?}{=} NP$ problem seems to be hard, surprisingly, one is even not able to capture the computational power of nondeterminism for fundamental models of finite automata. To contribute to

^{*} Supported by DFG grants HR 1416-1 and SCHN 50312-1.

the study of the relative power of nondeterminism and determinism in finite automata is the main goal of this paper.

The “classical” one-way deterministic finite automaton (1dfa) was independently introduced in [6,8,11] and the one-way nondeterministic finite state automaton (1nfa) was proposed by Rabin and Scott [13], who proved that for any 1nfa there is an equivalent 1dfa by the well-know subset construction. Let, for every regular language, $s(L)$ be the size of the minimal 1dfa that accepts L , and let $ns(L)$ be the size of a minimal 1nfa that accepts L . The subset construction [13] assures $s(L) \leq 2^{ns(L)}$ for every regular language L . Already more than 30 years ago Meyer and Fisher [9] and Moore [12] found regular languages with an exponential gap between $s(L)$ and $ns(L)$.

The most natural generalization of a 1dfa [1nfa] is the two-way deterministic [nondeterministic] finite automaton – 2dfa [2nfa]. Two-way automata recognize only regular languages and their size may be considerably smaller than the size of one-way automata [12]. The following question is natural. Let, for every regular language L , $s_2(L)$ denote the size of a minimal 2dfa accepting L , and let $ns_2(L)$ denote the size of a minimal 2nfa that accepts L .

Does there exists a polynomial p , such that

$$s_2(L) \leq p(ns_2(L))$$

for every regular language L ?

Unfortunately, this question is still open and so it became one of the fundamental, most challenging open problems on the border between automata theory and complexity theory. The importance of this problem is underlined by its relation to the famous open question, whether deterministic logarithmic space (DLOG) is a proper subset of nondeterministic logarithmic space (NLOG). Berman [1] and Sipser [16] showed that if one proves an exponential gap between nondeterminism and determinism for two-way automata and the words involved in the proof are polynomial in length, then $\text{DLOG} \neq \text{NLOG}$.

The first (at least partially successful) attempt to attack this problem was done by Sakoda and Sipser [14], who proved an exponential gap between non-determinism and determinism for special automata which are allowed to read the input several times from the left to the right. Sipser [16] generalized this result to the so-called sweeping automata which are two-way finite automata whose head may reverse (change the direction of its movement) only at the end-markers. More precisely, Sipser found a sequence $\{B_n\}_{n=1}^{\infty}$ of regular languages with

$$ns(B_n) = O(n) \text{ and } s_2(B_n) \geq 2^n$$

for every $n \in \mathcal{N}$. Recently Leung [7] proved a maximal possible exponential gap between nondeterminism and determinism in the sweeping automata model for a sequence of regular languages over¹ $\{0, 1\}$.

¹ Note, that the size of the alphabets of the Sipser languages B_n grows with n .

The above mentioned results do not solve the problem for general two-way automata because Micali [10] showed that deterministic sweeping automata may require a number of states that is exponential in $s_2(L)$ for some specific regular languages L .

Our hypothesis is that there is an exponential gap between $ns(B_n)$ and $s_2(B_n)$, where B_n are the languages of Sipser [16]. We are not able to prove it here, but the main goal of this paper is to prove an exponential gap between nondeterminism and determinism for more powerful versions of two-way finite automata than sweeping automata.

First we observe, that the number of reversals of any deterministic sweeping automaton is bounded by its number of states and so it is a constant with respect to the input length. Hence, one possibility to extend Sipser's result is to solve the problem for two-way finite automata with a constant number of reversals.

Another possibility is to say that a sweeping automaton is a special restricted version of oblivious two-way automata. Obliviousness for a two-way automaton means that, for every input length n , the order of tape cells visited by the reading head of the automaton is the same for all inputs of length n . Our experience with proving lower bounds on complexity of specific problems says that the real hardness of proving lower bounds starts with non-obliviousness. There are many examples of computing models where one can prove good lower bounds for the oblivious versions of these models by transparent arguments, but for the non-oblivious versions the proofs are very technical or even no known technique for proving lower bounds works. The considered problem of proving exponential lower bounds on $s_2(L)$ is exactly of this kind. Since non-obliviousness seems to be the core of the hardness of proving lower bounds for several fundamental computing models, Ďuriš et. al. [2] propose to start to measure the degree of non-obliviousness and to investigate the tradeoff between complexity and the degree of non-obliviousness.

Here, we introduce the degree of non-obliviousness of a 2dfa A as a function $f_A : \mathcal{N} \rightarrow \mathcal{N}$, where $f_A(n)$ is the number of different orders of the indexes of the tape cells appearing in computations of A on inputs of length n . The main result of this paper says that

There is an exponential gap between 1nfa's and 2dfa's with the degree of non-obliviousness bounded by $o(n)$.

This paper is organized as follows. In Section 2 an exponential gap between non-determinism and determinism for oblivious two-way automata² is established. This is done by proving an exponential lower bound for B_n . Section 3 shows how to prove the main result of this paper. We explain there the proof idea by showing the exponential gap between nondeterminism and determinism for the degree 2 of non-obliviousness. The technical details for the proof for any sublinear non-obliviousness are moved to the appendix.

² Note that these automata may have the maximal possible (linear) number of reversals.

2 Oblivious Two-Way Automata

The goal of this section is to extend the result of Sipser to oblivious two-way automata. We do it by the reduction method, i.e., we prove that the existence of a small oblivious two-way deterministic automaton for B_n implies the existence of a small deterministic sweeping automaton for B_n .

In what follows we always assume, that the input tape of an automaton contains $\zeta w\$$ for any input word w , where ζ is called the left endmarker and $\$$ is called the right endmarker. If a two-way automaton reads ζ [$\$$] it may not move to the left [right]. Moreover, one assumes that each two-way automaton has exactly one accepting state q_{accept} and exactly one rejecting state q_{reject} . There is no more possible from these two special states.

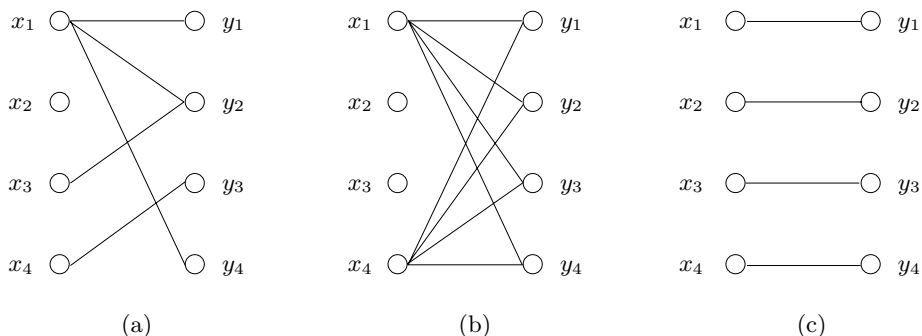


Fig. 1.

Now, let us describe the Sipser's language B_n . Let Σ_n be an alphabet of 2^{n^2} symbols where each symbol of Σ_n represents a bipartite graph of $2n$ vertices $x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_n$ with edges that lead from x -vertices to y -vertices only. Figure 1 shows examples of three symbols of Σ_4 . The symbol at Figure 1(c) corresponds to the bipartite graph

$$(\{x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4\}, \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)\}).$$

The bipartite graph of $2n$ vertices that contains exactly the edges (x_i, y_i) for $i = 1, \dots, n$ is called the dummy symbol of Σ_n and denoted by d_n . The concatenation of two symbols a and b of Σ_n represent a graph of $3n$ vertices that is obtained by identifying y_i of a with x_i of b for every $i \in \{1, \dots, n\}$. For instance, the concatenation of the bipartite graphs in Figure 1 (from the left to the right) results in the graph in Figure 2. Thus, any word w of Σ_n corresponds to a graph $G(w)$ of $n \cdot (|w| + 1)$ vertices. The language B_n consists of words $w \in (\Sigma_n)^+$ such that $G(w)$ contains a path of length $|w|$ that connects one of the n "left-most" vertices of $G(w)$ with one of the n "right-most" vertices of $G(w)$. For instance, the graph in Figure 2 corresponds to a word in B_n because there is a path from x_1 to y_1 of length 3.

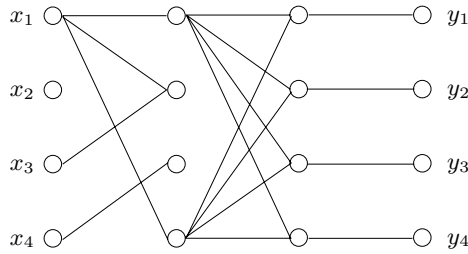


Fig. 2.

Our first useful observation is devoted to the dummy symbol d_n . Let h be a homomorphism defined by

$$h(a) = a \text{ for all } a \in \Sigma_n - \{d_n\}, \text{ and } h(d_n) = \epsilon.$$

where ϵ is the empty word.

Observation 1 For any $w \in \Sigma_n^*$,

$$h(w) \in L \text{ iff } w \in L.$$

Next we need to define some basic terms related to the computations of a 2dfa. A *configuration* of a 2dfa A is a triple $(q, \zeta w \$, i)$, where q is the state of A , $\zeta w \$$ is the content of the tape and i is the position of the reading head of A on the tape.

We always assume that ζ is on position 0 of the tape and hence $\$$ is on position $|w| + 1$. The pair (q, i) is called the *internal configuration* of configuration $(q, \zeta w \$, i)$. A *computation* of A is any sequence of configurations $C_1, \dots, C_m$ such that A can move from C_i to C_{i+1} in one step for $i = 1, \dots, m - 1$. Any subsequence $C_k, C_{k+1}, \dots, C_l$ of $C_1, \dots, C_m$, $1 \leq k < l \leq m$, is called a *computation part* of the computation $C_1, \dots, C_m$. Let q_0 be the initial state of A . The *computation of A on input w* is a computation of A that starts in the configuration $(q_0, \zeta w \$, 0)$ and finishes either in an accepting state or in a rejecting state. W.l.o.g. we may assume that any 2dfa has exactly one accepting state and exactly one rejecting state and that there are no more possible moves from these states.

For any configuration $C = (q, \zeta w \$, i)$, we set $state(C) = q$ and $pos(C) = i$. Any part $C_1, \dots, C_v$ of a computation $B_1, B_2, \dots, B_r, C_1, \dots, C_v, D_1, \dots, D_s$ is called a *cycle*, if $state(C_1) = state(C_v)$ and A reads the same symbol during this computation part (i.e., all tape cells visited in this computation part contain the same symbol).

Fact 1 If $C_1, \dots, C_v$ is a computation part with $state(C_1) = state(C_v)$ and $pos(C_1) = pos(C_v)$, then the computation containing $C_1, \dots, C_v$ is infinite and so it cannot be an accepting computation.

A cycle $D = C_1, \dots, C_v$ is called a *simple* cycle if

$$|\{\text{state}(C_1), \text{state}(C_2), \dots, \text{state}(C_v)\}| = v - 1,$$

i.e., the state $\text{state}(C_1) = \text{state}(C_v)$ is the only state that occurs twice in $C_1, \dots, C_v$.

We denote $\text{pos}(C_v) - \text{pos}(C_1)$ by $\text{move}(C_1, \dots, C_v) = \text{move}(D)$. If $\text{move}(D) > 0$, we say that D goes to the right, and if $\text{move}(D) < 0$, we say that D goes to the left. Let

$$\begin{aligned} \text{left}(D) &= \min\{\text{pos}(C_i) \mid i = 1, \dots, v\}, \text{ and} \\ \text{right}(D) &= \max\{\text{pos}(C_i) \mid i = 1, \dots, v\}. \end{aligned}$$

We denote by

$$\text{diff}(C_1, \dots, C_v) = \text{right}(D) - \text{left}(D)$$

the length of the part of the tape that is scanned during D . We observe that a simple cycle cannot cover many positions.

Observation 2 *Let $D = C_1, \dots, C_v$ be a simple cycle of a computation of a 2dfa $A = (Q, \Sigma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$. Then*

$$\text{diff}(D) \leq |Q|$$

and so $|\text{move}(D)| < |Q|$.

Observation 3 *Let $C = C_1, \dots, C_m$ be a part of a computation of a 2dfa A with k states. If $|\text{pos}(C_m) - \text{pos}(C_1)| \geq k$, $\text{pos}(C_i)$ lies between $\text{pos}(C_1)$ and $\text{pos}(C_m)$ for all $i = 1, \dots, m$, and all cells on the positions from $\text{pos}(C_1)$ to $\text{pos}(C_m)$ contain the same symbol, then C contains a simple cycle.*

Let $C = C_1, C_2, \dots, C_m$ be a part of a computation of a 2dfa. The sequence

$$\text{e-Traj}(C) = \text{pos}(C_1), \text{pos}(C_2), \dots, \text{pos}(C_m)$$

is called the *exact trajectory* of C . The *trajectory* of C is the maximal subsequence $\text{Traj}(C) = \alpha_1, \dots, \alpha_s$ of $\text{e-Traj}(C)$ such that

$$\alpha_1 = \text{pos}(C_1), \alpha_s = \text{pos}(C_m) \text{ and } \alpha_i \neq \alpha_{i+1} \text{ for } i = 1, \dots, s - 1.$$

Definition 1. *Let A be a 2dfa. We say that A is an oblivious 2dfa, if for every $n \in \mathbb{N}$, the trajectories of all computations of A on words of length n are the same.*

Observe that sweeping automata can be transformed to oblivious 2dfa's with the trajectories

$$(0, 1, 2, \dots, |w|, |w| + 1, |w|, \dots, 2, 1, 0)^j$$

for a $j \leq |Q|$ and $w \in \Sigma^*$. One can easily design a transformation to an oblivious 2dfa that causes at most a quadratic growth of the size of a given sweeping automaton.

Now, we are ready to present the main result of this section.

Theorem 1. *Let n be a positive integer. Every oblivious 2dfa that accepts B_n has at least 2^n states.*

Proof Outline. Let A_n be an oblivious 2dfa that accepts B_n , and let A_n have k states. We will show that there exists a sweeping 2dfa S_n that accepts B_n and has at most k states. To construct S_n we need first to show that the trajectory T_m of A_n on inputs of length m is “nice” in the sense of the following two facts. Let us call the first k symbols and the last k symbols of the input the *border of the input*.

Fact 2 *Let A_n have its head on $\dot{c}$ or $\$$ in a configuration of a computation on an input $w \in B_n$ of length $m > 2k$. Then in the next $(k+1)^2 + 1$ steps A_n either finishes its computation or leaves the border of the input.*

Proof. Fact 2 is a direct consequence of Fact 1.

To show that T_n must be nice we consider the input $w_m = (d_n)^m \in B_n$.

Fact 3 *Let $C = C_1, \dots, C_r$ be a computation part of A_n on w_m where $\text{pos}(C_1) = 0$ [$\text{pos}(C_r) = m+1$], $\text{pos}(C_r) = k+1$ [$\text{pos}(C_r) = m-k-1$] and $\text{pos}(C_i) \leq k+1$ [$\text{pos}(C_i) \geq m-k-1$] for all $i = 1, 2, \dots, r-1$. Then C contains a simple cycle C' that goes to the right [to the left].*

Proof. Fact 3 is a direct consequence of Observation 3.

Combining Fact 2 and Fact 3 one can obtain the following characterization of the trajectory T_m on w_m (and so on every word of length m from Σ_n).

Lemma 1. *Let T' be any part of T_m on w_m that starts on $\dot{c}$ [$\$$] and ends on $\$$ [$\dot{c}$]. Then after at most $(k+1)^2 + 1$ steps T' leaves the border and starts to move in a simple cycle going to the right [left] until T' reaches $\$$ [$\dot{c}$] (Figure 3).*

Thus, the computation $C(w)$ of A_n on $w_m = (d_n)^m$ alternates between short computation parts on the borders of the input and crossings of the input from the left to the right or from the right to the left in a simple cycle. The following lemma provides a property of $C(w)$ that is crucial for the construction of the sweeping 2dfa S_n with k states and $L(S_n) = B_n$.

Lemma 2. *Let $C_i, C_{i+1}, \dots, C_l$ be a part of $C(w_m)$ with the following properties:*

1. $\text{pos}(C_i) > 0$ and $\text{pos}(C_l) < m+1$ [$\text{pos}(C_i) < m+1$ and $\text{pos}(C_l) > 0$]
2. $\text{pos}(C_i) < \text{pos}(C_j) < \text{pos}(C_l)$ [$\text{pos}(C_i) > \text{pos}(C_j) > \text{pos}(C_l)$] for $j \in \{i+1, i+2, \dots, l-1\}$, and
3. $|\text{pos}(C_l) - \text{pos}(C_i)| \geq 2k$.

Then the position $\text{pos}(C_i)$ cannot be visited again in $C(w)$ before the end-marker $\$$ [$\dot{c}$] has been visited.

Now, we are ready to outline the construction of S_n . For every input

$$x = x_1 x_2 \dots x_r \in (\Sigma_n)^r,$$

S_n simulates the work of A_n on the input

$$\text{virtual}(x) = (d_n)^{2k} x_1 (d_n)^{2k} x_2 (d_n)^{2k} \dots (d_n)^{2k} x_r (d_n)^{2k}$$

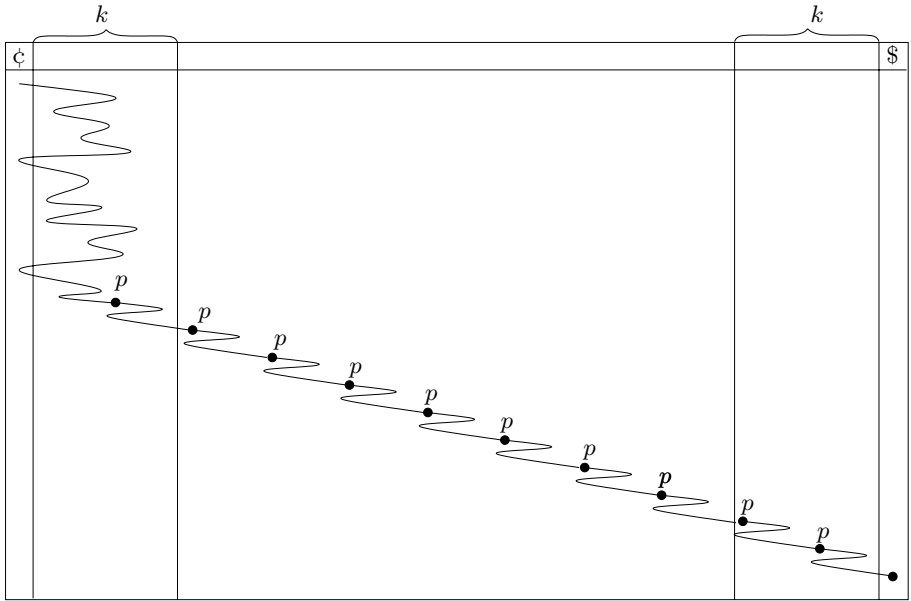
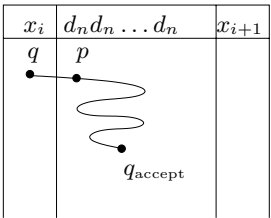


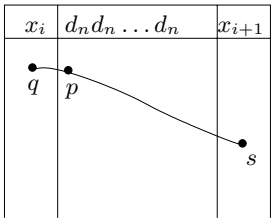
Fig. 3.

in the following way. If A_n reads a symbol x_i in a state q and after that it moves to the right [to the left] in a state p , then S_n looks in a table saying what happens when A_n enters the word $(d_n)^{2k}$ from the left [right] in the state p . There are only three possible situations (Figure 4):

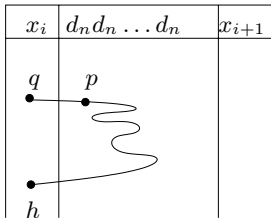
- (i) A_n finishes the computation in q_{accept} or q_{reject} without leaving the subword $(d_n)^{2k}$.
- (ii) A_n crosses $(d_n)^{2k}$ and leaves it on the other side in a state s .
- (iii) A_n leaves $(d_n)^{2k}$ and returns to x_i in a state h .



(i)



(ii)



(iii)

Fig. 4.

If (i) happens, S_n enters the corresponding state q_{accept} or q_{reject} without moving its head. If (ii) happens, then S_n moves the head to the right [left] to the position x_{i+1} [x_{i-1}] in the state s . If (iii) happens, S_n exchanges the state q for the state h without moving its input head.

One can easily observe that the table describing the behaviour of A_n on $(d_n)^{2k}$ can be stored in the transition function of S_n and that S_n uses the same set of states as A_n . The automaton S_n accepts B_n because of Observation 1 that claims

$$x \in B_n \Leftrightarrow \text{virtual}(x) \in B_n.$$

It remains to show that S_n is a sweeping 2dfa. But this is a direct consequence of Lemma 2 that claims that in a crossing of A_n on $\text{virtual}(x)$ from the left to the right [from the right to the left] one cannot return from x_{i+1} to x_i [from x_{i-1} to x_i] before visiting $\$$ [$\dagger$], i.e., if S_n simulates the work of A_n on $\text{virtual}(x)$ in the above described way then it makes reversals on the endmarkers only. Hence, S_n is a sweeping 2dfa.

This completes the proof of Theorem 1. $\square$

3 Bounded-Degree Non-oblivious Automata

In this section we present our main result and a proof idea.

Theorem 2. *Let n be a positive integer. Any 2dfa that accepts B_n with $o(n)$ degree of non-obliviousness has at least $2^{\Omega(n)}$ states.*

The idea of the proof is again the reduction to sweeping automata. We have to show that if there is a “small” 2dfa with sublinear degree of non-obliviousness for B_n , then there is a small sweeping 2dfa for B_n . Let D_n be a minimal 2dfa with a sublinear degree of non-obliviousness that accepts B_n . Let D_n have k_n states.

To simplify our argument we use a concept based on the following technical assertions.

Lemma 3. *For every $n \in \mathcal{N} - \{0\}$, there exists a positive integer r_n such that, any 2dfa accepting B_n and working in the sweeping manner on all inputs of length of at most r_n has at least $2^{\Omega(n)}$ states.*

Fact 4 *Let E be a 2dfa that accepts B_n and behaves in the sweeping manner on inputs of length r_n . Then there exists a 2dfa F with $L(E) = L(F) = B_n$, $\text{size}(F) = O(\text{size}(E))$, and F behaves in the sweeping manner on all inputs of lengths at most r_n .*

Following Lemma 3 and Fact 4 it is sufficient to show that the existence of a “small” 2dfa D_n with sublinear degree of non-obliviousness for B_n implies the existence of a small 2dfa that accepts B_n and works in the sweeping manner on all inputs of length r_n . First we outline the proof for the degree 2 of non-obliviousness and then we give an idea how to generalize it for proving Theorem 2.

Let $L(D_n) = B_n$, $\text{size}(D_n) = k_n$, and let the degree of non-obliviousness of D_n be at most 2.

Consider the work of D_n on inputs of the length

$$m = 3 \cdot (2k_n + 1) \cdot r_n.$$

Let T_1 and T_2 be the two possible trajectories of D_n on inputs of this length. Let T_1 be the trajectory on $(d_n)^m$. Following Lemma 1 and Lemma 2 the trajectory T_1 consists of crossings between $\dagger$ and $\$$ in which the head never moves back to a position in the distance $2k_n$ from the current position. Consider the set of inputs

$$X_1 = \{(d_n)^{2k_n} x_1 (d_n)^{2k_n} x_2 \dots x_{r_n-1} (d_n)^{2k_n} x_{r_n} y \mid \\ x_i \in \Sigma_n \text{ for } i = 1, \dots, n, y \in \{d_n\}^*, |y| = m - r_n \cdot (2k_n + 1)\}$$

We distinguish two possibilities with respect to the trajectories T_1 and T_2 .

1. Assume all words in X_1 have trajectory T_1 . Then in a similar way as in the proof of Theorem 1 one can construct a 2dfa H_n that for every input $x = x_1 x_2 \dots x_{r_n}$ of length r_n simulates the work of D_n on

$$\text{virtual}(x) = (d_n)^{2k_n} x_1 (d_n)^{2k_n} x_2 \dots x_{r_n} (d_n)^{2(k_n+1) \cdot r_n} \in X_1.$$

Thus, H_n computes in the sweeping manner on inputs of length r_n . Since H_n simulates the work of D_n on $\text{virtual}(y)$ for each $y \in \Sigma_n^*$ (i.e., for any input length), H_n accepts B_n . Since $\text{size}(H_n) = \text{size}(D_n)$, Lemma 4 and Fact 4 imply

$$\text{size}(D_n) = 2^{\Omega(n)}.$$

2. Assume a word $z \in X_1$ has trajectory T_2 . A reasonable generalization of Lemma 1 and Lemma 2 shows that the computations on all words with trajectory T_2 behave as described in Lemma 2 on the second half of these inputs. Since this is the case for T_1 too, D_n behaves “nicely” on the second half of all inputs. Considering the language

$$X_2 = \{(d_n)^{2(k_n+1)r_n} x_1 (d_n)^{2k_n} x_2 \dots x_{r_n-1} (d_n)^{2k_n} x_{r_n} (d_n)^{2k_n} \mid \\ x_i \in \Sigma_n \text{ for } i = 1, \dots, n\}$$

the proof can be completed in a similar way as in the first case.

Now we explain how to generalize this proof idea for proving Theorem 2.

Outline of the Proof of Theorem 2

Let D_n be a 2dfa that accepts B_n . Let the degree of non-obliviousness of D_n be bounded by a function $f(n) = o(n)$. Let $\text{Size}(B_n) = k(= k_n)$.

The idea of the proof is to show that there are some “nice” trajectories for some large group of words and then to use these trajectories to construct an automaton F_n with $L(F_n) = B_n$ that works in the sweeping manner on inputs of length r_n .

To formalize our idea we need the following terminology.

Definition 2. Let T be a trajectory of the computation $C(w)$ of D_n on an input word w . Let $w = xyz$ for some $x, y, z \in (\Sigma_n)^*$ and $|y| \geq 2k$. We say that T behaves nicely on the subword y of w if

1. T always crosses y from the left to the right or from the right to the left in a simple cycle (Lemma 1), and
2. in each crossing from the left to the right [from the right to the left] if T visits a position j of the tape, then in the rest of this crossing T does not visit any position $j - i$ [$j + i$] for $i > k$ (see Lemma 2).

The assertion of the following lemma can be proved in the same way as Lemma 1 and Lemma 2.

Lemma 4. Let $w = x(d_n)^k y (d_n)^k z \in (\Sigma_n)^+$ for a word $y \in \{d_n\}^*$, $|y| \geq 2k$. Then the trajectory of the computation of D_n on w behaves nicely on y .

Observation 4 Let T be a trajectory of D_n on two different words xyz and uvw with $|x| = |u|$, $|y| = |v|$ and $|z| = |w|$. If T behaves nicely on y , then T behaves nicely on v .

Thus, if T behaves nicely on y from xyz we can say that T behaves nicely on the (tape) interval $[|x| + 1, \dots, |x| + |y|]$.

Let c_n be a positive integer such that

$$f(m) < m / ((2k + 1) \cdot r_n) \quad (1)$$

for very $m \geq c_n$. Now, we study the computations of D_n on words of the length

$$m = (2k + 1) \cdot c_n \cdot r_n + 2k.$$

Since $m > c_n$, (1) implies that $f(m) < c_n$. Let $T_1, T_2, \dots, T_{f(m)}$ be all trajectories of D_n on inputs of the length m . Note, that some of these trajectories may be very complex and so they may be very far from being “nice” on any subpart of input. Our idea is to consider some classes $X_1, X_2, \dots, X_{C_n}$ of “nice” words and to show that the words of at least one of these classes have nice trajectories. Consider the sets:

$$Y = \left\{ (d_n)^{2k} x_1 (d_n)^{2k} x_2 \dots x_{r_n-1} (d_n)^{2k} x_{r_n} \mid x_i \in \Sigma_n \right\}$$

$$X_i = \left\{ (d_n)^{(2k+1) \cdot r_n \cdot (i-1)} y (d_n)^{(2k+1) \cdot r_n \cdot (c_n-i)+2k} \mid y \in Y \right\}$$

for all $i = 1, 2, \dots, c_n$. Observe (Figure 5) that for $i \neq j$, the intervals of non-dummy symbols of X_i and X_j do not overlap.

Since the number of intervals $[z_i, z_{i+1}]$ is larger than the number of trajectories, one can show that there exists a $j \in \{1, \dots, c_n - 1\}$ such that all words of X_j have trajectories that behave nicely on the interval $[z_{j-1} + k + 1, \dots, z_j]$ (Figure 5).

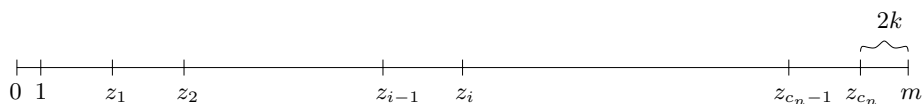


Fig. 5. Two words in X_i have their non-dummy symbols in the interval $[z_{i-1} + k, \dots, z_i - k]$, where $z_i = (2k + 1) \cdot r_n \cdot (i - 1)$ for $i = 1, \dots, c_n$.

Then the construction of F_n can be done as described as follows. For simplicity, consider $j = 2$. The automaton F_n working on some $y = y_1 \dots y_{r_n}$, $y_i \in \Sigma_n$ for $i = 1, \dots, r_n$ simulates the work of D_n on

$$\text{virtual}(y) = (d_n)^{(2k+1) \cdot r_n} (d_n)^{2k} y_1 \dots (d_n)^{2k} y_{r_n} (d_n)^{(2k+1) \cdot r_n (c_n - 2) + 2k} \in X_2.$$

To simulate the work on the block $(d_n)^{2k}$ between y_i and y_{i+1} one uses the strategy described in the construction of S_n in the proof of Theorem 1. If F_n reads $\dot{\varsigma}$ it simulates the work of D_n on $\dot{\varsigma}(d_n)^{(2k+1) \cdot r_n}$ and if F_n reads $\$$ then it simulates the work of D_n on $(d_n)^{(2k+1) \cdot r_n (c_n - 2) + 2k} \$$. These simulations can be performed by one step of F_n because the tables describing the input-output behaviour of D_n when entering $\dot{\varsigma}(d_n)^{(2k+1) \cdot r_n}$ from the left or $(d_n)^{(2k+1) \cdot r_n (c_n - 2) + 2k} \$$ from the right can be saved in the description of the transition function of F_n . Since all trajectories behave nicely on the interval $[z_1 + k + 1, \dots, m - k]$, F_n becomes a sweeping automaton on inputs of length r_n . □

References

1. P. Berman: A note on sweeping automata. In: *Proc. 7th ICALP. Lecture Notes in Computer Science* 85, Springer 1980, pp. 91–97.
2. P. Ďuriš, J. Hromkovič, S. Jukna, M. Sauerhoff, G. Schnitger: On multipartition communication complexity. In: *Proc. STACS '01. Lecture Notes in Computer Science* 2010, Springer 2001, pp. 206–217.
3. J. Hromkovič: *Communication Complexity and Parallel Computing*. Springer 1997.
4. J. Hromkovič, G. Schnitger: On the power of Las Vegas II: Two-way finite automata. *Theoretical Computer Science* 262 (2001), 1–24.
5. J. Hromkovič, J. Karhumäki, H. Klauck, G. Schnitger, S. Seibert: Measures of nondeterminism in finite automata. In: *Proc. 27th ICALP, Lecture Notes in Computer Science* 1853, Springer-Verlag 2000, pp. 194–21, full version: *Information and Computation* 172 (2002), 202–217.
6. D. A. Huffman: The synthesis of sequential switching circuits. *J. Franklin Inst.* 257, No. 3–4 (1954), pp. 161–190 and pp. 257–303.
7. H. Leung: Tight lower bounds on the size of sweeping automata. *J. Comp. System Sciences*, to appear.
8. G. M. Mealy: A method for synthesizing sequential circuits. *Bell System Technical Journal* 34, No. 5 (1955), pp. 1045–1079.
9. A. Meyer and M. Fischer: Economy in description by automata, grammars and formal systems. In: *Proc. 12th SWAT Symp.*, 1971, pp. 188–191.

10. S. Micali: Two-way deterministic automata are exponentially more succinct than sweeping automata. *Inform. Proc. Letters* 12 (1981), 103–105.
11. E. F. Moore: Gedanken experiments on sequential machines. In: [14], pp. 129–153.
12. F. Moore: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic and two-way finite automata. *IEEE Trans. Comput.* 10 (1971), 1211–1214.
13. M. O. Rabin, D. Scott: Finite automata and their decision problems. In: *IBM J. Research and Development*, 3 (1959), pp. 115–125.
14. W. J. Sakoda, M. Sipser: Nondeterminism and the size of two-way finite automata. In: *Proc. 10th ACM STOC*, 1978, pp. 275–286.
15. C. E. Shannon and J. McCarthy: *Automata Studies*. Princeton University Press, 1956.
16. M. Sipser: Lower bounds on the size of sweeping automata. *J. Comp. System Sciences* 21 (1980), 195–202.

Residual Languages and Probabilistic Automata

François Denis and Yann Esposito

LIF-CMI, UMR 6166
39, rue F. Joliot Curie
13453 Marseille Cedex 13 FRANCE
`fdenis,esposito@cmi.univ-mrs.fr`

Abstract. A stochastic generalisation of residual languages and operations on Probabilistic Finite Automata (PFA) are studied. When these operations are iteratively applied to a subclass of PFA called PRFA, they lead to a unique canonical form (up to an isomorphism) which can be efficiently computed from any equivalent PRFA representation.

1 Introduction

Probabilistic Automata are formal objects, equivalent to Hidden Markov Models under many aspects [6], which can be used to model stochastic processes in many application domains such as Pattern Recognition [1,2], Information Extraction [3], Bioinformatics [4,5]. A probabilistic automaton (PFA) has a structural component, which is a non deterministic automaton (NFA), and several continuous parameters which specify the probability for a state to be initial, to be terminal, and the probability to reach a state from another one while reading or emitting a given letter. A probabilistic automaton generates a regular stochastic language.

Determining an appropriate PFA structural component from a finite number of observations is an important open problem. In order to tackle this problem, it is necessary to identify subclasses of PFA which can be identified from given data. Deterministic PFA (PDFA), i.e. PFA whose structure is a deterministic NFA (DFA), have this property and have been used in several inference works [7, 8,9,10]. Unfortunately, contrary to the case of non stochastic regular languages, the class of stochastic languages which can be represented by PDFA is a very restricted subclass of the class of regular stochastic languages and it is necessary to find out new richer subclasses of PFA.

Several works have pointed out the importance of residual languages for Grammatical Inference [11,12]. A residual language of a language L is any language of the form $\{w|uw \in L\}$, for some word u . Most classical inference algorithms try to identify the residual languages of the target language L from a finite sample of L . A stochastic generalisation of residual languages has been introduced in [13] and has lead to the definition of Probabilistic Residual Finite State Automata (PRFA). A PRFA is a PFA whose states define residual languages of the language which is generated.

Here, we methodically pursue this study by introducing a reduction operator and a saturation operator which act on PFA (Section 3). We show that if a

stochastic language P can be generated by a PRFA, then iteratively applying the reduction and saturation operators to any PRFA which generates P provide a single object (up to an isomorphism): the canonical PRFA of L (Section 4). These canonical PRFAs are based on particular residual languages of P which cannot be decomposed by using other residual languages of P : we call them prime residual languages. Finally, we show in Section 5 that all the operations that we define are polynomial (whereas similar operations for non stochastic languages are PSPACE-complete [14]).

2 Preliminaries

2.1 Automata and Languages

Let Σ be a finite *alphabet*, and Σ^* be the set of words on Σ . We denote by ε the empty word and by $|u|$ the length of a word u . A *language* is a subset of Σ^* . A *nondeterministic finite automaton* (NFA) is a tuple $A = \langle \Sigma, Q, Q_0, F, \delta \rangle$ where Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, δ is the *transition function* defined from $Q \times \Sigma$ to 2^Q . We also denote by δ the extended transition function defined from $2^Q \times \Sigma^*$ to 2^Q . An NFA is *deterministic* (DFA) if Q_0 contains only one element q_0 and if $\forall q \in Q, \forall x \in \Sigma, \text{Card}(\delta(q, x)) \leq 1$. A word $u \in \Sigma^*$ is recognized by an NFA $A = \langle \Sigma, Q, Q_0, F, \delta \rangle$ if $\delta(Q_0, u) \cap F \neq \emptyset$ and the language recognized by A is $L_A = \{u \in \Sigma^* \mid \delta(Q_0, u) \cap F \neq \emptyset\}$. Let $Q' \subseteq Q$. We denote by $L_{A, Q'}$ the language $\{v \in \Sigma^* \mid \delta(Q', v) \cap F \neq \emptyset\}$. When Q' contains exactly one state q , we simply denote $L_{A, Q'}$ by $L_{A, q}$. It can be proved that the class of recognizable languages is identical to the class of regular languages (Kleene Theorem) and that every recognizable language can be recognized by a DFA. There exists a unique minimal DFA that recognizes a given recognizable language (minimal with regard to the number of states and unique up to an isomorphism). Let L be a language and u be a word. The *residual language* of L wrt u is $u^{-1}L = \{v \mid uv \in L\}$. A *Residual Finite State Automaton* (RFSA) is an NFA $A = \langle \Sigma, Q, Q_0, F, \delta \rangle$ such that, for each $q \in Q$, $L_{A, q}$ is a residual language of L_A [14].

2.2 Probabilistic Automata and Stochastic Languages

A *probabilistic finite state automaton* (PFA) is a tuple $\langle \Sigma, Q, \varphi, \iota, \tau \rangle$ where Q is a finite set of states, $\varphi : Q \times \Sigma \times Q \rightarrow [0, 1]$ is the transition function, $\iota : Q \rightarrow [0, 1]$ is the probability for each state to be initial and $\tau : Q \rightarrow [0, 1]$ is the probability for each state to be terminal. A PFA need satisfy $\sum_{q \in Q} \iota(q) = 1$ and for each state q , $\tau(q) + \sum_{a \in \Sigma} \sum_{q' \in Q} \varphi(q, a, q') = 1$. Let φ also denote the extension of the transition function, defined on $Q \times \Sigma^* \times Q$ by $\varphi(q, wa, q') = \sum_{q'' \in Q} \varphi(q, w, q'')\varphi(q'', a, q')$ and $\varphi(q, \varepsilon, q') = 1$ if $q = q'$ and 0 otherwise. We extend φ again on $Q \times 2^{\Sigma^*} \times 2^Q$ by $\varphi(q, U, R) = \sum_{w \in U} \sum_{r \in R} \varphi(q, w, r)$. The set of initial states is defined by $Q_I = \{q \in Q \mid \iota(q) > 0\}$, the set of reachable states is defined by $Q_{reach} = \{q \in Q \mid \exists r \in Q_I, \varphi(r, \Sigma^*, q) \neq 0\}$ and the set of

terminal states is defined by $Q_T = \{q \in Q \mid \tau(q) > 0\}$. The *support* of a PFA $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ is the NFA $\langle \Sigma, Q, Q_I, Q_T, \delta \rangle$ such that $\delta(q, x) = \{q' \mid \varphi(q, x, q') \neq 0\}$. A PFA is *admissible* if for any $q \in Q_{reach}$, $\varphi(q, \Sigma^*, Q_T) \neq 0$. We shall only consider admissible PFA. A *probabilistic deterministic finite state automaton* (PDFA) is a PFA whose support is deterministic.

A *stochastic language* on Σ is a function P defined from Σ^* to $[0, 1]$ such that $\sum_{u \in \Sigma^*} P(u) = 1$. For any $W \subseteq \Sigma^*$, let $P(W) = \sum_{w \in W} P(w)$. Let $S(\Sigma)$ be the set of stochastic languages on Σ . Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be an admissible PFA. Let P_A be the function defined on Σ^* by $P_A(u) = \sum_{q, q' \in Q \times Q} \iota(q) \varphi(q, u, q') \tau(q')$. It can be proved that P_A is a stochastic language on Σ which is called the stochastic language generated by A .

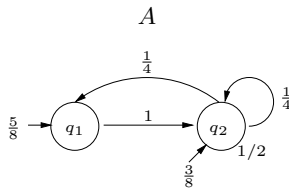


Fig. 1. An example of PFA A on $\Sigma = \{a\}$: $\iota(q_1) = 5/8, \iota(q_2) = 3/8, \phi(q_1, a, q_1) = 0, \phi(q_1, a, q_2) = 1, \phi(q_2, a, q_1) = 1/4, \phi(q_2, a, q_2) = 1/4, \tau(q_1) = 0$ and $\tau(q_2) = 1/2$. For sake of clarity, the letter a has not been drawn, nor null parameters such as $\phi(q_1, a, q_1)$ or $\tau(q_1)$. We have $P_A(\epsilon) = 3/16$ and $P_A(a) = 5/8 \cdot 1 \cdot 1/2 + 3/8 \cdot 1/4 \cdot 1/2 = 23/64$.

For every $q \in Q$, we denote by A_q the PFA $\langle \Sigma, Q, \varphi, \iota_q, \tau \rangle$ where $\iota_q(q) = 1$. $P_{A,q} = P_{A_q}$ is the stochastic language generated from q . Note that for any word u and any state q , $\varphi(q, u, Q) = P_{A,q}(u\Sigma^*)$. Let $\mathcal{L}_A = \{P_{A,q} \mid q \in Q\}$. Let $A = \langle \Sigma, Q, \varphi_A, \iota_A, \tau_A \rangle$ and $B = \langle \Sigma, Q, \varphi_B, \iota_B, \tau_B \rangle$ be two PFAs. A and B are *equivalent* if they define the same stochastic language, i.e. if $P_A = P_B$. A and B are *state-equivalent* if $P_A = P_B$ and if for every $q \in Q$, $P_{A,q} = P_{B,q}$. A and B are *isomorphic* ($A \sim B$) if they are state-equivalent and if they have the same support.

We extend the notion of residual languages to the stochastic case as follows. Let P be a *stochastic language*, the *residual language* $u^{-1}P$ of P with respect to u associates with every word w the probability $u^{-1}P(w) = P(uw)/P(u\Sigma^*)$ if $P(u\Sigma^*) \neq 0$. If $P(u\Sigma^*) = 0$, $u^{-1}P$ is not defined.

Let $\mathcal{L} \subseteq S(\Sigma)$ be a finite set of stochastic languages. We define the convex hull of $\mathcal{L}$ by $conv(\mathcal{L}) = \{L \in S(\Sigma) \mid \exists L_1, \dots, L_n \in \mathcal{L}, \exists \lambda_1, \dots, \lambda_n \geq 0 \mid L = \sum_{i=1}^n \lambda_i L_i\}$. For any $P \in conv(\mathcal{L})$, there exists a maximal subset of $\mathcal{L}$ that we denote by $cov(P, \mathcal{L})$ such that $P = \sum_{P_q \in cov(P, \mathcal{L})} \lambda_{P_q} P_q$ and $\lambda_{P_q} > 0$. We say that $\mathcal{L}$ is a *residual net* if for any $P \in \mathcal{L}$ and any letter $x \in \Sigma$, we have $x^{-1}P \in conv(\mathcal{L})$.

Example 1. Consider the PFA B on Fig. 2. We have $P_B = (P_{B,q_1} + P_{B,q_4})/2$, so $P_B \in conv(\{P_{B,q_1}, P_{B,q_4}\})$. As $P_{B,q_1} \neq P_{B,q_4}$, $cov(P_{B,q_1}, \{P_{B,q_1}, P_{B,q_4}\}) = \{P_{B,q_1}\}$. The set $\{P_{B,q_1}, P_{B,q_4}\}$ is not a residual net. Indeed, $a^{-1}P_{B,q_1} \notin$

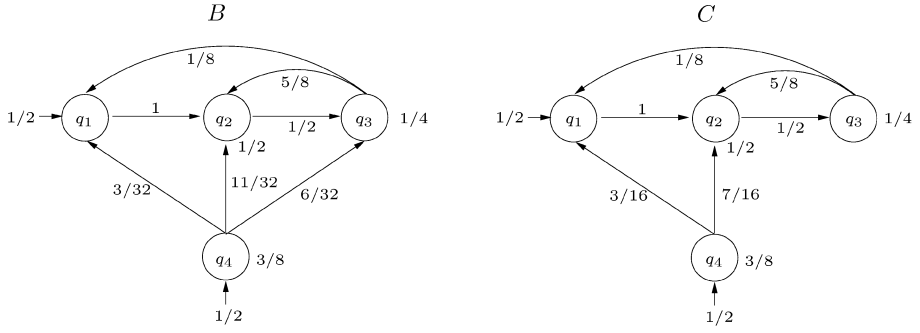


Fig. 2. B and C are two PFAs on $\Sigma = \{a\}$ which are state-equivalent but not isomorphic. They are equivalent to the PFA represented on Fig. 1.

$\text{conv}(\{P_{B,q_1}, P_{B,q_4}\})$ as $a^{-1}P_{B,q_1}(\epsilon) = 1/2$, $P_{B,q_1}(\epsilon) = 0$ and $P_{B,q_4}(\epsilon) = 3/8$. On the other hand, $\{P_{B,q_1}, P_{B,q_2}, P_{B,q_3}, P_{B,q_4}\}$ is a residual net.

A PRFA is a PFA $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ such that every state defines a residual language, i.e. such that $\forall q \in Q, \exists u \in \Sigma^*, L_{A,q} = u^{-1}P_A$ [13].

We denote by $\mathcal{L}_{PFA}(\Sigma)$ (resp. $\mathcal{L}_{PDFA}(\Sigma)$, resp. $\mathcal{L}_{PRFA}(\Sigma)$) the set of stochastic languages generated by some PFA (resp. PDFA, resp. PRFA). It can be shown that $\mathcal{L}_{PDFA}(\Sigma) \subsetneq \mathcal{L}_{PRFA}(\Sigma) \subsetneq \mathcal{L}_{PFA}(\Sigma)$ [13]. Each of these classes can be characterized in terms of residual languages [13].

Let P be a stochastic language:

- $P \in \mathcal{L}_{PDFA}(\Sigma)$ iff P has a finite number of residual languages.
- $P \in \mathcal{L}_{PRFA}(\Sigma)$ iff there exists a residual net $\mathcal{L}$ composed of residual languages of P such that $P \in \text{conv}(\mathcal{L})$.
- $P \in \mathcal{L}_{PFA}(\Sigma)$ iff there exists a residual net $\mathcal{L}$ such that $P \in \text{conv}(\mathcal{L})$.

3 Reduction and Saturation of Probabilistic Finite Automata

It is sometimes possible to suppress a state from a PFA while keeping the associated stochastic language. The reduction operator defined below takes as input a PFA A and a state q of A and outputs

- $\{A\}$ if $P_{A,q} \notin \text{conv}(\mathcal{L}_A \setminus \{P_{A,q}\})$,
- a set of PFAs equivalent to A which stem from the deletion of q otherwise.

Definition 1. Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be an admissible PFA, let $q \in Q$, let $Q' = Q \setminus \{q\}$ and let $\Lambda_q^A = \{(\lambda_r)_{r \in Q'} \mid \lambda_r \in \mathbb{R}^{\geq 0} \text{ and } P_{A,q} = \sum_{r \in Q'} \lambda_r P_{A,r}\}$.

- If $\Lambda_q^A = \emptyset$, i.e. $P_{A,q} \notin \text{conv}(\mathcal{L}_A \setminus \{P_{A,q}\})$, then $\text{red}(A, q) = \{A\}$,

- Otherwise, $\text{red}(A, q)$ is composed of the PFAs $A' = \langle \Sigma, Q', \varphi', \iota', \tau' \rangle$ such that there exists $(\lambda_r)_{r \in Q'} \in \Lambda_q^A$ such that
 - $\tau' = \tau|_{Q'}$,
 - $\iota'(r) = \iota(r) + \lambda_r \iota(q)$, for all $r \in Q'$,
 - $\varphi'(r, x, s) = \varphi(r, x, s) + \lambda_s \varphi(r, x, q)$ for all $r, s \in Q'$ and $x \in \Sigma$.

It can easily be checked that every element in $\text{red}(A, q)$ is an admissible PFA. Note that for any $A' \in \text{red}(A, q)$ and any states r and s of A' , $\varphi(r, x, s) \neq 0 \Rightarrow \varphi'(r, x, s) \neq 0$ and $\iota(r) \neq 0 \Rightarrow \iota'(r) \neq 0$. However, two different PFAs in $\text{red}(A, q)$ may have different support.

Example 2. Consider the PFA B defined on Fig. 2. We can show that $P_{B, q_3} = (P_{B, q_1} + P_{B, q_2})/2$ and that $P_{B, q_4} = (P_{B, q_1} + 5P_{B, q_2} + 2P_{B, q_3})/8 = (P_{B, q_1} + 3P_{B, q_2})/4$.

Proposition 1. *Let A be a PFA, let q be one of its states and let $A' \in \text{red}(A, q)$. Then, A' is equivalent to A and for any state r of A' , $P_{A, r} = P_{A', r}$.*

Proof. Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be a PFA, let $q \in Q$, let $A' = \langle \Sigma, Q', \varphi', \iota', \tau' \rangle \in \text{red}(A, q)$. Suppose that $A' \neq A$ and let $(\lambda_r) \in \Lambda_q^A$ be such that $P_{A, q} = \sum_{r \in Q'} \lambda_r P_{A, r}$. For any state r of Q' , we have $P_{A', r}(\varepsilon) = \tau(r) = P_{A, r}(\varepsilon)$. Now, assume that for any word w of length $\leq k$ and any state r of Q' we have $P_{A', r}(w) = P_{A, r}(w)$. Let x be a letter, we have:

$$\begin{aligned}
 P_{A', r}(xw) &= \sum_{s \in Q'} \varphi'(r, x, s) P_{A', s}(w) = \sum_{s \in Q'} (\varphi(r, x, s) + \lambda_s \varphi(r, x, q)) P_{A, s}(w) \\
 &= \sum_{s \in Q'} \varphi(r, x, s) P_{A, s}(w) + \varphi(r, x, q) \sum_{s \in Q'} \lambda_s P_{A, s}(w) \\
 &= \sum_{s \in Q'} \varphi(r, x, s) P_{A, s}(w) + \varphi(r, x, q) P_{A, q}(w) \\
 &= \sum_{s \in Q} \varphi(r, x, s) P_{A, s}(w) = P_{A, r}(xw).
 \end{aligned}$$

Then $P_{A', r} = P_{A, r}$ for any r of Q' . We remark that

$$\begin{aligned}
 P_{A'} &= \sum_{s \in Q'} \iota'(s) P_{A, s} = \sum_{s \in Q'} (\iota(s) + \lambda_s \iota(q)) P_{A, s} \\
 &= \sum_{s \in Q'} \iota(s) P_{A, s} + \iota(q) \sum_{s \in Q'} \lambda_s P_{A, s} = \sum_{s \in Q} \iota(s) P_{A, s} = P_A.
 \end{aligned}$$

We shall say that a PFA is reduced if none of its states can be reduced while preserving the associated language.

Definition 2. *A PFA A is reduced if for every state q , $\text{red}(A, q) = \{A\}$.*

Proposition 2. *Every PFA is equivalent to a reduced PFA.*

Proof. Any PFA is reduced or equivalent to a PFA which has less states. $\square$

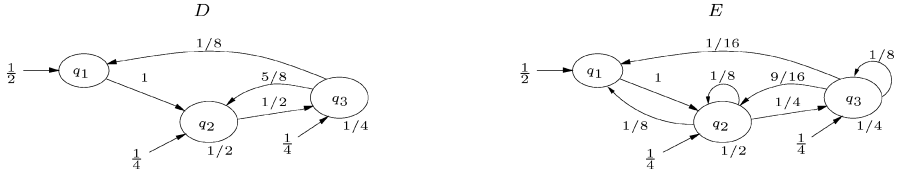


Fig. 3. $D \in \text{red}(C, q_4)$ using $P_{C,q_4} = \frac{1}{2}P_{C,q_2} + \frac{1}{2}P_{C,q_3}$ and $E \in \text{sat}(D)$.

Two elements of $\text{red}(A, q)$ may not be isomorphic, even if they are reduced (see Fig. 4). We shall obtain a unique element (up to an isomorphism) by adding as much transitions as possible while preserving the associated stochastic language. This will be achieved by using the saturation operator.

Definition 3. Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be a PFA. We define $\text{sat}(A)$ as the set of PFA $A' = \langle \Sigma, Q, \varphi', \iota', \tau \rangle$ such that for any states $q, r \in Q$, any letter $x \in \Sigma$, there exist non negative real numbers $\lambda_{q,r}^x$ such that

- $x^{-1}P_{A,q} = \sum_{r \in Q} \lambda_{q,r}^x P_{A,r}$ and $[P_{A,r} \in \text{cov}(x^{-1}P_{A,q}, \mathcal{L}_A) \Rightarrow \lambda_{q,r}^x > 0]$,
- $P_A = \sum_{r \in Q} \iota'(r)P_{A,r}$ and $[P_{A,r} \in \text{cov}(P_A, \mathcal{L}_A) \Rightarrow \iota'(r) > 0]$,
- $\varphi'(r, x, s) = \lambda_{r,s}^x \varphi(r, x, Q)$.

It can easily be checked that any element in $\text{sat}(A)$ is an admissible PFA.

Proposition 3. Let A be a PFA and let $A' \in \text{sat}(A)$. Then, A and A' are state-equivalent and for any state q of A , $P_{A,q} = P_{A',q}$.

Proof. Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be a PFA and let $A' = \langle \Sigma, Q, \varphi', \iota', \tau \rangle \in \text{sat}(A)$. We have for any state q , $P_{A',q}(\varepsilon) = \tau(q) = P_{A,q}(\varepsilon)$. Now assume that for any word w of length $\leq k$, and for any state q , $P_{A',q}(w) = P_{A,q}(w)$. Let x be a letter, we have:

$$\begin{aligned}
 P_{A',q}(xw) &= \sum_{r \in Q} \varphi'(q, x, r) P_{A',r}(w) \\
 &= \sum_{r \in Q} \lambda_{q,r}^x \varphi(q, x, Q) P_{A,r}(w) \text{ where the } \lambda_{q,r}^x \text{ satisfy the conditions of Def. 3,} \\
 &= \varphi(q, x, Q) \cdot \left[\sum_{r \in Q} \lambda_{q,r}^x P_{A,r} \right] (w) = P_{A,q}(x\Sigma^*) \cdot [x^{-1}P_{A,q}] (w) = P_{A,q}(xw).
 \end{aligned}$$

Then for any state q , $P_{A,q} = P_{A',q}$. We remark that $P_{A'} = \sum_{q \in Q} \iota'(q)P_{A',q} = \sum_{q \in Q} \iota'(q)P_{A,q} = P_A$, which concludes the proof. $\square$

We say that a PFA is saturated if it has a maximal number of transitions.

Definition 4. A PFA A is saturated if A is in $\text{sat}(A)$.

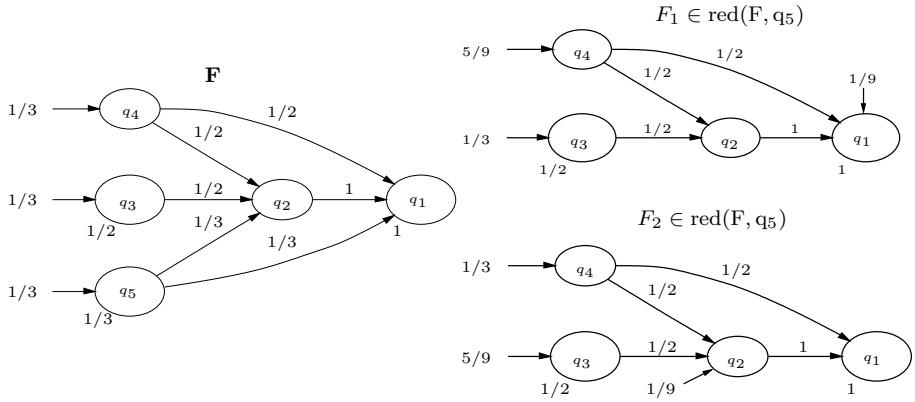


Fig. 4. Two non isomorphic reduced PFAs of F .

Next proposition states a number of properties of the *sat* operator. Proofs are omitted.

Proposition 4. *If A and B are state-equivalent, then $\text{sat}(A) = \text{sat}(B)$. A PFA A is saturated iff for any states r, s and any letter x we have $P_{A,s} \in \text{cov}(x^{-1}P_{A,r}, \mathcal{L}_A) \Rightarrow \varphi(r, x, s) \neq 0$ and $P_{A,r} \in \text{cov}(P_A, \mathcal{L}_A) \Rightarrow \iota(r) \neq 0$. Any element B of $\text{sat}(A)$ is saturated and moreover $\text{sat}(A) = \text{sat}(B)$. Any two elements of $\text{sat}(A)$ are isomorphic. If B is isomorphic to A and if A is saturated then B is saturated.*

Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be a PFA and let $\mathcal{A}$ be the set of PFAs $A' = \langle \Sigma, Q, \varphi', \iota', \tau \rangle$ such that A' is state equivalent to A . Define the relation $\prec$ on $\mathcal{A}$ by:

$$B \prec C \text{ iff } \iota_B(q) \neq 0 \Rightarrow \iota_C(q) \neq 0 \text{ and } \varphi_B(q, x, q') \neq 0 \Rightarrow \varphi_C(q, x, q') \neq 0$$

for any states q, q' and any letter x , where $B = \langle \Sigma, Q, \varphi_B, \iota_B, \tau \rangle$ and $C = \langle \Sigma, Q, \varphi_C, \iota_C, \tau \rangle$.

Proposition 5. *$(\mathcal{A}/\sim, \prec)$ is a semi-upper lattice whose maximal element is $\text{sat}(A)$.*

Proof. Let $B = \langle \Sigma, Q, \varphi_B, \iota_B, \tau \rangle, C = \langle \Sigma, Q, \varphi_C, \iota_C, \tau \rangle \in \mathcal{A}$. Define the PFA $B \vee C = \langle \Sigma, Q, \varphi', \iota', \tau \rangle$ where for any states r, s and any letter x , we have $\iota'(r) = (\iota_B(r) + \iota_C(r))/2$ and $\varphi'(r, x, s) = (\varphi_B(r, x, s) + \varphi_C(r, x, s))/2$. Check that $B \prec B \vee C, C \prec B \vee C$ and that for any D such that $B \prec D$ and $C \prec D$, we have $B \vee C \prec D$.

Now, it is clear from the definition of *cov* and from Prop. 4 that the elements of $\text{sat}(A)$ define a class which is the maximal element of $(\mathcal{A}/\sim, \prec)$. $\square$

Let $\mathcal{A}$ be a set of PFAs defined on the same alphabet and the same set of states Q and let $q \in Q$. Define $\text{sat}(\mathcal{A}) = \cup_{A \in \mathcal{A}} \text{sat}(A)$ and $\text{red}(\mathcal{A}, q) = \cup_{A \in \mathcal{A}} \text{red}(A, q)$.

Proposition 6. *Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be a PFA and let q be a state of A . Let $B \in \text{sat}(\text{red}(A, q))$ and $C \in \text{red}(\text{sat}(A), q)$. Then B and C are isomorphic.*

Proof. Let $A' \in \text{sat}(A)$ such that $C \in \text{red}(A', q)$. Let r, s be states of C and let x be a letter such that $P_{C,s} \in \text{cov}(x^{-1}P_{C,r}, \mathcal{L}_C)$. From Prop. 1, $P_{A',s} \in \text{cov}(x^{-1}P_{A',r}, \mathcal{L}_{A'})$. From Prop. 4, A' is saturated and then $\varphi_{A'}(r, x, s) \neq 0$. Therefore, $\varphi_C(r, x, s) \neq 0$. In a similar way, it can be shown that if $P_{C,s} \in \text{cov}(P_C, \mathcal{L}_C)$ then $\iota_C(s) \neq 0$. From Prop. 4, C is saturated. Now, $\text{sat}(B) = \text{sat}(C)$ as B and C are state-equivalent ; $C \in \text{sat}(B)$ as C is saturated. Therefore C is isomorphic to B from Prop. 5. $\square$

Proposition 7. *Let A be a saturated PFA and let q_1 and q_2 be two states from A . Let $B \in \text{red}(\text{red}(A, q_1), q_2)$ and $C \in \text{red}(\text{red}(A, q_2), q_1)$. Then, B and C are isomorphic.*

Proof. From Prop. 1, B and C are state-equivalent. Then, from Prop. 4, $\text{sat}(B) = \text{sat}(C)$. From Prop. 6, B and C are saturated. So, $B \in \text{sat}(C)$ and B and C are isomorphic. $\square$

Given a PFA, saturating and reducing it while it is possible provides an equivalent PFA which is reduced, saturated and unique up to an isomorphism. However, there exist non isomorphic reduced saturated equivalent PFAs (see Fig. 5).

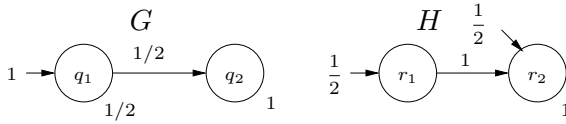


Fig. 5. Two non isomorphic reduced saturated equivalent PFAs.

4 Canonical PRFA

The application of reduction or saturation to a PRFA always yields a PRFA.

Proposition 8. *Let A be a PRFA and let q be a state of A . Then, all elements of $\text{red}(A, q) \cup \text{sat}(A)$ are PRFAs.*

Proof. As reduction and saturation do not change the languages generated from the states, every state will continue to generate a residual language.

Definition 5. *Let P be a stochastic language, a residual language $u^{-1}P$ is said to be composed if there exist residual languages $u_1^{-1}P, \dots, u_k^{-1}P$ such that $u^{-1}P \neq u_i^{-1}P$ for any $i = 1, \dots, k$ and such that $u^{-1}P \in \text{conv}(\{u_1^{-1}P, \dots, u_k^{-1}P\})$. A residual language is prime if and only if it is not composed.*

Clearly, a stochastic language generated by a PRFA with n states has at most n prime residual languages. The converse is false. Let $A = \langle \{a\}, \{q_1, q_2\}, \varphi, \iota, \tau \rangle$ be a PFA such that $\iota(q_1) = \iota(q_2) = 1/2$, $\tau(q_1) = 1 - \alpha$, $\tau(q_2) = 1 - \beta$, $\varphi(q_1, a, q_1) = \alpha$, $\varphi(q_2, a, q_2) = \beta$. The stochastic language P_A has only one prime residual language and cannot be generated by a PRFA. We have $P(a^n) = \frac{\alpha^n(1-\alpha) + \beta^n(1-\beta)}{2}$.

If $\alpha < \beta$, $\varepsilon^{-1}P_A$ is the unique prime residual language and for any integer $n > 0$, $(a^n)^{-1}P_A$ is composed of $\varepsilon^{-1}P_A$ and $(a^{n+1})^{-1}P_A$. However, it can easily be shown that a stochastic language whose set of prime residual languages is a finite residual net is in $\mathcal{L}_{PRFA}$. Furthermore, if $P \in \mathcal{L}_{PRFA}$ and if $\mathcal{P}$ is the set of its prime residual languages, every residual language of P is in $\text{conv}(\mathcal{P})$.

Proposition 9. *Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be a PRFA. Then, for any prime residual language $u^{-1}P_A$ and any $q \in \delta(Q_I, u)$, we have $P_{A,q} = u^{-1}P_A$, where δ is the transition function of the support of A . If A is reduced, then there exists only one state $q \in Q$ such that $P_{A,q} = u^{-1}P_A$. Moreover, any $P_{A,q}$ is a prime residual language of P_A .*

Proof. Let $R = \delta(Q_I, u)$, there exist non negative real numbers $(\alpha_r)_{r \in R}$ such that $u^{-1}P_A = \sum_{r \in R} \alpha_r P_{A,r}$. As $u^{-1}P_A$ is prime and as A is a PRFA, there must exist $r \in R$ such that $u^{-1}P_A = P_{A,r}$. Let $S = \{r \in R \mid P_{A,r} = u^{-1}P_A\}$, $\bar{S} = R \setminus S$ and let $\alpha = \sum_{s \in \bar{S}} \alpha_s$. If $\alpha < 1$, we would have $u^{-1}P_A = \sum_{s \in \bar{S}} \frac{\alpha_s}{1-\alpha} P_{A,s}$ which is impossible since each $P_{A,s}$ is a residual language of P_A and $u^{-1}P_A$ is prime. Therefore, $\alpha = 1$ and $S = \delta(Q_I, u)$. If A is reduced, there cannot be two distinct states of A which define the same stochastic language. Finally, as any residual language is composed of prime residual languages, any $P_{A,q}$ is a prime residual language of P_A if A is reduced. $\square$

As a corollary, it can be shown that the support of reduced PRFAs are exactly RFSAs $\langle \Sigma, Q, Q_0, F, \delta \rangle$ such that for every state $q \in Q$, there exists $u \in \Sigma^*$ such that $\delta(Q_0, u) = \{q\}$. So, not all RFSAs can be the support of a PRFA.

Proposition 10. *Let $P \in \mathcal{L}_{PRFA}$, let $\mathcal{P} = \{P_1, \dots, P_k\}$ be the set of all prime residual languages of P . Let $\alpha_{i,j}^x$ and β_i be non negative real numbers defined for all $1 \leq i, j \leq k$ and $x \in \Sigma$ such that*

- $x^{-1}P_i = \sum_{j=1}^k \alpha_{i,j}^x P_j$ with $P_j \in \text{cov}(x^{-1}P_i, \mathcal{P}) \Rightarrow \alpha_{i,j}^x > 0$
- $P = \sum_{i=1}^k \beta_i P_i$ with $P_i \in \text{cov}(P, \mathcal{P}) \Rightarrow \beta_i > 0$.

Let $A = \langle \Sigma, \mathcal{P}, \varphi, \iota, \tau \rangle$ be the PFA such that $\varphi(P_i, x, P_j) = \alpha_{i,j}^x P_i(x\Sigma^*)$, $\iota(P_i) = \beta_i$ and $\tau(P_i) = P_i(\varepsilon)$ for any $1 \leq i, j \leq k$ and any letter x . Then, A is a reduced saturated PRFA which generates P .

Proof. First, we prove by induction that for any state P_i of A , $P_{A,P_i} = P_i$. We have $P_{A,P_i}(\varepsilon) = P_i(\varepsilon)$. Assume now that for any state P_i and any word w of length $\leq l$, $P_{A,P_i}(w) = P_i(w)$. Let x be a letter, then we have:

$$\begin{aligned} P_{A,P_i}(xw) &= \sum_{j=1}^k \varphi(P_i, x, P_j) P_{A,P_j}(w) = \sum_{j=1}^k \alpha_{i,j}^x P_i(x\Sigma^*) P_j(w) \\ &= P_i(x\Sigma^*) \sum_{j=1}^k \alpha_{i,j}^x P_j(w) = P_i(x\Sigma^*) [x^{-1}P_i](w) = P_i(xw). \end{aligned}$$

Then for any state P_i , $P_{A,P_i} = P_i$. We have

$$P_A = \sum_{i=1}^k \beta_i P_{A,P_i} = \sum_{i=1}^k \beta_i P_i = P$$

so A generates P . Therefore, A is a PRFA. It is clear that A is reduced and saturated as every p is a prime residual language. $\square$

Let $\text{can}(P)$ be the set of *canonical* PRFAs obtained by the last construction. It is clear that any two elements of $\text{can}(P)$ are isomorphic.

Theorem 1. *Let $P \in \mathcal{L}_{PRFA}$. All reduced PRFAs that generate P are state-equivalent. All saturated reduced PRFAs that generate P are canonical PRFAs.*

Proof. From Prop. 9, all reduced PRFAs that generate P are state-equivalent. From Prop. 5 and 10, all saturated reduced PRFAs that generate P are in $\text{can}(P)$. $\square$

Previous results have a geometrical interpretation: the (possibly infinite) set of residual languages of a stochastic language generated by a PRFA is contained in a polytope whose vertices are its prime residual languages.

5 Decision and Complexity Problems

Deciding whether two NFAs are equivalent is a PSPACE-complete problem but deciding whether two PFAs are equivalent can be done within polynomial time [15]. Given a PFA $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$, there exist states $q_1, \dots, q_k$ s.t. any $P_{A,q}$ can be uniquely written as a linear combination of $P_{A,q_1}, \dots, P_{A,q_k}$, i.e. $P_{A,q} = \sum_{i=1}^k \alpha_q^i P_{A,q_i}$ where the α_q^i need not be non negative. Also, by adapting results from [16] and [15], it can easily be shown that there exists a polynomial algorithm which computes such states q_i and coefficients α_q^i from a given PFA. So, given a PFA $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$, $q \in Q$, $x \in \Sigma$ and $R \subseteq Q$, it can be decided within polynomial time whether P_A or $x^{-1}P_{A,q}$ belongs to $\text{conv}(\{P_{A,r} \mid r \in R\})$. Moreover, $S_I = \{r \in Q \mid P_{A,r} \in \text{cov}(P_A, \mathcal{L}_A)\}$ and $S_{q,x} = \{r \in Q \mid P_{A,r} \in \text{cov}(x^{-1}P_{A,q}, \mathcal{L}_A)\}$ can also be computed within polynomial time. Finally, using linear programming techniques, strictly positive coefficients such that

$$x^{-1}P_{A,q} = \sum_{r \in S_{q,x}} \alpha_{q,r}^x P_{A,r} \text{ and } P_A = \sum_{r \in S_I} \beta_r P_{A,r}$$

can be found within polynomial time. So,

Theorem 2. *Given a PFA A ,*

- *it is decidable in polynomial time whether A is reduced,*
- *a reduction of A can be computed within polynomial time,*
- *it is decidable in polynomial time whether A is saturated,*
- *a saturation of A can be computed within polynomial time.*

Note that these results contrast dramatically with the situation on NFAs. It has been shown in [14] that deciding whether an NFA is saturated or deciding whether it can be reduced are PSPACE-complete problems.

Proposition 11. *It is decidable whether a given reduced PFA is a PRFA.*

Proof. Let $A = \langle \Sigma, Q, \varphi, \iota, \tau \rangle$ be a reduced PFA. A is a PRFA iff its support is an RFSA $\langle \Sigma, Q, Q_0, F, \delta \rangle$ such that for every state $q \in Q$, there exists some $u \in \Sigma^*$ such that $\delta(Q_0, u) = \{q\}$. This last property can be decided, for example by using the subset construction to determinize A .

Proposition 12. *It is decidable whether a given PFA is equivalent to some PRFA having at most n states.*

Proof. Each state of a PRFA A having n states is uniquely reachable by a word whose length is $\leq 2^n$. So, $P_A \in \mathcal{L}_{PRFA}$ iff for any word $u \in \Sigma^{2^n}$ and any letter x , $(ux)^{-1}P_A \in \text{conv}(\{v^{-1}P_A | v \in \Sigma^{\leq 2^n}\})$ and this last property is decidable. $\square$

We do not know whether the following problems are decidable:

- given a PFA A , $P_A \in \mathcal{L}_{PDFA}$?
- given a PFA A , $P_A \in \mathcal{L}_{PRFA}$?
- given a PRFA A , $P_A \in \mathcal{L}_{PDFA}$?

The first problem is decidable when A is non ambiguous, i.e. if each word recognized by the support of A has only one derivation [17,18]. The second problem has an interesting geometrical formulation as for any $P \in \mathcal{L}_{PFA}$, the set $\{u^{-1}P | u \in \Sigma^*\}$ can be naturally embedded into a vector space of finite dimension: $P_A \in \mathcal{L}_{PRFA}$ iff the polyhedron $\text{conv}(\{u^{-1}P_A | u \in \Sigma^{\leq n}\})$ is stationary from some index n .

6 Conclusion

Residual languages are natural components of stochastic languages. This notion proves to be as useful as it is in classical language theory. In particular, it allows to define interesting subclasses of PFA and of regular stochastic languages. The fact that languages generated by PRFA have a unique canonical PRFA representation which can be computed within polynomial time from any equivalent PRFA is promising and should allow to design specific inference algorithms: this is a work in progress. Deciding whether a regular stochastic language can be generated by a PDFA is a classical difficult open problem. Deciding whether such a language can be generated by a PRFA seems to be at least as difficult as the previous problem.

References

1. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* **77** (1989) 257–285
2. Jelinek, F.: *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, Massachusetts (1997)
3. Freitag, D., McCallum, A.: Information extraction with HMM structures learned by stochastic optimization. In: *AAAI/IAAI*. (2000) 584–589
4. Baldi, P., Brunak, S.: *Bioinformatics: The Machine Learning Approach*. MIT Press (1998)
5. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. CUP (1998)
6. Casacuberta, F.: Some relations among stochastic finite state networks used in automatic speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-12** (1990) 691–695
7. Carrasco, R., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: *International Conference on Grammatical Inference*, Heidelberg, Springer-Verlag (1994) 139–152
8. Carrasco, R.C., Oncina, J.: Learning deterministic regular grammars from stochastic samples in p polynomial time. *RAIRO (Theoretical Informatics and Applications)* **33** (1999) 1–20
9. Thollard, F., Dupont, P., de la Higuera, C.: Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In: *Proc. 17th International Conf. on Machine Learning*, (KAUFM) 975–982
10. Higuera, C.D.L., Thollard, F.: Identification in the limit with probability one of stochastic deterministic finite automata. In: *Grammatical Inference: Algorithms and Applications*, 5th International Colloquium, ICGI 2000, Lisbon, Portugal, September 11 - 13, 2000 ; *Proceedings*. Volume 1891 of *Lecture Notes in Artificial Intelligence*., Springer (2000) 141–156
11. Denis, F., Lemay, A., Terlutte, A.: Learning regular languages using RFSA. In Springer-Verlag, ed.: *Proceedings of the 12th International Conference on Algorithmic Learning Theory (ALT-01)*. Number 2225 in *Lecture Notes in Computer Science* (2001) 348–359
12. Denis, F., Lemay, A., Terlutte, A.: Learning regular languages using non deterministic finite automata. In: *ICGI'2000*, 5th International Colloquium on Grammatical Inference. Volume 1891 of *LNAI*., Springer Verlag (2000) 39–50
13. Esposito, Y., Lemay, A., Denis, F., Dupont, P.: Learning probabilistic residual finite state automata. In: *ICGI'2002*, 6th International Colloquium on Grammatical Inference. *LNAI*, Springer Verlag (2002)
14. Denis, F., Lemay, A., Terlutte, A.: Residual Finite State Automata. *Fundamenta Informaticae* **51** (2002) 339–368
15. Balasubramanian, V.: Equivalence and reduction of hidden markov models. Technical Report AITR-1370, MIT (1993)
16. Paz, A.: *Introduction to probabilistic automata*. Academic Press, London (1971)
17. Mohri, M.: Finite-state transducers in language and speech processing. *Computational Linguistics* **23** (1997) 269–311
18. Allauzen, C., Mohri, M.: Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics* (to appear, 2002)

A Testing Scenario for Probabilistic Automata^{*}

Mariëlle Stoelinga¹ and Frits Vaandrager²

¹ Dept. of Computer Engineering, University of California, Santa Cruz
`marielle@soe.ucsc.edu`

² Nijmegen Institute for Computing and Information Sciences,
University of Nijmegen, The Netherlands
`fvaan@cs.kun.nl`

Abstract. Recently, a large number of equivalences for probabilistic automata has been proposed in the literature. Except for the probabilistic bisimulation of Larsen & Skou, none of these equivalences has been characterized in terms of an intuitive testing scenario. In our view, this is an undesirable situation: in the end, the behavior of an automaton is what an external observer perceives. In this paper, we propose a simple and intuitive testing scenario for probabilistic automata and we prove that the equivalence induced by this scenario coincides with the trace distribution equivalence proposed by Segala.

1 Introduction

A fundamental idea in concurrency theory is that two systems are deemed to be equivalent if they cannot be distinguished by observation. Depending on the power of the observer, different notions of behavioral equivalence arise. For systems modeled as labeled transition systems, this idea has been thoroughly explored and a large number of behavioral equivalences has been characterized operationally, algebraically, denotationally, logically, and via intuitive “testing scenarios” (also called “button pushing experiments”). We refer to Van Glabbeek [Gla01] for an excellent overview of results in this area of *comparative concurrency semantics*.

Testing scenarios provide an intuitive understanding of a behavioral equivalence via a machine model. A process is modeled as a black box that contains as its interface to the outside world (1) a display showing the name of the action that is currently carried out by the process, and (2) some buttons via which the observer may attempt to influence the execution of the process. A process autonomously chooses an execution path that is consistent with its position in the labeled transition system sitting in the black box. Trace semantics, for instance, is explained in [Gla01] with the *trace machine*, depicted in Figure 1 on the left. As one can see, this machine has no buttons at all. A slightly less trivial example is the *failure trace machine*, depicted in Figure 1 on the right. Apart

^{*} Research supported by PROGRESS Project TES4199, Verification of Hard and Softly Timed Systems (HaaST). A preliminary version of this paper appeared in the PhD thesis of the first author [Sto02a].

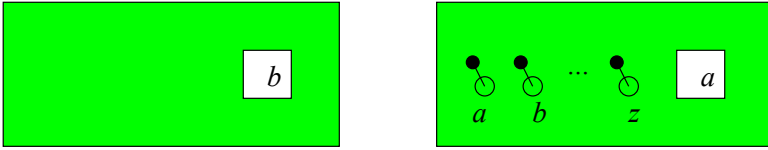


Fig. 1. The trace machine (left) and the failure trace machine (right).

from the display, this machine contains as its interface to the outside world a switch for each observable action. By means of these switches, an observer can determine which actions are *free* and which are *blocked* and may be changed at any time during a run of a process. The display becomes empty if (and only if) a process cannot proceed due to the circumstance that all actions are blocked. If, in such a situation, the observer changes her mind and allows one of the actions the process is ready to perform, an action will become visible again in the display. Figure 2 gives an example of two labeled transition systems that can be

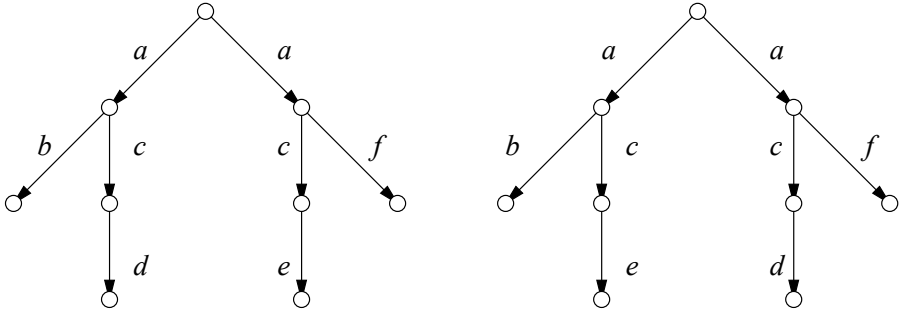


Fig. 2. Trace equivalent but not failure trace equivalent.

distinguished by the failure trace machine but not by the trace machine. Since both transition systems have the same traces (ε , a , ab , ac , af , acd and ace), no difference can be observed with the trace machine. However, via the failure trace machine an observer can see a difference by first blocking actions c and f , and only unblocking action c if the display becomes empty. In this scenario an observer of the left system may see an e , whereas in the right system the observer may see a d , but no e . We refer to [Gla01] for an overview of testing scenarios for labeled transition systems.

Probabilistic automata have become a popular mathematical framework for the specification and analysis of probabilistic systems. They have been developed by Segala [Seg95b,SL95,Seg95a] and serve the purpose of modeling and analyzing asynchronous, concurrent systems with discrete probabilistic choice in a formal and precise way. We refer to [Sto02b] for an introduction to probabilis-

tic automata, and a comparison with related models. In this paper, we propose and study a simple and intuitive testing scenario for probabilistic automata: we just add a *reset* button to the trace machine. The resulting *trace distribution machine* is depicted in Figure 3. By resetting the machine it returns to its initial

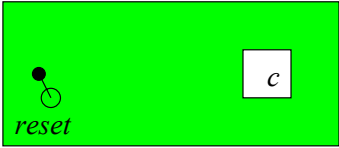


Fig. 3. The trace distribution machine.

state and starts again from scratch. In the non-probabilistic case the presence of a *reset* button does not make a difference¹, but in the probabilistic case it does: we can observe probabilistic behavior by repeating experiments and applying methods from statistics. Consider the two probabilistic automata in Figure 4. Here the arcs indicate probabilistic choice (as opposed to the nondeterministic



Fig. 4. Probabilistic automata representing a fair and an unfair coin.

choice in Figure 2), and probabilities are indicated adjacent to the edges. These automata represent a fair and an unfair coin, respectively. We assume that the trace distribution machine has an “oracle” at its disposal which resolves the probabilistic choices according to the probability distributions specified in the automaton. As a result, an observer can distinguish the two systems of Figure 4 by repeatedly running the machine until the display becomes empty and then restart it using the *reset* button. For the left process the number of occurrences of trace *ab* will approximately equal the number of occurrences of trace *ac*, whereas for the right process the ratio of the occurrence of the two traces will converge to 1 : 2. Elementary methods from statistics allow one to come up with precise definitions of distinguishing tests.

¹ For this reason, the *reset* button does not occur in the testing scenarios of [Gla01]. An obvious alternative to the *reset* button would be a *on/off* button.

The situation becomes more interesting when both probabilistic and nondeterministic choices are present. Consider the probabilistic automaton in Figure 5. If we repeatedly run the trace distribution machine with this automaton in-

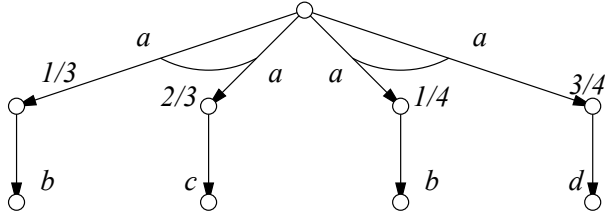


Fig. 5. The combination of probabilistic and nondeterministic choice.

side, the ratio between the various traces does not need to converge to a fixed value. However, if we run the machine sufficiently often we will observe that a weighted sum of the number of occurrences of traces ac and ad will approximately equal the number of occurrences of traces ab . Restricting attention to the cases where the left transition has been chosen, we observe $\frac{1}{2}\#[ac] \approx \#[ab]$. Restricting attention to the cases where the right transition has been chosen, we observe $\frac{1}{3}\#[ad] \approx \#[ab]$. Since in each execution either the left or the right transition will be selected, we have:

$$\frac{1}{2}\#[ac] + \frac{1}{3}\#[ad] \approx \#[ab].$$

Even though our testing scenario is simple, the combination of nondeterministic and probabilistic choice makes it far from easy to characterize the behavioral equivalence on probabilistic automata which it induces. The main technical contribution of this paper is a proof that the equivalence on probabilistic automata induced by our testing scenario coincides with the trace distribution equivalence proposed by Segala [Seg95a].

Being a first step, this paper limits itself to a simple class of probabilistic processes and to observers with limited capabilities. First of all, only *sequential* processes are investigated: processes capable of performing at most one action at a time. Furthermore, we only study *concrete* processes in which no internal actions occur. Finally, observers can only interact with machines in an extremely limited way: apart from observing termination and the occurrence of actions, the only way in which they can influence the course of events is via the *reset* button². It will be interesting to extend our result to richer classes of processes and more powerful observers, and to consider for instance a probabilistic version of the failure trace machine described earlier in this introduction.

² This ensures that our testing scenario truly is a “button pushing experiment” in the sense of Milner [Mil80]!

Related work. Several testing preorders and equivalences for probabilistic processes have been proposed in the literature [Chr90, Seg96, GN98, CDSY99, JY01]. All these papers study testing relations (i.e. testing equivalences or preorders) in the style of De Nicola and Hennessy [DNH84]. That is, they define a test as a (probabilistic) process that interacts with a system via shared actions and that reports success or failure in some way, for instance via success states or success actions. When a test is run on a system, the probability on success is computed, or if nondeterminism is present in either the test or the system, a set of these. By comparing the probabilities on success, one can say whether or not two systems are in the testing equivalence or preorder. For instance, two systems $\mathcal{A}$ and $\mathcal{B}$ are in the testing preorder of [JY01] if and only if for all tests T the maximal probability on success in $\mathcal{A} \parallel T$ is less than or equal to the maximal probability on success in $\mathcal{B} \parallel T$. The different testing relations in the mentioned papers arise by considering different kinds of probabilistic systems, by studying tests with different power (purely nondeterministic tests, finite trees or unrestricted probabilistic processes) and by using different ways to compare two systems under test (e.g. may testing versus must testing). All of the mentioned papers provide alternative characterizations of their testing relation in terms of trace-based relations.

Thus, these testing relations are button pushing experiments in the sense that a test interacts with a system via synchronization on shared actions. However, in our opinion these relations are not entirely observational, because it is not described how the *probability* on success can be observed. In our view, this is an undesirable situation: in the end, the behavior of an automaton is what an external observer perceives. Therefore, we believe that any behavioral equivalence should either be characterized via some plausible testing scenario, or be strictly finer than such an equivalence and be justified via computational arguments.

The only other paper containing a convincing testing scenario for probabilistic systems is by Larsen & Skou [LS91]. They define a notion of tests for *reactive* probabilistic processes, that is, processes in which all outgoing transitions of a state have different labels. Furthermore, the observer is allowed to make arbitrary many copies of any state. For those tests, a fully observable characterization of probabilistic bisimulation based on hypothesis testing is given. (We note that copies of tests can both serve to discover the branching structure of a system – as in the nondeterministic case – and to repeat a certain experiment a number of times.) Our work differs from the approach in [LS91] in the following aspects.

- We present our results in the more general probabilistic automaton model, whereas [LS91] considers the reactive model. As a consequence, the composition of a system and a test in [LS91] is purely probabilistic, that is, it does not contain nondeterministic choices, and theory from classical hypothesis testing applies. In contrast to this, the probabilistic automata that we consider do contain nondeterministic choices. To distinguish between likely and unlikely outcomes in these automata, we have to extend (some parts of) hypothesis testing with nondeterminism, which is technically quite involved.

- The main result of this paper, which is the characterization of trace distribution inclusion as a testing scenario, is established for all finitely branching systems, which is much more general than the minimal derivation assumption needed for the results in [LS91].
- The possibility in the testing scenario of Larsen & Skou to make copies of processes in any state (at any moment), is justified for instance in the case of a sequential system where one can make core dumps at any time. But for many distributed systems, it is not possible to make copies in any but the initial state. Therefore, it makes sense to study scenarios in which copying is not possible, as done in this paper.

Overview. Even though readers may not expect this after our informal introduction, the rest of this paper is actually quite technical. Section 2 recalls the definitions of probabilistic automata and their behavior and Section 3 presents the characterization of the testing preorder induced by the trace distribution machine as trace distribution inclusion. Sketches of some of the proofs are included in Appendix A. For complete proofs of all our results we refer to the full version of this paper [SV03].

2 Probabilistic Automata

We first recall a few basic notions from probability theory and introduce some notation.

Definition 1. A probability distribution over a set X is a function $\mu : X \rightarrow [0, 1]$ such that $\sum_{x \in X} \mu(x) = 1$. We denote the set of all probability distributions over X by $\text{Distr}(X)$. The probability distribution that assigns 1 to a certain element $x \in X$ and 0 to all other elements, is called the Dirac distribution over x and is denoted by $\{x \mapsto 1\}$.

Definition 2. A probability space is a triple $(\Omega, \mathcal{F}, \mathbf{P})$, where

- Ω is a set, called the sample space,
- $\mathcal{F} \subseteq 2^\Omega$ is σ -field, i.e. a collection of subsets of Ω which is closed under countable³ union and complement, and which contains Ω ,
- $\mathbf{P} : \mathcal{F} \rightarrow [0, 1]$ is a probability measure on $\mathcal{F}$, which means that $\mathbf{P}[\Omega] = 1$ and for any countable collection $\{C_i\}_i$ of pairwise disjoint subsets in $\mathcal{F}$ we have $\mathbf{P}[\cup_i C_i] = \sum_i \mathbf{P}[C_i]$.

Now, we recall the notion of a probabilistic automaton from Segala and Lynch [Seg95a, SL95]. Basically, a probabilistic automaton is a non-probabilistic automaton with the only difference that, rather than a single state, the target of a transition is a probability distribution over next states. We consider systems with only external actions, taken from a given, finite set Act . For technical reasons, we assume that Act contains a special element δ , referred to as the *halting* action.

³ In our terminology, countable objects include finite ones.

Definition 3. A probabilistic automaton (PA) is a triple $\mathcal{A} = (S, s^0, \Delta)$ with

- S a set of states,
- $s^0 \in S$ the initial state, and
- $\Delta \subseteq S \times \text{Act} \times \text{Distr}(S)$ a transition relation.

We write $s \xrightarrow{a} \mu$ for $(s, a, \mu) \in \Delta$ and $s \xrightarrow{a, \mu} t$ if $s \xrightarrow{a} \mu$ and $\mu(t) > 0$. We refer to the components of $\mathcal{A}$ as $S_{\mathcal{A}}, s_{\mathcal{A}}^0, \Delta_{\mathcal{A}}$. Moreover, $\mathcal{A}$ is finitely branching if for each state s , the set $\{(a, \mu, t) \mid s \xrightarrow{a, \mu} t\}$ is finite, i.e. if every state in $\mathcal{A}$ has finitely many outgoing transitions and the target distribution of each transition assigns a positive probability to finitely many elements.

For the remainder of this section, we fix a PA $\mathcal{A} = (S, s^0, \Delta)$ and assume that Δ contains no transition labeled with δ .

As in the non-probabilistic case, an execution of $\mathcal{A}$ is obtained by resolving the nondeterministic choices in $\mathcal{A}$. This choice resolution is described by an adversary, a function which in each state of the system determines the next transition to be taken. Adversaries are (1) randomized, i.e. make their choices probabilistically, (2) history-dependent, i.e. make choices depending on the path leading to the current state, and (3) partial, i.e. they may choose to halt the execution at any point in time. For technical simplicity, we prefer adversaries that only produce infinite sequences, even if the execution is halted. Therefore, we define the adversaries of a PA $\mathcal{A}$ via its halting extension.

Definition 4. A path of $\mathcal{A}$ is an alternating, finite or infinite sequence

$$\pi = s_0 a_1 \mu_1 s_1 a_2 \mu_2 s_2 \dots$$

of states, actions, and distributions over states such that (1) π starts with the initial state,⁴ i.e. $s_0 = s^0$, (2) if π is finite, it ends with a state, (3) $s_i \xrightarrow{a_{i+1}, \mu_{i+1}} s_{i+1}$, for each nonfinal i . We set the length of π , notation $|\pi|$, to the number of actions occurring in it and denote the set of all finite paths of $\mathcal{A}$ by $\text{Path}^*(\mathcal{A})$. If π is finite, then $\text{last}(\pi)$ denotes its last state. We define the associated trace of π , notation $\text{trace}(\pi)$, by $\text{trace}(\pi) = a_1 a_2 a_3 \dots$.

Definition 5. The halting extension of $\mathcal{A}$ is the PA $\delta\mathcal{A} = (S \cup \{\perp\}, s^0, \Delta')$, where Δ' is the least relation such that

1. $s \xrightarrow{\delta}_{\delta\mathcal{A}} \{\perp \mapsto 1\}$,
2. $s \xrightarrow{a}_{\mathcal{A}} \mu \implies s \xrightarrow{a}_{\delta\mathcal{A}} (\mu \cup \{\perp \mapsto 0\})$.

Here we assume that $\perp$ is fresh. The transitions with label δ are referred to as halting transitions.

⁴ Here we deviate from the standard definition, as we do not need paths starting from non-initial states.

Definition 6. A (partial, randomized, history-dependent) adversary E of $\mathcal{A}$ is a function

$$E : \text{Path}^*(\delta\mathcal{A}) \rightarrow \text{Distr}(\text{Act} \times \text{Distr}(S_{\delta\mathcal{A}}))$$

such that, for each finite path π , if $E(\pi)(a, \mu) > 0$ then $\text{last}(\pi) \xrightarrow{a}_{\delta\mathcal{A}} \mu$.

We say that E is deterministic if, for each π , $E(\pi)$ is a Dirac distribution. An adversary E halts on a path π if it extends π with the halting transition, i.e.,

$$E(\pi)(\delta, \{\perp \mapsto 1\}) = 1.$$

For $k \in \mathbb{N}$, we say that the adversary E halts after k steps if it halts on all paths with length greater than or equal to k . We denote by $\text{Adv}(\mathcal{A}, k)$ the set of all adversaries of $\mathcal{A}$ that halt after k steps and by $\text{Dadv}(\mathcal{A}, k)$ the set of deterministic adversaries in $\text{Adv}(\mathcal{A}, k)$. Finally, we call E finite if $E \in \text{Adv}(\mathcal{A}, k)$, for some $k \in \mathbb{N}$.

The probabilistic behavior of an adversary is summarized by its associated probability space. First we introduce the function $\mathbf{Q}^E$, which yields the probability that E assigns to finite paths.

Definition 7. Let E be an adversary of $\mathcal{A}$. The function $\mathbf{Q}^E : \text{Path}^*(\delta\mathcal{A}) \rightarrow [0, 1]$ is defined inductively by $\mathbf{Q}^E(s_0) = 1$ and $\mathbf{Q}^E(\pi a \mu s) = \mathbf{Q}^E(\pi) \cdot E(\pi)(a, \mu) \cdot \mu(s)$.

Definition 8. Let E be an adversary of $\mathcal{A}$. The probability space associated to E is the probability space given by

1. $\Omega_E = \text{Path}^\infty(\delta\mathcal{A})$,
2. $\mathcal{F}_E$ is the smallest σ -field that contains the set $\{C_\pi \mid \pi \in \text{Path}^*(\delta\mathcal{A})\}$, where $C_\pi = \{\pi' \in \Omega_E \mid \pi \text{ is a prefix of } \pi'\}$,
3. $\mathbf{P}_E$ is the unique measure on $\mathcal{F}_E$ such that $\mathbf{P}_E[C_\pi] = \mathbf{Q}^E(\pi)$, for all $\pi \in \text{Path}^*(\delta\mathcal{A})$.

The fact that $(\Omega_E, \mathcal{F}_E, \mathbf{P}_E)$ is a probability space follows from standard measure theory arguments, see for instance [Coh80].

As for non-probabilistic automata, the visible behavior of $\mathcal{A}$ is obtained by removing the non-visible elements (in our case, the states) from an execution (adversary). This yields a trace distribution of $\mathcal{A}$, which assigns a probability to (certain) sets of traces.

Definition 9. The trace distribution H of an adversary E , denoted $\text{trd}(E)$, is the probability space given by

1. $\Omega_H = \text{Act}^\infty$,
2. $\mathcal{F}_H$ is the smallest σ -field that contains the sets $\{C_\beta \mid \beta \in \text{Act}^*\}$, where $C_\beta = \{\beta' \in \Omega_H \mid \beta \text{ is a prefix of } \beta'\}$,
3. $\mathbf{P}_H$ is the unique measure on $\mathcal{F}_H$ such that $\mathbf{P}_H[X] = \mathbf{P}_E[\text{trace}^{-1}(X)]$.

Standard measure theory arguments [Coh80] ensure again that $\text{trd}(E)$ is well-defined. The set of trace distributions of adversaries of $\mathcal{A}$ is denoted by $\text{trd}(\mathcal{A})$ and $\text{trd}(\mathcal{A}, k)$ denotes the set of trace distributions that arise from adversaries of $\mathcal{A}$ halting after k steps. We write $\mathcal{A} \equiv_{\text{TD}} \mathcal{B}$ if $\text{trd}(\mathcal{A}) = \text{trd}(\mathcal{B})$; $\mathcal{A} \sqsubseteq_{\text{TD}} \mathcal{B}$ if $\text{trd}(\mathcal{A}) \subseteq \text{trd}(\mathcal{B})$ and $\mathcal{A} \sqsubseteq_{\text{TD}}^k \mathcal{B}$ if $\text{trd}(\mathcal{A}, k) \subseteq \text{trd}(\mathcal{B}, k)$.

3 Characterization of Testing Preorder

This section characterizes the observations of a trace distribution machine. That is, we define the set $\text{Obs}(\mathcal{A})$ of sequences of traces that are likely to be produced when the trace distribution machine operates as specified by the PA $\mathcal{A}$. Then, our main characterization theorem states that two PAs have the same observations if and only if they have the same trace distributions.

Define a *sample* O of depth k and width m to be an element of $(\text{Act}^k)^m$, i.e., a sequence consisting of m sequences of actions of length k . A sample describes what an observer may potentially record when running m times an experiment of length k on the trace distribution machine. Note that if, during a run, the machine halts before k observable actions have been performed, we can still obtain a sequence of k actions by attaching a number of δ actions at the end. We write $\text{freq}(O)$ for the function in $\text{Act}^k \rightarrow \mathbb{Q}$ that assigns to each sequence β in Act^k the frequency with which β occurs in O . That is, for $O = \beta_1, \beta_2, \dots, \beta_m$ let

$$\text{freq}(O)(\beta) = \frac{\#\{i \mid \beta_i = \beta, 1 \leq i \leq m\}}{m}.$$

Note that $\text{freq}(O)$ is a probability distribution over $(\text{Act}^k)^m$. We base our statistical analysis on $\text{freq}(O)$ rather than just O . This means we ignore some of the information contained in samples, which more advanced statistical methods may want to explore. If, for instance, we consider the sample O of depth one and width 2000 that consists of 1000 *head* actions followed by 1000 *tail* actions, then it is quite unlikely that this will be a sample of a trace distribution machine implementing a fair coin. However, the frequency function $\text{freq}(O)$ can very well be generated by a fair coin.

Assume that the process sitting in the black box is given by the PA $\mathcal{A}$. This means that, when operating, the trace distribution machine chooses a trace $\mathcal{A}$ according to some trace distribution H of $\mathcal{A}$. Thus, when running m experiments on the trace distribution, we obtain a sample O length m generated by a sequence of m trace distributions in $\text{trd}(\mathcal{A}, k)$.

For a trace distribution $H \in \text{trd}(\mathcal{A}, k)$, we denote by $\mu_H : \text{Act}^k \rightarrow [0, 1]$ the probability distribution given by $\mu_H(\beta) = \mathbf{P}_H[C_\beta]$. Since H halts after k steps, $\mu_H(\beta)$ yields the probability that the sequence β is picked when we generate a trace according to H . In other words, $\mu_H(\beta)$ yields the probability that during a run, the trace distribution machine produces the action sequence β , when it resolves its nondeterministic choices according to an adversary E with $\text{trd}(E) = H$. Now, we generate a sample of width m by independently

choosing m sequences according to distributions $H_1, \dots, H_m$ respectively. Then, the probability to pick the sample $O = \beta_1, \beta_2, \dots, \beta_m$ is given by

$$\mathbf{P}_{H_1, \dots, H_m}[O] = \prod_{i=1}^m \mu_{H_i}(\beta_i).$$

Finally, the probability that an element from the set $\mathcal{O} \subseteq (Act^k)^m$ is picked equals

$$\mathbf{P}_{H_1, \dots, H_m}[\mathcal{O}] = \sum_{O \in \mathcal{O}} \mathbf{P}_{H_1, \dots, H_m}[O].$$

Given $H_1, H_2, \dots, H_m$, we want to distinguish between samples that are likely to be generated by $H_1, H_2, \dots, H_m$, and those which are not. To do so, we first fix an $\alpha \in (0, 1)$ as the desired level of significance. Our goal is to define the set $\mathcal{K}_{H_1, H_2, \dots, H_m}$, of likely outcomes in such a way that

1. $\mathbf{P}_{H_1, \dots, H_m}[\mathcal{K}_{H_1, H_2, \dots, H_m}] > 1 - \alpha$,
2. $\mathcal{K}_{H_1, H_2, \dots, H_m}$ is, in some sense, minimal.

Condition (1) will ensure that, most likely, $H_1, \dots, H_m$ generate an element in $\mathcal{K}_{H_1, H_2, \dots, H_m}$. The probability that we reject O as a sample generated by $H_1, \dots, H_m$ while it is so, is at most α . Condition (2) will ensure that $\mathbf{P}_{H'_1, \dots, H'_m}[\mathcal{K}_{H_1, H_2, \dots, H_m}]$ is as small as possible for sequences $H'_1, \dots, H'_m$ different from $H_1, \dots, H_m$. (How small this probability is highly depends on which H'_i 's we take.) Therefore, the probability that we consider O to be an execution while it is not, is as small as possible. In terminology from hypothesis testing: our null hypothesis states that O is generated by $H_1, \dots, H_m$ and condition (1) bounds the probability on false rejection and (2) minimizes the probability on false acceptance. The set $\mathcal{K}_{H_1, H_2, \dots, H_m}$ is the complement of the critical section. Note that in classical hypothesis testing all subsequent experiments $\beta_1, \dots, \beta_m$ are drawn from the same probability distribution, whereas in our setting, each experiment is governed by a different probability mechanism given by H_i .

The idea behind the definition of $\mathcal{K}_{H_1, \dots, H_m}$ is as follows. The *expected frequency* of a sequence β in a sample generated by $H_1, \dots, H_m$ is given by

$$\mathbf{E}_{H_1, \dots, H_m}(\beta) = \frac{1}{m} \sum_{i=1}^m \mu_{H_i}(\beta).$$

Since fluctuations around the expected value are likely, we allow deviations of at most ε from the expected value. Here, we choose ε as small as possible, but large enough such that the probability on a sample whose frequency deviates at most ε from $\mathbf{E}_{H_1, \dots, H_m}$ is bigger than α . Then, conditions (1) and (2) above are met. Formally, define the ε -sphere $B_\varepsilon(\mu)$ with center μ as

$$B_\varepsilon(\mu) = \{\nu \in \text{Distr}(Act^k) \mid \text{dist}(\mu, \nu) \leq \varepsilon\},$$

where dist is the standard distance on $\text{Distr}(Act^k)$ given by $\text{dist}(\mu, \nu) = \sqrt{\sum_{\beta \in Act^k} |\mu(\beta) - \nu(\beta)|^2}$.

Definition 10. For a sequence $H_1, H_2, \dots, H_m$ of trace distributions in $\text{trd}(\mathcal{A}, k)$, we define $\mathcal{K}_{H_1, \dots, H_m}$ as the smallest⁵ sphere $B_\varepsilon(\mathbf{E}_{H_1, \dots, H_m})$ such that

$$\mathbf{P}_{H_1, \dots, H_m}[\{O \in (\text{Act}^k)^m \mid \text{freq}(O) \in B_\varepsilon(\mathbf{E}_{H_1, \dots, H_m})\}] > 1 - \alpha.$$

We say that O is an observation of $\mathcal{A}$ (of depth k and width m) if

$$O \in \mathcal{K}_{H_1, \dots, H_m}.$$

We write $\text{Obs}(\mathcal{A})$ for the set of observations of $\mathcal{A}$.

Example 1. We take $\alpha = 0.05$ as the level of significance. First, consider the leftmost PA in Figure 4 and samples of depth 2 and width 100. This means that the probabilistic trace machine is run 100 times and each time we get a trace of length 2.

Then any sample O_1 in which the sequence ab occurs 42 times and ac 58 times is an observation of $\mathcal{A}$, but samples in which ab occurs 38 times and ac 62 times are not. Let E be the adversary that, in each state of $\mathcal{A}$, schedules with probability one the unique transition leaving that state, if there is such a transition. Otherwise, E schedules the halting transition with probability one. For $H = \text{trd}(E)$, we have $\mu_H(ab) = \mu_H(ac) = \frac{1}{2}$ and $\mu_H(\beta) = 0$ for all other sequences. Let $H^{100} = (H_1, \dots, H_{100})$ be sequence of adversaries with $H_i = H$. Then $\mathbf{E}_{H^{100}} = \mu_H$ and, since μ_H assigns a positive probability only to ab and ac , we have that $\mathbf{P}_{H^{100}}[B_\varepsilon(\mu_H)] = \mathbf{P}_{H^{100}}[\{O_1 \mid \frac{1}{2} - \varepsilon \leq \text{freq}(O_1)(ab) \leq \frac{1}{2} + \varepsilon\}]$. One can show that then smallest sphere such that $\mathbf{P}_{H^{100}}[B_\varepsilon(\mu_H)] > 0.95$ is obtained by taking $\varepsilon = \frac{1}{10}$. Since $\text{freq}(O_1) \in B_\varepsilon(\mu_H)$, O_1 is an observation.

Then, a sample O_2 containing with 20 $\delta\delta$'s, 42 ab 's and 58 ac 's is an observation of depth 2 and width 120. It arises from taking 100 times adversary E as above and 20 adversaries that halt with probability one on every path. Now, consider the automaton in Figure 5. Consider the scheduler E_3 that in the initial state, schedules both a transitions with probability $\frac{1}{2}$. In the other states, E_3 schedules with probability one the unique outgoing transition if available and halts otherwise. Let $H_3 = \text{trd}(E_3)$ and let H_3^{120} be the sequence consisting of 120 times the adversary H_3 . The expected frequency of H_3^{120} is $\frac{7}{24}$ for ab , $\frac{8}{24}$ for ac , and $\frac{9}{24}$ for ad . Then $\mathcal{K}_{H_3^{120}} = B_{\frac{1}{11}}(\mathbf{E}_{H_3^{120}})$ and for instance, the sequence with 40 ab 's, 40 ac 's and 40 ad 's is an observation of the mentioned PA.

We can now state our main characterization theorem.

Theorem 1. For all finitely branching PAs $\mathcal{A}$ and $\mathcal{B}$

$$\text{Obs}(\mathcal{A}) = \text{Obs}(\mathcal{B}) \iff \mathcal{A} \equiv_{\text{TD}} \mathcal{B}.$$

Acknowledgement. The ideas worked out in this paper were presented in preliminary form at the seminar “Probabilistic Methods in Verification”, which took place from April 30 – May 5, 2000, in Schloss Dagstuhl, Germany. We thank the organizers, Moshe Vardi, Marta Kwiatkowska, Christoph Meinel and Ulrich Herzog, for inviting us to participate in this inspiring meeting.

⁵ This minimum exists, because there are finitely many samples.

References

- [BBK87] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51(1/2):129–176, 1987.
- [BK86] J.A. Bergstra and J.W. Klop. Verification of an alternating bit protocol by means of process algebra. In W. Bibel and K.P. Jantke, editors, *Math. Methods of Spec. and Synthesis of Software Systems '85*, *Math. Research* 31, pages 9–23, Berlin, 1986. Akademie-Verlag.
- [CDSY99] R. Cleaveland, Z. Dayar, S. A. Smolka, and S. Yuen. Testing preorders for probabilistic processes. *Information and Computation*, 154(2):93–148, 1999.
- [Chr90] I. Christoff. Testing equivalence and fully abstract models of probabilistic processes. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR 90*, Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
- [Coh80] D.L. Cohn. *Measure Theory*. Birkhäuser, Boston, 1980.
- [DNH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
- [Gla01] R.J. van Glabbeek. The linear time — branching time spectrum I. The semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland, 2001.
- [GN98] C. Gregorio-Rodríguez and M. Núñez. Denotational semantics for probabilistic refusal testing. In M. Huth and M.Z. Kwiatkowska, editors, *Proc. ProbMIV'98*, volume 22 of *Electronic Notes in Theoretical Computer Science*, 1998.
- [JY01] B. Jonsson and W. Yi. Compositional testing preorders for probabilistic processes. *Theoretical Computer Science*, 2001.
- [LS91] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.
- [Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.
- [Seg95a] R. Segala. Compositional trace-based semantics for probabilistic automata. In *Proc. CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 234–248, 1995.
- [Seg95b] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995. Available as Technical Report MIT/LCS/TR-676.
- [Seg96] R. Segala. Testing probabilistic automata. In *Proc. CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 299–314, 1996.
- [SL95] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
- [Sto02a] M.I.A. Stoelinga. *Allea jacta est: verification of probabilistic, real-time and parametric systems*. PhD thesis, University of Nijmegen, the Netherlands, April 2002. Available via <http://www.soe.ucsc.edu/~marielle>.

- [Sto02b] M.I.A. Stoelinga. An introduction to probabilistic automata. In G. Rozenberg, editor, *EATCS bulletin*, volume 78, pages 176–198, 2002.
- [SV03] M.I.A. Stoelinga and F.W. Vaandrager. A testing scenario for probabilistic automata. Technical Report NIII-R0307, Nijmegen Institute for Computing and Information Sciences, University of Nijmegen, 2003. Available via <http://www.soe.ucsc.edu/~marielle>.

A Appendix

This appendix proves the main characterization theorem of this paper, which says that the testing equivalence induced by the trace distribution machine coincides with the trace distribution equivalence. Our proof uses various auxiliary results which are stated, but the reader is referred to [SV03] for their proofs.

The first result we need states that each finite adversary in a finitely branching PA can be written as a convex combination of deterministic adversaries.

Lemma 1. *Let $k \in \mathbb{N}$, let $\mathcal{A}$ be a finitely branching PA and let E be an adversary in $\text{Adv}(\mathcal{A}, k)$. Then E can be written as a convex combination of deterministic adversaries in $\text{Dadv}(\mathcal{A}, k)$, i.e., there exists a probability distribution ν over $\text{Dadv}(\mathcal{A}, k)$ such that, for all π , a and μ ,*

$$E(\pi)(a, \mu) = \sum_{D \in \text{Dadv}(\mathcal{A}, k)} \nu(D) \cdot D(\pi)(a, \mu) \quad \text{and} \quad \mathbf{Q}^E(\sigma) = \sum_{D \in \text{Dadv}(\mathcal{A}, k)} \nu(D) \cdot \mathbf{Q}^D(\sigma).$$

A crucial result needed to characterize the testing equivalence is the *Approximation Induction Principle (AIP)* (cf. [BK86, BBK87]). This result is interesting in itself and was first observed in [Seg96]. A proof can be found in [SV03].

Theorem 2 (Approximation Induction Principle). *Let $\mathcal{A}$ and $\mathcal{B}$ be PAs and let $\mathcal{B}$ be finitely branching. Then*

$$\forall k. \mathcal{A} \sqsubseteq_{\text{TD}}^k \mathcal{B} \implies \mathcal{A} \sqsubseteq_{\text{TD}} \mathcal{B}.$$

By Chebychev's Inequality, one easily derives the following.

Proposition 1. *Let $\alpha, \varepsilon > 0$. Then there exists an $m' \in \mathbb{N}$ such that the following holds. For all $m \geq m'$, and all sequences $X_1, X_2, \dots, X_m$ of m independent random variables, where X_i has a Bernoulli distribution with parameter p_i , for some $p_i \in [0, 1]$ (i.e. $\mathbf{P}[X_i = 1] = p_i$, $\mathbf{P}[X_i = 0] = 1 - p_i$), we have that*

$$\mathbf{P}[|Z_m - \mathbf{E}[Z_m]| > \varepsilon] \leq \alpha.$$

Here, $Z_m = \frac{1}{m} \sum_{i=1}^m X_i$ yields the frequency of the number of times that a 1 has been drawn in $(X_1, \dots, X_m)$.

One can reformulate this proposition as follows.

Corollary 1. *Let $\alpha, \varepsilon > 0$ and $k \in \mathbb{N}$. Then there exists an $m' \in \mathbb{N}$ such that for all $m \geq m'$ and all trace distributions $H_1, H_2, \dots, H_m \in \text{trd}(\mathcal{A}, k)$*

$$\mathbf{P}_{H_1, \dots, H_m}[\{O \in (\text{Act}^k)^m \mid \text{freq}(O) \in B_\varepsilon(\mathbf{E}_{H_1, \dots, H_m})\}] > 1 - \alpha.$$

The following results are elementary. The second part follows from Lemma 1.

Proposition 2. 1. $H = K \iff \mu_H = \mu_K$.

2. For every $H \in \text{trd}(\mathcal{A}, k)$, μ_H can be written as a convex combination of distributions μ_{H_i} , where H_i is generated by a deterministic adversary. That is, there exists a probability distribution ν over the set $\text{Dadv}(\mathcal{A}, k)$ such that, for all $\sigma \in \text{Act}^k$, $\mu_K(\sigma) = \sum_{D \in \mathcal{D}} \nu(D) \cdot \mu_{\text{trd}(D)}(\sigma)$.

Now, we can prove our main theorem.

Theorem 3. For all finitely branching PAs $\mathcal{A}$ and $\mathcal{B}$

$$\text{Obs}(\mathcal{A}) = \text{Obs}(\mathcal{B}) \iff \mathcal{A} \equiv_{\text{TD}} \mathcal{B}.$$

Proof: The “ $\Leftarrow$ ” follows immediately from the definitions. To prove “ $\Rightarrow$ ” assume that $\mathcal{A} \not\equiv_{\text{TD}} \mathcal{B}$. We show that $\text{Obs}(\mathcal{A}) \not\subseteq \text{Obs}(\mathcal{B})$.

By Theorem 2, there exists a k such that $\mathcal{A} \not\equiv_{\text{TD}}^k \mathcal{B}$, i.e. $\text{trd}(\mathcal{A}, k) \not\subseteq \text{trd}(\mathcal{B}, k)$. Let H be a trace distribution in $\text{trd}(\mathcal{A}, k)$ that is not a trace distribution in $\text{trd}(\mathcal{B}, k)$. Then, Proposition 2(1) concludes that there is no $K \in \text{trd}(\mathcal{B}, k)$ such that $\mu_H = \mu_K$. Moreover, Proposition 2(2) states that the set $\{\mu_K \mid K \in \text{trd}(\mathcal{B}, k)\}$ is a polyhedron. Therefore, there is minimal distance $d > 0$ between μ_H and any μ_K with K in $\text{trd}(\mathcal{B}, k)$.

We write H^m for the sequence $(H_1, H_2, \dots, H_m)$ with $H_i = H$ for all $1 \leq i \leq m$. By Corollary 1, we can find $m_{\mathcal{A}}$ and $m_{\mathcal{B}}$ such that for all $m \geq m_{\mathcal{A}}$ and $m \geq m_{\mathcal{B}}$ and all trace distributions $K_1, K_2, \dots, K_m$ in $\text{trd}(\mathcal{B}, k)$

$$\mathbf{P}_{H^m}[\{O \in (\text{Act}^k)^m \mid \text{freq}(O) \in B_{\frac{d}{3}}(\mathbf{E}_{H^m})\}] > 1 - \alpha$$

and

$$\mathbf{P}_{K_1, \dots, K_m}[\{O \in (\text{Act}^k)^m \mid \text{freq}(O) \in B_{\frac{d}{3}}(\mathbf{E}_{K_1, \dots, K_m})\}] > 1 - \alpha.$$

Hence, $\mathcal{K}_{H^m} \subseteq B_{\frac{d}{3}}(\mathbf{E}_{H^m}) = B_{\frac{d}{3}}(\mu_H)$. On the other hand, for $1 \leq i \leq m$, let $E_i \in \text{trd}(\mathcal{B}, k)$ be such that $K_i = \text{trd}(E_i)$ and take $K = \text{trd}(\frac{1}{m} \sum_{i=1}^m E_i)$. One easily shows that $\mathbf{E}_{K_1, \dots, K_m} = \mathbf{E}_{K^m} = \mu_K^m$. Therefore, $\mathcal{K}_{K_1, \dots, K_m} \subseteq B_{\frac{d}{3}}(\mathbf{E}_{K_1, \dots, K_m}) = B_{\frac{d}{3}}(\mu_K)$. Since $|\mu_H - \mu_K| \geq d > 0$, we have $B_{\frac{d}{3}}(\mu_H) \cap B_{\frac{d}{3}}(\mu_K) = \emptyset$, and therefore, $\mathcal{K}_{H^m} \cap \mathcal{K}_{K_1, \dots, K_m} = \emptyset$. Hence, none of the observations in $\mathcal{K}_{H^m}$ is an observation of $\mathcal{B}$, i.e. $\text{Obs}(\mathcal{A}) \not\subseteq \text{Obs}(\mathcal{B})$. $\square$

The Equivalence Problem for t -Turn DPDA Is Co-NP

Géraud Sénizergues

LaBRI and Université de Bordeaux I*

Abstract. We introduce new tools allowing to deal with the equality-problem for prefix-free languages. We illustrate our ideas by showing that, for every fixed integer $t \geq 1$, the equivalence problem for t -turn deterministic pushdown automata is co-NP. This complexity result refines those of [Val74, Bee76].

Keywords: deterministic pushdown automata; equivalence problem; complexity; matrix semi-groups.

1 Introduction

Summary. The so-called “equivalence problem for deterministic pushdown automata” (denoted by $\text{Eq}(D_0, D_0)$ for short) is the following decision problem:

INSTANCE : two dpda A, B ; QUESTION : $L(A) = L(B)$?

i.e. do the given automata recognize the *same* language? This problem was shown to be decidable in ([Sén97],[Sén01a, sections 1-9]). This decidability result has been generalised in ([Sén01a, section 11],[Sén98],[Sén99]) and some simplifications of the method presented in [Sén97] have been found in [Sti99, Sti01]. Nevertheless, the intrinsic complexity of this problem is far from being understood. A first progress in this direction has been achieved in [Sti02] by showing that $\text{Eq}(D_0, D_0)$ is primitive recursive. We present here some new tools allowing to tackle this question.

General motivation. The equivalence problem for d.p.d.a., which was at first a kind of puzzle raised in [GG66], became a challenging important problem while people established links with the equivalence problem for program schemes, in the 70's. Since these years other links between automata on one side and rewriting systems, infinite graphs, formal power series, on the other side, have been found. A detailed description of this “connection” process and of the connections themselves is given in [Sén01a, p.3-5,155-158] and in [Sén01b]. More recently the study of pushdown automata of level k has demonstrated connections between

* mailing adress:LaBRI and UFR Math-info, Université Bordeaux1 351 Cours de la libération -33405- Talence Cedex. email:ges@labri.u-bordeaux.fr; fax: 05-56-84-66-69; URL: <http://dept-info.labri.u-bordeaux.fr/~ges/>

these automata and higher-level program-schemes ([KNU02]) and also some connections with the study of recurrent integer sequences ([FS03]).

As a consequence, any progress in the understanding of the *structure* and the *complexity* of the problem $\text{Eq}(D_0, D_0)$ is likely to have some impact on all the areas quoted above.

Framework. Following (and generalising) the point of view of [HHY79], we represent the computations of a dpda, via the notion of strict-deterministic grammar $G = \langle X, V, P \rangle$, as a right-action $\odot$ of X^* over a subset of matrices of “polynomials” over the set of variables V . Every equation, generated from an initial equation $v_1 \equiv v_2$, can be put in the form:

$$\alpha \cdot A_1 A_2 \cdots A_\lambda \cdot S \equiv \beta \cdot A_1 A_2 \cdots A_\lambda \cdot S, \quad (1)$$

where the A_i are square matrices, α, β (resp. S) are row-vectors (resp. a column-vector).

New tools. The first new tool introduced here is lemma 7 which states a property of this algebra of matrices: if λ is the dimension of the matrices A_i and all the equations obtained by removing some of the matrices (the same on both sides) are valid, then the equation (1) must be valid too.

A second ingredient allowing to cut down the complexity of comparison algorithms is the following observation: suppose that equation (1) occurred after λ successive “stacking” derivations. Then, the smaller equations corresponding to the $2^\lambda - 1$ strict subwords of the word $A_1 A_2 \dots A_\lambda$ must occur, if not on the same branch, in the same comparison-tree.

We introduce (in section 4) a notion of deduction relation which can be seen as an “extended semi-ring congruence closure”. Our lemma 3 expresses a commutation between this deduction relation and the right-action $\odot$.

Lemma 7 and lemma 3 are *structural* results. We strongly believe that, beside the single application given in next paragraph, they will be useful for dropping the complexity of several equivalence problems and, as well, for establishing new decidability results.

Result. We choosed to illustrate these new tools on the class of t -turn pushdown automata: it consists of the d.p.d.a. where every computation has at most t “turns”, i.e. changes of direction of the pushdown moves. It has been established in [Val74] that the equivalence problem for t -turn d.p.d.a. is decidable and [Bee76] proved that it belongs to $\text{DTIME}(2^{2^{c \cdot t}})$. We obtain here a *polynomial* upper-bound on the divergence of two non-equivalent t -turn dpda (theorem 14). It follows that the equivalence problem for t -turn dpda is in co-NP (corollary 15). Whether this problem is co-NP *complete* is left open.

Contents. Section 2 describes our general framework. We show in section 3 our crucial lemma: the “subwords lemma”. We introduce in section 4 our deduction relation. In section 5, we show the main result: the divergence of two finite-turn dpda A, B is upper-bounded by a polynomial function of the size of (A, B) .

2 Preliminaries

2.1 Grammars

Let us recall [Har78, definition 11.4.1].

Definition 1 Let $G = \langle X, V, P \rangle$ be a context-free grammar. G is said *strict-deterministic* iff there exists an equivalence relation $\sim$ over $X \cup V$ fulfilling the following conditions:

- 1- X is a class (mod $\sim$)
- 2- for every $v, v' \in V, \alpha, \beta, \beta' \in (X \cup V)^*$, if $v \rightarrow_P \alpha \cdot \beta$ and $v' \rightarrow_P \alpha \cdot \beta'$ and $v \sim v'$, then either:
 - 2.1- both $\beta, \beta' \neq \epsilon$ and $\beta[1] \sim \beta'[1]$ (mod $\sim$)
 - 2.2- or $\beta = \beta' = \epsilon$ and $v = v'$.

(In the above definition, $\gamma[1]$ denotes the first letter of the word γ). Any equivalence $\sim$ satisfying the above condition is said to be a *strict equivalence* for the grammar G . The grammar G is said *normalised* iff, in addition, every rule $(v, \gamma) \in P$ is such that $\gamma \in X \cup (X \cdot V) \cup (X \cdot V \cdot V)$. In what follows, we consider normalised grammars only. It is well-known that every strict-deterministic grammar can be reduced in such a normalised form, in polynomial time.

2.2 Right-Actions

We use the framework of formal power series: $(\mathbb{B}\langle\langle W \rangle\rangle, +, \cdot, 0, 1)$ denotes the semi-ring of *boolean series* over W , which is isomorphic to the semi-ring $(\mathcal{P}(W^*), \cup, \cdot, \emptyset, \epsilon)$; similarly $\mathbb{B}\langle W \rangle$ denotes the sub-semi-ring of polynomials over the undeterminates W . The *length* of a polynomial S is defined by: $|S| = \max\{|u| \mid u \in W^*, S_u = 1\}$. The reader is referred to [Sén02, section 2.3] for all details concerning right-actions of a monoid over a semi-ring.

Residual action. We recall the following classical σ -right-action $\bullet$ of the monoid W^* over the semi-ring $\mathbb{B}\langle\langle W \rangle\rangle$: for all $S, S' \in \mathbb{B}\langle\langle W \rangle\rangle, u \in W^*$

$$S \bullet u = S' \Leftrightarrow \forall w \in W^*, (S'_w = S_{u \cdot w}),$$

(i.e. $S \bullet u$ is the *left-quotient* of S by u , or the *residual* of S by u).

Grammatical action. Let $(V, \sim)$ be the structured alphabet associated with some normalised strict-deterministic grammar $G = \langle X, V, P \rangle$. We define the right-action $\odot$ as the unique σ -right-action of the monoid X^* over the semi-ring $\mathbb{B}\langle\langle V \rangle\rangle$ such that: for every $v \in V, \beta \in V^*, x \in X$,

$$(v \cdot \beta) \odot x = \left(\sum_{(v, h) \in P} h \bullet x \right) \cdot \beta, \quad \epsilon \odot x = \emptyset.$$

2.3 Equivalence

Let us consider the unique substitution $\varphi : \mathbb{B}\langle V \rangle \rightarrow \mathbb{B}\langle X \rangle$ fulfilling: for every $v \in V$, $\varphi(v) = \{u \in X^* \mid v \xrightarrow{*}_P u\}$. In other words, φ maps every subset $L \subseteq V^*$ on the language generated by the grammar G from the set of axioms L . We denote by $\equiv$ the kernel of φ : for every $S, T \in \mathbb{B}\langle W \rangle$, $S \equiv T \Leftrightarrow \varphi(S) = \varphi(T)$.

For every integer n , we introduce the relations $=_n$ over $\mathbb{B}\langle X \rangle$ and $\equiv_n$ over $\mathbb{B}\langle V \rangle$ defined by:

$$U =_n U' \Leftrightarrow U \cap X^{\leq n} = U' \cap X^{\leq n}; \quad S \equiv_n S' \Leftrightarrow \varphi(S) =_n \varphi(S').$$

The equivalence relation $\equiv$ (resp. $=_n, \equiv_n$) is extended, componentwise, to matrices (see §2.4).

2.4 Matrices

Let us call a structured alphabet any pair $(W, \sim)$ such that $\sim$ is an equivalence relation over W .

The equivalence relation $\sim$ is extended to W^* by: for every $w_1, w_2 \in W^*$, $w_1 \sim w_2$ iff either $w_1 = w_2$ or there exists $w \in W^*$, $v_1, v_2 \in W$, $w'_1, w'_2 \in W^*$ such that

$$w_1 = wv_1w'_1, w_2 = wv_2w'_2, v_1 \neq v_2, v_1 \sim v_2.$$

Let us denote by $\mathbb{B}_{n,m}\langle W \rangle$ the set of (n, m) -matrices with entries in the semi-ring $\mathbb{B}\langle W \rangle$.

Definition 2 Let $S \in \mathbb{B}_{n,m}\langle W \rangle$. S is said deterministic iff, for every $i \in [1, n]$, $j, k \in [1, m]$, $w, w' \in W^*$,

- 1- $w \in S_{i,j}, w \in S_{i,k} \Rightarrow j = k$
- 2- $w \in S_{i,j}, w' \in S_{i,k} \Rightarrow w \sim w'$

The set of deterministic matrices in $\mathbb{B}_{n,m}\langle W \rangle$ is denoted by $\mathbb{DB}_{n,m}\langle W \rangle$. The right-actions $\bullet$ and $\odot$ are extended componentwise to matrices.

Lemma 3 Let $S \in \mathbb{DB}_{1,m}\langle W \rangle, T \in \mathbb{B}_{m,s}\langle W \rangle, u \in W^*$. Exactly one of the following cases is true:

- 1- $\exists j, S_j \bullet u \notin \{\emptyset, \epsilon\}$; in this case $(S \cdot T) \bullet u = (S \bullet u) \cdot T$.
- 2- $\exists j_0, \exists u', u'', u = u' \cdot u'', S_{j_0} \bullet u' = \epsilon$; in this case $(S \cdot T) \bullet u = T_{j_0} \bullet u''$.
- 3- $\forall j, \forall u' \preceq u, S_j \bullet u = \emptyset, S_j \bullet u' \neq \epsilon$; in this case $(S \cdot T) \bullet u = \emptyset = (S \bullet u) \cdot T$.

Lemma 4 For every $S \in \mathbb{DB}_{n,m}\langle W \rangle, T \in \mathbb{DB}_{m,s}\langle W \rangle, u \in W^*$,

- 1- $S \cdot T \in \mathbb{DB}_{n,s}\langle W \rangle$.
- 2- $S \bullet u \in \mathbb{DB}_{n,m}\langle W \rangle$.

Both lemma 3,4 still hold for $W = V, u \in X^*$ and the action $\odot$ (instead of $\bullet$). Let us introduce an operation on row-vectors. Given $S \in \mathbb{DB}_{1,m}\langle W \rangle$ and

$1 \leq j_0 \leq m$ we define the vector $S' = \nabla_{j_0}^*(S)$ as follows:
if $S = (a_1, \dots, a_j, \dots, a_m)$ then $S' = (a'_1, \dots, a'_j, \dots, a'_m)$ where

$$a'_j = a_{j_0}^* \cdot a_j \text{ if } j \neq j_0, \quad a'_j = \emptyset \text{ if } j = j_0.$$

Lemma 5 *Let $S \in \mathbb{D}\mathbb{B}_{1,m} \langle \langle W \rangle \rangle$ and $1 \leq j_0 \leq m$.
Then $\nabla_{j_0}^*(S) \in \mathbb{D}\mathbb{B}_{1,m} \langle \langle W \rangle \rangle$.*

2.5 Matrices Expressing Derivations

Let us define here handful notations in order to describe derivations of a grammar within a matricial formalism. For every $1 \leq n, 1 \leq i \leq n$, we define the row-vectors $\epsilon_i^n, \emptyset^n$ as: $\epsilon_i^n = (\epsilon_{i,j}^n)_{1 \leq j \leq n}$ where

$$\epsilon_{i,j}^n = \emptyset \text{ (if } i \neq j\text{)}; \quad \epsilon_{i,i}^n = \epsilon; \quad \emptyset^n = (\emptyset, \dots, \emptyset).$$

Given a normalised strict-deterministic grammar G (see §2.1) we fix some system of representatives for the equivalence $\sim$, restricted to V : $\mathcal{E} = \{E_1, \dots, E_q\}$.

We let $N = \text{Card}(V)$, $N_i = \text{Card}([E_i]_{\sim})$, $[E_i]_{\sim} = \{E_{i,1}, E_{i,2}, \dots, E_{i,N_i}\}$. We define the row-vectors:

$$[E_i] = (0, \dots, 0, E_{i,1}, E_{i,2}, \dots, E_{i,N_i}, 0, \dots, 0)$$

where $E_{i,1}$ is placed in column $N_1 + N_2 + \dots + N_{i-1} + 1$. For every class $[E_i]_{\sim}$, and every letter $x \in X$, one of the three cases is realised:

$$[E_i] \odot x = [E_j] \cdot M_{i,x}, \text{ for some } M_{i,x} \in \mathbb{D}\mathbb{B}_{N,N} \langle \langle V \rangle \rangle \quad (2)$$

where at least one line of $M_{i,x}$ with index $k \in [(\sum_{\ell=1}^{i-1} N_\ell) + 1, \sum_{\ell=1}^i N_\ell]$ has one entry of length ≥ 1 , or

$$[E_i] \odot x = [E_j] \cdot M_{i,x}, \text{ for some } M_{i,x} \in \mathbb{D}\mathbb{B}_{N,N} \langle \langle V \rangle \rangle \quad (3)$$

where all the lines of $M_{i,x}$, are either null or equal to some ϵ_k^N , or

$$[E_i] \odot x = \epsilon_j^N, \text{ where } j \in [(\sum_{\ell=1}^{i-1} N_\ell) + 1, \sum_{\ell=1}^i N_\ell]. \quad (4)$$

2.6 Derivations

For every $S, S' \in \mathbb{D}\mathbb{B}_{1,\lambda} \langle \langle V \rangle \rangle$ and every $x \in X$, such that $S \neq \epsilon_i^\lambda$ (for every $i \in [1, \lambda]$) and $S \odot x = S'$, we must have

$$S = [E_i] \cdot T, \quad S' = ([E_i] \odot x) \cdot T,$$

for some $i \in [1, q]$ and some $T \in \mathbb{D}\mathbb{B}_{1,\lambda} \langle \langle V \rangle \rangle$.

We write $S \uparrow (x) S'$ if the couple $([E_i], x)$ fulfills condition (2) or (3).

We write $S \downarrow (x)S'$ if the couple $([E_i], x)$ fulfills condition(3) or (4).
Given a word $u = x_1x_2 \cdots x_\ell$, the notation $S \uparrow (u)S'$ means that:

$$S \uparrow (x_1)S \odot x_1, S \odot x_1 \uparrow (x_2)S \odot x_1x_2, \dots, S \odot x_1x_2 \cdots x_{\ell-1} \uparrow (x_\ell)S'.$$

The notation $S \downarrow (u)S'$ is defined similarly from the one-step relation $\downarrow$.

Let us notice that, when simultaneously $S_1 \uparrow (x)S'_1$ and $S_2 \uparrow (x)S'_2$, then

$$S_1 = \alpha_1 \cdot S, S_2 = \alpha_2 \cdot S; S'_1 = \alpha'_1 \cdot M \cdot S, S'_2 = \alpha'_2 \cdot M \cdot S, \quad (5)$$

where

$$\alpha_1 = ([E_{i_1}], 0^N), \alpha_2 = (0^N, [E_{i_2}]), \alpha'_1 = ([E_{j_1}], 0^N), \alpha'_2 = (0^N, [E_{j_2}]),$$

$$M = \begin{pmatrix} M_{i_1, x} & 0 \\ 0 & M_{i_2, x} \end{pmatrix}, S = \begin{pmatrix} T_1 \\ T_2 \end{pmatrix}.$$

A sequence of deterministic row-vectors $S_0, S_1, \dots, S_\ell$ is a *derivation* iff there exist $x_1, \dots, x_\ell \in X$ such that $S_0 \odot x_1 = S_1, \dots, S_{n-1} \odot x_\ell = S_\ell$. We denote this derivation by $S_0 \xrightarrow{u} S_\ell$.

A derivation $S_0, S_1, \dots, S_\ell$ is said to be *increasing* (resp. *decreasing*) iff it is the derivation associated to a pair (S, u) such that $S = S_0$ and $S_0 \uparrow (u)S_n$ (resp. $S_0 \downarrow (u)S_n$).

3 Systems of Equations

We recall the *divergence* between two languages $S, S' \subseteq X^*$ is defined by $\text{Div}(S, S') = \inf\{|u| \mid u \in S \Delta S'\}$ (where Δ denotes the symmetric difference operation). The *valuation* of a language S can be defined as $\text{Val}(S) = \text{Div}(S, \emptyset)$.

Lemma 6¹ *Let $\alpha, \beta, S, T \in \mathbb{DB}\langle\langle X \rangle\rangle, u \in X^*$. The following relations hold:*

$$1\text{-}\text{Div}(\alpha S, \beta S) = \text{Div}(\alpha, \beta) + \text{Val}(S)$$

$$2\text{-}\text{Div}(\alpha S, \alpha T) = \text{Val}(\alpha) + \text{Div}(S, T)$$

$$3\text{-}\text{Val}(S \cdot T) = \text{Val}(S) + \text{Val}(T)$$

$$4\text{-}\text{Div}(S, T) \leq \text{Div}(S \bullet u, T \bullet u) + |u|$$

$$5\text{-}\text{Div}(S, \alpha \cdot S + \beta) \leq \text{Div}(S, \alpha^* \beta)$$

$$(if (\alpha, \beta) \in \mathbb{DB}_{1,2}\langle\langle X \rangle\rangle, \alpha \neq \epsilon).$$

Lemma 7 (subwords lemma) *Let $\lambda \in \mathbb{N}, n \in \mathbb{N}, \alpha, \beta \in \mathbb{DB}_{1,\lambda}\langle\langle X \rangle\rangle, A_1, A_2, \dots, A_\lambda \in \mathbb{DB}_{\lambda,\lambda}\langle\langle X \rangle\rangle, S \in \mathbb{DB}_{\lambda,1}\langle\langle X \rangle\rangle$. If it holds that*

$$\alpha \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S =_n \beta \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S$$

$$for 0 \leq p \leq \lambda - 1, 1 \leq i_1 < i_2 < \dots < i_p \leq \lambda$$

then the following equation holds too:

$$\alpha \cdot A_1 A_2 \cdots A_\lambda \cdot S =_n \beta \cdot A_1 A_2 \cdots A_\lambda \cdot S.$$

¹ See [Sén03, p.8-9] for a proof, which is pure routine, anyway.

Due to the space requirements, we can only outline the proof².

Sketch of proof: We prove the lemma by induction on λ .

Basis: $\lambda=1$.

By lemma 1, $\text{Div}(\alpha AS, \beta AS) = \text{Div}(\alpha, \beta) + \text{Val}(AS) = \text{Div}(\alpha, \beta) + \text{Val}(A) + \text{Val}(S) = \text{Div}(\alpha S, \beta S) + \text{Val}(A)$. Hence, for every $n \geq 0$, $\alpha S =_n \beta S \Rightarrow \alpha AS =_n \beta AS$.

Induction step: $\lambda \rightarrow \lambda + 1$:

Let us suppose that, for every $0 \leq p \leq \lambda$, $1 \leq i_1 < i_2 < \dots < i_p \leq \lambda + 1$,

$$\alpha \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S =_n \beta \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S \quad (6)$$

Let $u \in X^*$, $|u| \leq n$.

Case 1: $\forall v \preceq u, \forall j \in [1, \lambda + 1], \alpha \bullet v \neq \epsilon_j^{\lambda+1}, \beta \bullet v \neq \epsilon_j^{\lambda+1}$.

Then, by lemma 3, points 1,3,

$$(\alpha \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S) \bullet u = (\alpha \bullet u) \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S;$$

$$(\beta \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S) \bullet u = (\beta \bullet u) \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S$$

$$\text{hence } (\alpha \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S) \bullet u =_0 (\beta \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S) \bullet u.$$

Case 2: $\exists v \preceq u, \exists j \in [1, \lambda + 1], \alpha \bullet v = \epsilon_j^{\lambda+1} = \beta \bullet v$.

Then, by lemma 3, point 2, letting w be the suffix such that $u = v \cdot w$, we have

$$(\alpha \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S) \bullet u = (\epsilon_j^{\lambda+1} \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S) \bullet w = (\beta \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S) \bullet u.$$

Case 3: $\exists v \preceq u$ such that

$$\exists j \in [1, \lambda + 1], \alpha \bullet v = \epsilon_j^{\lambda+1} \Leftrightarrow \beta \bullet v \neq \epsilon_j^{\lambda+1}. \quad (7)$$

Let $v \preceq u$ be the smallest prefix of u fulfilling (7). Without l.o.g. we may suppose that

$$\alpha \bullet v = \epsilon_{j_0}^{\lambda+1}, \beta \bullet v \neq \epsilon_{j_0}^{\lambda+1} \text{ and } j_0 = \lambda + 1.$$

By lemma 3 it follows that, for every $0 \leq p \leq \lambda$, $1 \leq i_1 < i_2 < \dots < i_p \leq \lambda + 1$,

$$(\alpha \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S) \bullet v = \epsilon_{j_0}^{\lambda+1} \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S \quad (8)$$

$$(\beta \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S) \bullet v = (\beta \bullet v) \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S \quad (9)$$

Note $n' = n - |v|$ and letting v act (by $\bullet$) on both sides of equations (6):

$$\epsilon_{\lambda+1}^{\lambda+1} \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S =_{n'} (\beta \bullet v) \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S. \quad (10)$$

Let $\gamma \in \mathbb{D}\mathbb{B}_{1,\lambda} \langle\langle X \rangle\rangle$ defined by:

$$\gamma = ((\beta_{\lambda+1} \bullet v)^*(\beta_1 \bullet v), (\beta_{\lambda+1} \bullet v)^*(\beta_2 \bullet v), \dots, (\beta_{\lambda+1} \bullet v)^*(\beta_\lambda \bullet v))$$

Each equation (10) gives rise to the equation

$$\epsilon_{\lambda+1}^{\lambda+1} \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S =_{n'} (\gamma, 0) \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S. \quad (11)$$

² see [Sén03, p.10-12] for a detailed proof

Let us introduce $P = \begin{pmatrix} I_\lambda \\ \gamma \end{pmatrix} \in \mathbb{DB}_{\lambda+1,\lambda} \langle\langle X \rangle\rangle$ and $\bar{P} = (I_\lambda \ 0_\lambda^1) \in \mathbb{DB}_{\lambda,\lambda+1} \langle\langle X \rangle\rangle$. Each equation (11) can be rewritten as:

$$A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S =_{n'} P \cdot \bar{P} \cdot A_{i_1} A_{i_2} \cdots A_{i_p} \cdot S. \quad (12)$$

Let us consider one equation (10) where $i_1 = 1, p \leq \lambda$. In such a single equation, taking into account the different equations (12) associated with all the suffixes of the sequence $i_2, i_3, \dots, i_p$, we obtain the new equation:

$$\begin{aligned} & (\epsilon_{\lambda+1}^{\lambda+1} A_1) (P \bar{P} A_{i_2}) \cdot (P \bar{P} A_{i_3}) \cdots (P \bar{P} A_{i_p}) \cdot (P \bar{P} S) = \\ & n' (\beta \bullet v A_1) (P \bar{P} A_{i_2}) \cdot (P \bar{P} A_{i_3}) \cdots (P \bar{P} A_{i_p}) \cdot (P \bar{P} S). \end{aligned}$$

which can be bracketed, as well,

$$\begin{aligned} & (\epsilon_{\lambda+1}^{\lambda+1} A_1 P) \cdot (\bar{P} A_{i_2} P) \cdot (\bar{P} A_{i_3} P) \cdots (\bar{P} A_{i_p} P) \cdot (\bar{P} S) = \\ & n' (\beta \bullet v A_1 P) \cdot (\bar{P} A_{i_2} P) \cdot (\bar{P} A_{i_3} P) \cdots (\bar{P} A_{i_p} P) \cdot (\bar{P} S). \end{aligned} \quad (13)$$

Let us take:

$$\alpha' = \epsilon_{\lambda+1}^{\lambda+1} A_1 P, \beta' = (\beta \bullet v) A_1 P, A'_j = \bar{P} A_j P \text{ (for } 2 \leq j \leq \lambda + 1), S' = \bar{P} S.$$

The items $n', \alpha', \beta', A'_j (2 \leq j \leq \lambda + 1), S'$ are fulfilling the hypothesis of the lemma for the integer λ . By induction hypothesis, it must be true that

$$\alpha' \cdot A'_2 A'_3 \cdots A'_{\lambda+1} \cdot S' =_{n'} \beta' \cdot A'_2 A'_3 \cdots A'_{\lambda+1} \cdot S'. \quad (14)$$

Using now the equations (12) “backwards”, we succeed to derive from equation (14) the conclusion that:

$$(\alpha \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S) \bullet u =_0 (\beta \cdot A_1 A_2 \cdots A_{\lambda+1} \cdot S) \bullet u.$$

As this is true for every $|u| \leq n$, the lemma is proved. $\square$

4 Deduction Rules

4.1 The Deduction Relation

We denote by $\mathcal{A}$ the set $\mathbb{DB} \langle\langle V \rangle\rangle \times \mathbb{DB} \langle\langle V \rangle\rangle$.

We define then a binary relation $\vdash\!\!\!-\subseteq \mathcal{P}(\mathcal{A}) \times \mathcal{A}$, the *elementary deduction relation*, as the set of all the pairs having one of the following forms:

(R0) $\emptyset$ (R1) $\{(T, T')\}$ (R2) $\{(T, T'), (T', T'')\}$ (R3) $\{(S_1, T_1), (S_2, T_2)\}$ (R4) $\{(T, T')\}$ (R5) $\{(T, T')\}$ (R6) $\{(U_1 \cdot T + U_2, T)\}$ (R7) $\{(\varepsilon, U_1)\}$ (R8) $\{(\alpha A_{i_1} A_{i_2} \cdots A_{i_p} S, \beta A_{i_1} A_{i_2} \cdots A_{i_p}) \mid 1 \leq i_1 < i_2 < \dots < i_p \leq n, 0 \leq p \leq n-1\}$	$\vdash\!\!\!- (T, T)$ $\vdash\!\!\!- (T', T)$ $\vdash\!\!\!- (T, T'')$ $\vdash\!\!\!- (S_1 + S_2, T_1 + T_2)$ $\vdash\!\!\!- (T \cdot U, T' \cdot U)$ $\vdash\!\!\!- (U \cdot T, U \cdot T')$ $\vdash\!\!\!- (U_1^* \cdot U_2, T)$ $\vdash\!\!\!- (T', T'')$ $\vdash\!\!\!- (\alpha A_1 A_2 \cdots A_n S, \beta A_1 A_2 \cdots A_n S)$

for $T, T', T'', U \in \mathbb{D}\mathbb{B}\langle\langle V \rangle\rangle, (S_1, S_2), (T_1, T_2), (U_1, U_2) \in \mathbb{D}\mathbb{B}_{1,2}\langle\langle V \rangle\rangle, U_1 \neq \epsilon, \alpha, \beta \in \mathbb{D}\mathbb{B}_{1,n}\langle\langle V \rangle\rangle, A_1, A_2, \dots, A_n \in \mathbb{D}\mathbb{B}_{n,n}\langle\langle V \rangle\rangle, S \in \mathbb{D}\mathbb{B}_{n,1}\langle\langle V \rangle\rangle$. (It follows from §2.4, that the above rules really belong to $\mathcal{P}(\mathcal{A}) \times \mathcal{A}$).

Finally, the binary relation $\vdash \subseteq \mathcal{P}(\mathcal{A}) \times \mathcal{P}(\mathcal{A})$ is defined by: $\forall P, Q \in \mathcal{P}(\mathcal{A})$

$$P \vdash Q \Leftrightarrow (\forall q \in Q - P, \exists P' \subseteq P, \text{ such that } P' \vdash q).$$

By $\vdash^*$ we denote the reflexive and transitive closure of $\vdash$, we call it the *deduction relation*.

4.2 Properties

Lemma 8 : For every $P, Q \in \mathcal{P}(\mathcal{A})$, $P \vdash^* Q \Rightarrow \text{Div}(P) \leq \text{Div}(Q)$.

This follows easily from lemma 1, point 5 and lemma 7.

Lemma 9 ³ For every $P, Q \in \mathcal{P}(\mathcal{A}), x \in X$, if $(P \vdash^* Q)$ then,

$$(P \cup P \odot x \vdash^* Q \odot x).$$

The most delicate point is to treat the case where $P \vdash_{R8}^* \{q\}$. It consists essentially in following the flow of arguments of the proof of lemma 7 and replace every “external” algebraic argument by some sequence of applications of the rules R0-R8.

4.3 Self-Provable Sets

A subset $P \subseteq \mathcal{A}$ is said: ε -consistent iff $\forall (S, T) \in P, (S = \varepsilon) \Leftrightarrow (T = \varepsilon)$;

right-stable iff $\forall x \in X, P \vdash^* P \odot x$; self-provable iff P is ε -consistent and right-stable.

Lemma 10 If P is self-provable then $\text{Div}(P) = \infty$.

5 Application to t -Turn dpda

We show here that the divergence of two t -turn d.p.d.a. is upper-bounded by some polynomial function of the size of the automata.

5.1 Turns and Weights

Definition 11 Let $G = \langle X, V, P \rangle$ be a normalised strict-deterministic context-free grammar and let k be an integer. G is said to be k -weighted iff there exists a map $\tau : V \rightarrow [0, k]$ such that every rule of P has one of the following forms:

1- $v \rightarrow x \cdot v_1 v_2$, with $v, v_1, v_2 \in V, x \in X, \tau(v) \geq \tau(v_1) + \tau(v_2)$ and $\tau(v_1) \geq 1$

2- $v \rightarrow x \cdot v_1$, with $v, v_1 \in V, x \in X, \tau(v) \geq \tau(v_1)$

3- $v \rightarrow x$, with $v \in V, x \in X$.

³ Full proof in [Sén03, p.14-17]

One can check that k -weighted strict-deterministic context-free grammars are corresponding to $2k - 1$ -turn dpda and that this correspondence can be computed in P-time in both directions ⁴. We fix a k -weighted strict-deterministic c.f. grammar $G = \langle X, V, P \rangle$ within this section. We also fix two variables $v_1, v_2 \in V$ and deal with the equality problem for $L(G, v_1), L(G, v_2)$.

5.2 Parallel Derivations

We show that every long increasing derivation, must contain a sequence of equations which is a “germ” for an application of rule R8 (which is based on the subwords lemma).

Lemma 12 *Let us suppose that $S_1, S_2, S'_1, S'_2 \in \mathbb{DB}_{1,1}(\langle V \rangle), u \in X^*$ and $S_i \uparrow (u) S'_i$ (for $i \in \{1, 2\}$). If $|u| \geq 2N^3$, there exist $\alpha, \beta \in \mathbb{DB}_{1,2N}(\langle V \rangle), |\alpha| \geq 1, |\beta| \geq 1, u', u'' \in X^*, u_{2N}, u_{2N-1}, \dots, u_1 \in X^+, M_{2N}, M_{2N-1}, \dots, M_1 \in \mathbb{DB}_{2N,2N}(\langle V \rangle), S \in \mathbb{DB}_{2N,1}(\langle V \rangle)$ such that:*

- 1- $u' \cdot u_{2N} \cdot u_{2N-1} \cdots u_1 \cdot u'' = u$
- 2- $S_1 \uparrow (u') \alpha \cdot S \uparrow (u_{2N}) \alpha \cdot M_{2N} \cdot S \dots \uparrow (u_i) \alpha \cdot M_i M_{i+1} \cdots M_{2N} \cdot S \dots \uparrow (u_1) \alpha \cdot M_1 M_2 \cdots M_{2N} \cdot S \uparrow (u'') S'_1$
- 3- $S_2 \uparrow (u') \beta \cdot S \uparrow (u_{2N}) \beta \cdot M_{2N} \cdot S \dots \uparrow (u_i) \beta \cdot M_i M_{i+1} \cdots M_{2N} \cdot S \dots \uparrow (u_1) \beta \cdot M_1 M_2 \cdots M_{2N} \cdot S \uparrow (u'') S'_2$

Sketch of proof: It suffices to notice the form of the transitions (2,3), to use the trick mentionned in equation(5), and to apply the “pigeon-hole” principle. $\square$

5.3 A Right-Stable Set

For every integer $n \geq 0$ let us define $P_n = \{(v_1 \odot u, v_2 \odot u) \mid u \in X^{\leq n}\}$. Let $N_1 = 1 + 2 \cdot (1 + 2 \cdot N^3)^{4k}$.

Lemma 13 *The set P_{N_1} is right-stable.*

Sketch of proof: In step 1, we show that, for every $u \in X^{N_1}$, there exists a prefix $u' \preceq u$ such that $P_{|u'-1} \stackrel{*}{\vdash} \{(v_1 \odot u', v_2 \odot u')\}$. In step 2, we conclude that P_{N_1} is right-stable.

Step 1 Let $u_0 \in X^{N_1}$.

Case 1 Suppose that the word u_0 admits a decomposition $u_0 = u'_0 u u''_0$ such that, for every $i \in 1, 2$, $v_i \xrightarrow{u'_0} S_i \uparrow (u) S'_i \xrightarrow{u''_0} S_i$ with $|u| \geq 2N^3$.

In this case, points (1),(2),(3) of lemma 12 are true. Notice that, for every subsequence of indices: $1 \leq i_1 < i_2 < \dots < i_p \leq 2N, 0 \leq p \leq 2N - 1$, we have

$$\begin{aligned} (v_1 \odot (u'_0 \cdot u_{i_p} \cdots u_{i_2} \cdot u_{i_1}), v_2 \odot (u'_0 \cdot u_{i_p} \cdots u_{i_2} \cdot u_{i_1})) = \\ (\alpha M_{i_1} M_{i_2} \cdots M_{i_p} S, \beta M_{i_1} M_{i_2} \cdots M_{i_p} S) \end{aligned} \quad (15)$$

⁴ See subsection 5.1 of [Sén03].

Every lefthand-side of an identity (15) belongs to $P_{|u'_0 u| - 1}$. The set of all the righthand-sides of identities (15) allows to deduce the equation $(v_1 \odot u'_0 u, v_2 \odot u'_0 u)$, by rule R8. Therefore, the prefix $u' = u'_0 u \preceq u_0$ has the property that

$$P_{|u'| - 1} \vdash^* \{(v_1 \odot u', v_2 \odot u')\}. \quad (16)$$

Case 2 Let us suppose that the word u_0 admits no decomposition of the form assumed in case 1.

The whole parallel derivation $(v_1, v_2) \xrightarrow{u_0} (v_1 \cdot u_0, v_2 \cdot u_0)$ can be factorised into at most $4k$ derivations: $v_i = S_{i,0} \xrightarrow{u_1} S_{i,1} \xrightarrow{u_2} S_{i,2} \xrightarrow{u_3} S_{i,3} \cdots \xrightarrow{u_{4k}} S_{i,4k}$, (for every $i \in \{1, 2\}$), with $u = u_1 \cdot u_2 \cdots u_{4k}$, such that every derivation $S_{i,j} \xrightarrow{u_j} S_{i,j+1}$ is monotone i.e. either increasing or decreasing. Let us denote

$$H(j) = \max\{|S_{1,j}|, |S_{2,j}|\}; \quad T(j) = |u_j|.$$

A careful analysis of the four types of monotonicity ⁵ shows that:

$$H(0) = 1; \quad T(0) = 0; \quad H(j) \leq H(j-1) \cdot (1 + 2 \cdot N^3); \quad T(j) \leq H(j-1) \cdot (4 \cdot N^3).$$

It follows that $H(4k) \leq (1 + 2 \cdot N^3)^{4k}$ and $T(4k) < N_1$. Hence we would have $|u_0| \leq T(4k) < N_1$, which is impossible. It follows that case 1 must occur, which achieves step 1.

Step 2

Let $p \in P_{N_1}$ and $x \in X$. The equation p must have the form

$$p = (v_1 \odot u, v_2 \odot u) \text{ for some } u \in X^*, |u| \leq N_1.$$

If $|u| < N_1$, then $p \odot x = (v_1 \odot ux, v_2 \odot ux) \in P_{N_1}$. In this case $P_{N_1} \vdash^* \{p \odot x\}$. Suppose that $|u| = N_1$. In step 1 we established that there exists some decomposition $u = u' \cdot u''$ such that $P_{|u'| - 1} \vdash^* \{(v_1 \odot u', v_2 \odot u')\}$. Applying now $|u''| + 1$ times lemma 3, we obtain that: $P_{|u|} \vdash^* \{(v_1 \odot ux, v_2 \odot ux)\}$, i.e. $P_{N_1} \vdash^* \{p \odot x\}$. $\square$

Theorem 14 *There exists a constant $K \in \mathbb{N}$ such that, for every positive integer $k \geq 1$ and every strict-deterministic c.f. grammar $G = \langle X, V, P \rangle$ which is k -weighted and every $v_1, v_2 \in V$, it holds that, either $v_1 \equiv v_2$ or:*
 $\text{Div}(\text{L}(G, v_1), \text{L}(G, v_2)) \leq K \cdot (2 \cdot \|G\|)^{12 \cdot k}$.

Proof: Let us consider the subset P_{N_1} : either it is ε -consistent, and, by lemma 10, $v_1 \equiv v_2$; or it is not ε -consistent, which means that $\exists u \in X^{\leq N_1} \cap (\text{L}(G, v_1) \Delta \text{L}(G, v_2))$. $\square$

Corollary 15 *For every positive integer $k \geq 1$, the equivalence problem for k -weighted strict-deterministic c.f. grammars (resp. for $(2k-1)$ -turn deterministic pushdown automata) is in co-NP.*

⁵ See a full proof in subsection 5.3 of [Sén03]

Final remark: The above method can be pushed further in order to obtain an upper-bound on the divergence which is polynomial as a function of the size of the grammars and of the maximal weight k (see in subsection 6.1 of [Sén03] the precise statements, as well as other refinements).

References

- [Bee76] C. Beeri. An improvement on Valiant's decision procedure for equivalence of deterministic finite-turn pushdown automata. *TCS* 3, pages 305–320, 1976.
- [FS03] S. Fratani and G. Sénizergues. Iterated pushdown automata and sequences of rational numbers. Technical report, LaBRI, 2003. Draft. Pages 1–45. Available on the authors personal web-pages.
- [GG66] S. Ginsburg and S. Greibach. Deterministic context-free languages. *Information and Control*, pages 620–648, 1966.
- [Har78] M.A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Mass., 1978.
- [HHY79] M.A. Harrison, I.M. Havel, and A. Yehudai. On equivalence of grammars through transformation trees. *TCS* 9, pages 173–205, 1979.
- [KNU02] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*. LNCS, 2002.
- [Sén97] G. Sénizergues. The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In *Proceedings ICALP 97*, pages 671–681. Springer, LNCS 1256, 1997.
- [Sén98] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In Rajeev Motwani, editor, *Proceedings FOCS'98*, pages 120–129. IEEE Computer Society Press, 1998.
- [Sén99] G. Sénizergues. $T(A) = T(B)$? In *Proceedings ICALP 99*, volume 1644 of *LNCS*, pages 665–675. Springer-Verlag, 1999. Full proofs in technical report 1209–99 of LaBRI, $T(A) = T(B)$?, pages 1–61.
- [Sén01a] G. Sénizergues. $L(A) = L(B)$? decidability results from complete formal systems. *Theoretical Computer Science*, 251:1–166, 2001.
- [Sén01b] G. Sénizergues. Some applications of the decidability of dpda's equivalence. In *Proceedings MCU'01*, volume 2055 of *LNCS*, pages 114–132. Springer-Verlag, 2001.
- [Sén02] G. Sénizergues. $L(A) = L(B)$? a simplified decidability proof. *Theoretical Computer Science*, 281:555–608, 2002.
- [Sén03] G. Sénizergues. The equivalence problem for t -turn DPDA is co-NP. Technical report, LaBRI, 2003. Pages 1–26.
- [Sti99] C. Stirling. Decidability of dpda's equivalence. Technical report, Edinburgh ECS-LFCS-99-411, 1999. Pages 1–25.
- [Sti01] C. Stirling. Decidability of dpda's equivalence. *Theoretical Computer Science*, 255:1–31, 2001.
- [Sti02] C. Stirling. Deciding DPDA Equivalence is Primitive Recursive. In *Proceedings ICALP 02*, pages 821–832. Springer, LNCS 2380, 2002.
- [Val74] L.G. Valiant. The equivalence problem for deterministic finite-turn pushdown automata. *Information and Control* 25, pages 123–133, 1974.

Flip-Pushdown Automata: $k + 1$ Pushdown Reversals Are Better than k

Markus Holzer¹ and Martin Kutrib²

¹ Institut für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching bei München, Germany
holzer@informatik.tu-muenchen.de

² Institut für Informatik, Universität Gießen,
Arndtstraße 2, D-35392 Gießen, Germany
kutrib@informatik.uni-giessen.de

Abstract. Flip-pushdown automata are pushdown automata with the additional power to flip or reverse its pushdown, and were recently introduced by Sarkar [13]. We solve most of Sarkar's open problems. In particular, we show that $k + 1$ pushdown reversals are better than k for both deterministic and nondeterministic flip-pushdown automata, i.e., there are languages which can be recognized by a deterministic flip-pushdown automaton with $k + 1$ pushdown reversals but which cannot be recognized by a k -flip-pushdown (deterministic or nondeterministic). Furthermore, we investigate closure and non-closure properties as well as computational complexity problems such as fixed and general membership.

1 Introduction

A pushdown automaton is a one-way finite automaton with a separate pushdown store (PD), that is a last-in first-out (LIFO) storage structure, which is manipulated by pushing and popping. Probably, such machines are best known for capturing the family of context-free languages $\mathcal{L}(\text{CFL})$, which was independently established by Chomsky [4] and Evey [6]. Pushdown automata have been extended in various ways. Examples of extensions are variants of stacks [8], queues or dequeues, while restrictions are for instance counters or one-turn pushdowns [9]. The results obtained for these classes of machines hold for a large variety of formal language classes, when appropriately abstracted. This led to the rich theory of abstract families of automata (AFA), which is the equivalent of abstract families of languages (AFL) theory; for the general treatment of machines and languages we refer to Ginsburg [7].

In this paper, we consider a recently introduced extension of pushdown automata, so called flip-pushdown automata [13]. Basically, a flip-pushdown automaton is an ordinary pushdown automaton with the additional ability to flip its pushdown during the computation. This allows the machine to push and pop at both ends of the pushdown. Hence, a flip-pushdown is a form of a dequeue storage structure, and thus becomes equally powerful to Turing machines, since a dequeue automaton can simulate two pushdowns. On the other hand, if the

number of pushdown flips or pushdown reversals is zero, obviously the family of context-free languages is characterized. Thus it remains to investigate the number of pushdown reversals as a natural computational resource.

By Sarkar [13] it was shown that if the number of pushdown flips is bounded by a constant, then a nonempty hierarchy of language classes is introduced, and it was conjectured that the hierarchy is strict. Obviously, since by a single pushdown reversal one can accept the non-context-free language $\{ww \mid w \in \{a, b\}^*\}$, the base level of that hierarchy is already separated. But what about the other levels? In fact, in this paper we solve most of the open problems stated by Sarkar, especially the above mentioned one. More precisely, we show that $k + 1$ pushdown reversals are better than k for both deterministic and nondeterministic flip-pushdown automata. To this end, we develop a technique to decrease the number of pushdown reversals, which simply speaking shows that flipping the pushdown is equivalent to reverse part of the remaining input, hence calling our technique the “flip-pushdown input-reversal” theorem. An immediate consequence of this theorem is that every flip-pushdown language accepted by a flip-pushdown with a constant number of pushdown reversals obeys a semi-linear Parikh mapping.

Moreover, we also investigate closure and non-closure properties for the language families under consideration. It turns out, that the family of flip-pushdown languages share similar closure and non-closure properties as the family of context-free languages like, e.g., closure under intersection with regular sets, or the non-closure under complementation. Not surprisingly, the family of flip-pushdown languages is shown to be a full TRIO. Nevertheless, there are some interesting differences as, e.g., the non-closure under concatenation and Kleene star. Again, the flip-pushdown input-reversal theorem turns out to be very helpful in order to obtain the mentioned non-closure results.

Finally, computational complexity aspects of flip-pushdown languages with a constant number of pushdown reversals are considered. Again similarities to context-free languages are found. At first glance, we show that every flip-pushdown language accepted by a flip-pushdown automata with a constant number of pushdown reversals is context-sensitive. Moreover, it is proven that auxiliary flip-pushdown automata with exactly k pushdown reversals, i.e., a flip-pushdown automaton with a resource-bounded working-tape, capture P when their space is logarithmically bounded, and catch the important complexity class $\text{LOG(CFL)} \subseteq \text{P}$ when additionally their time is polynomially bounded. This nicely resembles the known results on auxiliary pushdown automata given by Cook [5] and Sudborough [14].

The paper is organized as follows: The next section contains preliminaries, and we show basics on flip-pushdown automata. Then Section 3 is devoted to our main technique, the flip-pushdown input-reversal theorem and its application in the separation of the flip-pushdown hierarchy for both deterministic and nondeterministic machines. The next section deals with closure and non-closure properties and in the penultimate Section 5 we investigate computational complexity aspects of flip-pushdown languages. Finally we summarize our results and highlight the remaining open questions in Section 6.

2 Definitions

We assume the reader to be familiar with the basics of formal language theory, where we refer to the book of Hopcroft and Ullman [10]. Consider the strict chain of inclusions $\mathcal{L}(\text{REG}) \subset \mathcal{L}(\text{CFL}) \subset \mathcal{L}(\text{CS}) \subset \mathcal{L}(\text{RE})$, where $\mathcal{L}(\text{REG})$ denotes the family of regular languages, $\mathcal{L}(\text{CFL})$ the family of context-free languages, $\mathcal{L}(\text{CS})$ the family of context-sensitive languages, and $\mathcal{L}(\text{RE})$ the family of recursively enumerable languages. Moreover, we also need some notions from complexity theory as contained in the book of Balcázar *et al.* [2].

In the following we consider pushdown automata with the ability to flip their pushdowns. These machines were recently introduced by Sarkar [13] and are defined as follows:

Definition 1. A (nondeterministic) flip-pushdown automaton (NFPDA) is a system $A = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, F)$, where Q is a finite set of states, Σ is the finite input alphabet, Γ is a finite pushdown alphabet, δ is a mapping from $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$ called the transition function, Δ is a mapping from Q to 2^Q , $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is a particular pushdown symbol, called the bottom-of-pushdown symbol, which initially appears on the pushdown store, and $F \subseteq Q$ is the set of final states.

A configuration or instantaneous description of a flip-pushdown automaton is a triple (q, w, γ) , where q is a state in Q , w a string of input symbols, and γ is a string of pushdown symbols. A flip-pushdown automaton A is said to be in configuration (q, w, γ) if A is in state q with w as remaining input, and γ on the pushdown store, the rightmost symbol of γ being the top symbol on the pushdown. If a is in $\Sigma \cup \{\lambda\}$, w in Σ^* , γ and β in Γ^* , and Z is in Γ , then we write $(q, aw, \gamma Z) \vdash_A (p, w, \gamma\beta)$, if the pair (p, β) is in $\delta(q, a, Z)$, for “ordinary” pushdown transitions and $(q, aw, Z_0\gamma) \vdash_A (p, aw, Z_0\gamma^R)$, if p is in $\Delta(q)$, for pushdown-flip or pushdown-reversal transitions. Whenever, there is a choice between an ordinary pushdown transition or a pushdown reversal, then the automaton nondeterministically chooses the next move. Observe, that we do not want the flip-pushdown automaton to move the bottom-of-pushdown symbol when the pushdown is flipped. As usual, the reflexive transitive closure of $\vdash_A$ is denoted by $\vdash_A^*$. The subscript A will be dropped from $\vdash_A$ and $\vdash_A^*$ whenever the meaning remains clear.

Let k be a natural number. For a flip-pushdown automaton A we define $T_k(A)$, the language accepted by final state and exactly k pushdown reversals¹, to be

$$T_k(A) = \{ w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_A^* (q, \lambda, \gamma) \text{ with exactly } k \\ \text{pushdown reversals, for any } \gamma \in \Gamma^* \text{ and } q \in F \}.$$

¹ One may define language acceptance of flip-pushdown automata with *at most* k pushdown reversals. Since a flip-pushdown automaton can count the number of reversals performed during its computation in its finite control, it is an easy exercise to show that these to language acceptance mechanisms coincide.

Also, we define $N_k(A)$, the language *accepted by empty pushdown and exactly k pushdown reversals*, to be

$$N_k(A) = \{ w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_A^* (q, \lambda, \lambda) \text{ with exactly } k \text{ pushdown reversals, for any } q \in Q \}.$$

If the number of pushdown reversals is not limited, the language accepted by final state (empty pushdown, respectively) is analogously defined as above and denoted by $T(A)$ ($N(A)$, respectively). When accepting by empty pushdown, the set of final states is irrelevant. Thus, in this case, we usually let the set of final states be the empty set.

In order to clarify our notation we give a small example.

Example 2. Let $A = (\{q_0, q_1\}, \{a, b\}, \{A, B, Z_0\}, \delta, \Delta, q_0, Z_0, \emptyset)$ be a flip-pushdown automaton where

1. $\delta(q_0, a, Z_0) = \{(q_0, Z_0A)\}$	6. $\delta(q_0, b, B) = \{(q_0, BB)\}$
2. $\delta(q_0, b, Z_0) = \{(q_0, Z_0B)\}$	7. $\delta(q_1, a, A) = \{(q_1, \lambda)\}$
3. $\delta(q_0, a, A) = \{(q_0, AA)\}$	8. $\delta(q_1, b, B) = \{(q_1, \lambda)\}$
4. $\delta(q_0, b, A) = \{(q_0, AB)\}$	9. $\delta(q_1, \lambda, Z_0) = \{(q_1, \lambda)\}$
5. $\delta(q_0, a, B) = \{(q_0, BA)\}$	

and $\Delta(q_0) = \{q_1\}$ that accepts by empty pushdown the non-context-free language $L = \{ww \mid w \in \{a, b\}^*\}$. This is seen as follows.

The transitions (1) through (6) allow A to store the input on the pushdown. If A decides that the middle of the input string has been reached, then the flip operation specified by $\Delta(q_0) = \{q_1\}$ is selected and A goes to state q_1 and tries to match the remaining input symbols with the reversed pushdown content. This is done with the transitions (7) and (8). Thus, if the guess of A was right, and the input is of the form ww , then the inputs will match, and A will empty its pushdown with transition (9), and therefore accept the input string (by empty pushdown).

The next theorem can be shown with a simple adaption of the proof for ordinary pushdown automata. Thus, we omit the proof.

Theorem 3. *Let k be some natural number. Then language L is accepted by some flip-pushdown automaton A_1 with empty pushdown making exactly k pushdown reversals, i.e., $L = N_k(A_1)$, if and only if language L is accepted by some flip-pushdown automaton A_2 by final state making exactly k pushdown reversals, i.e., $L = T_k(A_2)$. The statement remains valid for flip-pushdown automata with an unbounded number of pushdown reversals. $\square$*

The family of languages accepted by flip-pushdown automata with empty pushdown or equivalently by final state making exactly k or equivalently at most k pushdown reversals is denoted by $\mathcal{L}(\text{NFPDA}_k)$. Furthermore, we define $\mathcal{L}(\text{NFPDA}_{fin}) = \bigcup_{k=0}^{\infty} \mathcal{L}(\text{NFPDA}_k)$ and if the number of pushdown reversals is

unbounded, the corresponding language family is referred to $\mathcal{L}(\text{NFPDA})$. We recall the following theorem of Sarkar [13].

Theorem 4. $\mathcal{L}(\text{CFL}) = \mathcal{L}(\text{NFPDA}_0) \subseteq \mathcal{L}(\text{NFPDA}_1) \subseteq \dots \subseteq \mathcal{L}(\text{NFPDA}_{fin}) \subseteq \mathcal{L}(\text{NFPDA}) = \mathcal{L}(\text{RE})$.

An immediate question that arises from the previous theorem is, whether the hierarchy on pushdown reversals is a strict hierarchy, and whether the upper bound can be improved to the family of context-sensitive languages $\mathcal{L}(\text{CS})$. In the next sections we positively affirm these questions in the sense, that the hierarchy is strict and that the upper bound can be improved.

3 The Flip-Pushdown Input-Reversal Technique

In this section we prove an essential technique for flip-pushdown automata, which will be called “flip-pushdown input-reversal” since flipping the pushdown can be simulated by reversing the (remaining) input. The main theorem of this section reads as follows:

Theorem 5. *Let k be a natural number. Language L is accepted by a flip-pushdown $A_1 = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, \emptyset)$ by empty pushdown with $k+1$ pushdown reversals, i.e., $L = N_{k+1}(A_1)$, if and only if language*

$$L_R = \{ wv^R \mid (q_0, w, Z_0) \vdash_{A_1}^* (q_1, \lambda, Z_0\gamma) \text{ with } k \text{ reversals, } q_2 \in \Delta(q_1), \\ \text{and } (q_2, v, Z_0\gamma^R) \vdash_{A_1}^* (q_3, \lambda, \lambda) \text{ without any reversal} \}$$

is accepted by a flip-pushdown automaton A_2 by empty pushdown with k pushdown reversals, i.e., $L_R = N_k(A_2)$. The statement remains valid if state acceptance is considered.

In order to simplify presentation, we introduce the notion of a generalized flip-pushdown automaton $A = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, F)$, where $Q, \Sigma, \Gamma, \Delta, q_0 \in Q$, $Z_0 \in \Gamma$, and $F \subseteq Q$ are as in the case of ordinary flip-pushdown automata, and δ is a *finite domain* mapping from $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma^*$ to the finite subsets of $Q \times \Gamma^*$. With standard techniques one can construct an ordinary flip-pushdown automaton from a given generalized one, without increasing the number of pushdown-flips. Due to the ability to read words instead of symbols, the necessary checks, whether a push or pop action can be performed in the backward simulation becomes easier to describe.

Proof (of Theorem 5). We only prove the direction from left to right. The converse implication can be shown by similar arguments.

Let $A_1 = (Q, \Sigma, \Gamma, \delta, \Delta, q_0, Z_0, \emptyset)$ be a flip-pushdown automaton satisfying $\gamma \in \{\lambda\} \cup \{ZX \mid X \in \Gamma\}$ for all $(p, \gamma) \in \delta(q, a, Z)$, where $p, q \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $Z \in \Gamma$. This normal form can be easily achieved. Then we define a generalized flip-pushdown automaton

$$A_2 = (Q \cup \mathbf{Q} \cup \{\mathbf{q}_f\}, \Sigma, \Gamma \cup \Gamma \cup Q, \delta', \Delta', q_0, Z_0, \{\mathbf{q}_f\}),$$

where $\mathbf{Q} = \{\mathbf{q} \mid q \in Q\}$, $\Gamma = \{\mathbf{Z} \mid Z \in \Gamma\}$, and δ' and Δ' are specified as follows:

1. For all $q \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $Z \in \Gamma$, set $\delta'(q, a, Z)$ includes all elements of $\delta(q, a, Z)$ and
2. for all $q \in Q$, let $\Delta'(q)$ contain all elements of $\Delta(q)$.
3. For all $r \in Q$, if $\Delta(r) \neq \emptyset$, then $\delta'(r, a, Z)$ contains $(\mathbf{q}, Z Z_0 r \mathbf{Z}_0)$, where $q \in Q$ satisfies $(p, \lambda) \in \delta(q, a, Z_0)$ for some $p \in Q$ and $a \in \Sigma \cup \{\lambda\}$.
4. For all $p, q \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $X, Y \in \Gamma$, let $\delta'(\mathbf{q}, a, \mathbf{XY})$ contain $(\mathbf{p}, \mathbf{X})$ if $(q, XY) \in \delta(p, a, X)$.
5. For all $p, q, r \in Q$, $a \in \Sigma \cup \{\lambda\}$, and $X, Y \in \Gamma$, then
 - a) let $\delta'(\mathbf{q}, a, \mathbf{X})$ contain $(\mathbf{p}, \mathbf{XY})$ if $(q, \lambda) \in \delta(p, a, Y)$ and
 - b) let $\delta'(\mathbf{q}, a, Xr\mathbf{X})$ contain $(\mathbf{p}, r\mathbf{Y})$ if $(q, \lambda) \in \delta(p, a, Y)$.
6. For all $X \in \Gamma$ and $p \in \Delta(r)$, for some $r \in Q$, let $\delta'(\mathbf{p}, \lambda, Z_0 X r \mathbf{X})$ contain $(\mathbf{q}_f, \lambda)$.

Transitions from (1) and (2) cause A_2 to simulate A_1 step-by-step until the $(k + 1)$ st pushdown reversal done by A_1 appears. All elements described in (3), (4), (5), and (6) allow A_2 to start a backward simulation of A_1 on the reverse remaining input. To be more precise, the transitions in (3) start the backward simulation of A_2 by undoing the very last step of A_1 , i.e., by pushing $Z_0 r \mathbf{Z}_0$ onto the pushdown, reading symbol a , and continuing with state $\mathbf{q}$, whenever A_1 has used transition $(p, \lambda) \in \delta(q, a, Z_0)$, for some $p \in Q$, in its last computation step. Then in (4) push moves of A_1 are simulated as pop moves by A_2 , always assuming to have a boldface symbol on top of the pushdown. Moreover, transitions specified in (5) simulate pop moves of A_1 by push moves of A_2 . Here we have to consider two cases, namely starting a sub-computation which (a) comes back to the same pushdown height or (b) comes not back to the same pushdown height. In the latter case A_2 has to pop a compatible non-boldface symbol together with a boldface symbol in order to decrease the pushdown height. Finally, in (6) the termination of the computation is done, by checking that the pushdown contains a string of the form $Z_0 X r \mathbf{X}$ for some $X \in \Gamma$ and $r \in Q$, and has reached some state in $\Delta(r)$.

Now assume that $w \in N_{k+1}(A_1)$ such that $w = uva$ with

$$(q_0, uva, Z_0) \vdash_{A_1}^* (q_1, va, Z_0 X \gamma) \vdash_{A_1} (q_2, va, Z_0 \gamma^R X) \vdash_{A_1}^* (q_3, a, Z_0) \vdash_{A_1} (q_4, \lambda, \lambda),$$

where $u, v \in \Sigma^*$, $a \in \Sigma \cup \{\lambda\}$, $X \in \Gamma \cup \{\lambda\}$, $\gamma \in \Gamma^*$, $X = \lambda$ implies $\gamma = \lambda$, and the last pushdown reversal appears at $(q_1, va, Z_0 X \gamma) \vdash_{A_1} (q_2, va, Z_0 \gamma^R X)$. Thus, by our previous considerations we find the simulation

$$(q_0, uav^R, Z_0) \vdash_{A_2}^* (q_1, av^R, Z_0 X \gamma) \vdash_{A_2} (\mathbf{q}_3, v^R, Z_0 X \gamma Z_0 q_1 \mathbf{Z}_0) \\ \vdash_{A_2}^* (\mathbf{q}_2, \lambda, Z_0 X q_1 \mathbf{X}) \vdash_{A_2} (\mathbf{q}_f, \lambda, \lambda),$$

and therefore $uav^R = u(va)^R$ belongs to $T_k(A_2)$, since the number of reversals was decreased by one. By similar reasoning, if $u(va)^R \in T_k(A_2)$, then $uva \in N_{k+1}(A_1)$. Since state acceptance and acceptance by empty pushdown coincides for flip-pushdown automata, the claim follows. $\square$

An immediate consequence of Theorem 5 is the following corollary, which we state without proof.

Corollary 6. *If L is a unary language accepted by some flip-pushdown automaton with exactly k flips, for some $k \geq 0$, then L is a regular language.* $\square$

Another consequence of the flip-pushdown-input reversal theorem is, that we can separate the hierarchy of pushdown reversal language families for both deterministic and nondeterministic flip-pushdown automata.

Theorem 7. $\mathcal{L}(\text{DFPDA}_k) \subset \mathcal{L}(\text{DFPDA}_{k+1})$ and $\mathcal{L}(\text{NFPDA}_k) \subset \mathcal{L}(\text{NFPDA}_{k+1})$, for all $k \geq 0$, where $\mathcal{L}(\text{DFPDA}_k)$ denotes the family of languages accepted by deterministic flip-pushdown automata with exactly k pushdown reversals.

Proof. It suffices to prove that $\mathcal{L}(\text{DFPDA}_{k+1}) \setminus \mathcal{L}(\text{NFPDA}_k) \neq \emptyset$. To this end, we define, for $k \geq 1$, the language

$$L_k = \{ \#w_1\$w_1\#w_2\$w_2\# \dots \#w_k\$w_k\# \mid w_i \in \{a, b\}^* \text{ for } 1 \leq i \leq k \}.$$

Obviously, language L_{k+1} is accepted by a (deterministic) flip-pushdown automaton making exactly $k+1$ pushdown reversals. Hence $L_{k+1} \in \mathcal{L}(\text{DFPDA}_{k+1})$.

Next we prove that $L_{k+1} \notin \mathcal{L}(\text{NFPDA}_k)$. Assume to the contrary, that language L_{k+1} is accepted by some flip-pushdown automaton A with exactly k pushdown reversals. Then applying the flip-pushdown input-reversal Theorem 5 exactly k times, results in a context-free language L . Now the idea is to pump an appropriate word from the context-free language and to undo the flip-pushdown input-reversals, in order to obtain a word that must be in L_{k+1} . If the pumping is done such that no input reversal boundaries in the word are pumped, then the flip-pushdown input-reversals can be undone. Therefore, we need a generalization of Ogden's lemma, which is due to Bader and Moura [1] and incooperates excluded positions².

Let n be the constant in the generalization of Ogden's lemma for L and $z = (\#a^{n^{2k+1}}b^{n^{2k+1}}\$a^{n^{2k+1}}b^{n^{2k+1}})^{k+1}\#$ be in L_{k+1} . Consider the word z when transformed into an instance z' of the context-free language L . When applying Theorem 5 to a word wv it becomes wv^R , then we mark the last position of w and the first position of v^R as excluded. Hence, after k applications of Theorem 5 word z' in L contains at most $2k$ excluded positions e . Moreover, since only k flip-pushdown input-reversals are allowed, and $k+1$ blocks $\#a^{n^{2k+1}}b^{n^{2k+1}}\$a^{n^{2k+1}}b^{n^{2k+1}}\#$ exist, due to the pigeon-hole principle there must be at least one block, which was not cut and (its remaining input) reversed. We

² For any context-free language L , there exists a natural number n , such that for all words z in L , if d positions in z are "distinguished" and e positions are "excluded," with $d > n^{e+1}$, then there are words u, v, w, x , and y such that $z = uvwxy$ and (1) vx contains at least one distinguished position and no excluded positions, (2) if r is the number of distinguished positions and s is the number of excluded positions in vwx , then $r \leq n^{s+1}$, and (3) word $uv^iwx^i y$ is in L for all $i \geq 0$.

pick one of these intact blocks in z' and mark all its positions as distinguished. Thus, there are $d = 4 \cdot n^{2k+1} + 2$ distinguished positions in z' , with $d > n^{e+1}$.

Now assume that words u , v , w , x , and y satisfy the properties of the generalization of Ogden's lemma. First, we can easily see that if either v or x contains symbols $\$$ or $\#$, then we obtain a contradiction by considering word uv^2wx^2y , since every word in L (L_{k+1} , respectively) contains exactly $k + 1$ symbols $\$$ and exactly $k + 2$ symbols $\#$. Second, we know that because vx contains at least one distinguished position, word v or x lies completely within our chosen intact block $\#a^{n^{2k+1}}b^{n^{2k+1}}\$a^{n^{2k+1}}b^{n^{2k+1}}\#$ (excluding the symbols $\$$ and $\#$). Then we distinguish three cases:

1. Both words v and x are within the block under consideration. Then the number of excluded positions in $vw x$ equals zero, and hence $|vwx| \leq n$. Then we obtain, that the block under consideration loses its "copy" form in the word $\tilde{z}' = uv^2wx^2y$, i.e., the block we are looking at is not of the form $\#w\$w\#$, for some w , anymore.
2. Word v is within the block under consideration, but x is not. Then the number of excluded positions in $vw x$ is at most $2k$, and hence $|v| \leq n^{2k+1}$. Again, the block under consideration loses its form in the word $\tilde{z}' = uv^2wx^2y$.
3. Word v is not within the block under consideration, but x is. Then a similar reasoning as in the case above applies.

Since we know little about the context-free language L , we now transform our pumped string $\tilde{z}'$ back towards language L_{k+1} , according to Theorem 5. Now the advantage of the excluded positions comes into play. Since we have never pumped on excluded positions, the pushdown reversal move is still valid. Hence, word $\tilde{z}'$ leads us to a word $\tilde{z}$, where the original intact block considered so far is now not of the form $\#w\$w\#$, for some w anymore. Observe, that the application of Theorem 5 is done exactly in the reverse order as above. This means, that an input reversal appears only at excluded positions (or in-between two excluded ones). In particular, the block considered so far remains untouched during this process. Therefore, word $\tilde{z}$ is not a member of language L_{k+1} . This contradicts our assumption, and thus $L_{k+1} \notin \mathcal{L}(\text{NFPDA}_k)$. $\square$

4 Closure Properties of Flip-Pushdown Languages

In this section we consider closure properties of the family of flip-pushdown languages. For the below given theorem, we need the notion of a rational a -transducer, where we refer to Berstel [3]. Since the proof of the following theorem is an adaption from the context-free case, we omit the proof.

Theorem 8. *The language families $\mathcal{L}(\text{NFPDA}_k)$, for $k \geq 0$, and $\mathcal{L}(\text{NFPDA}_{\text{fin}})$ are closed under rational a -transduction. Hence, the families under consideration are full TRIO's, i.e., closed under intersection with regular languages, arbitrary homomorphism, and inverse homomorphism.* $\square$

Next we consider the boolean operations union, intersection, and complement as well as concatenation and Kleene star

Theorem 9. *The language families $\mathcal{L}(\text{NFPDA}_k)$, for $k \geq 0$, and $\mathcal{L}(\text{NFPDA}_{fin})$ are both closed under union, but both families are not closed under intersection and complementation. Moreover, $\mathcal{L}(\text{NFPDA}_k)$ is not closed under concatenation, while $\mathcal{L}(\text{NFPDA}_{fin})$ is closed, and both language families are not closed under Kleene star.*

Proof. The closures are immediate. The non-closure results are seen as follows: In case of intersection it suffices to show that the language $L = \{a^n b^n c^n \mid n \geq 1\}$, which is the intersection of two context-free languages is not a flip-pushdown language. Assume to the contrary, that language L belongs to $\mathcal{L}(\text{NFPDA}_k)$ for some k . Then we k times apply the flip-pushdown input-reversal Theorem 5 to L obtaining a context-free language. Since we do the input reversal from right-to-left, the block of c 's remains intact in all words. Hence a word w in the context-free language reads as $w = a^{n_1} b^{m_1} c^n b^{m_2} a^{n_2}$, where $n_1 + n_2 = m_1 + m_2 = n$. Then it is an easy exercise to show that this language cannot be context-free using Ogden's lemma. This contradicts our assumption, and thus, language L doesn't belong to $\mathcal{L}(\text{NFPDA}_k)$, for any $k \geq 0$. This shows the non-closure under intersection and complementation due to DeMorgan's law.

For concatenation and Kleene star we argue as follows: Let $k \geq 1$. Obviously, language L_{k+1} , defined in the proof of Theorem 7, satisfies $L_{k+1} = L_k \cdot \{w\$w\# \mid w \in \{a, b\}^*\}$, where both languages on the right-hand side of the equation belong to the family $\mathcal{L}(\text{NFPDA}_k)$. Since by Theorem 7 language $L_{k+1} \in \mathcal{L}(\text{NFPDA}_{k+1}) \setminus \mathcal{L}(\text{NFPDA}_k)$, the non-closure of the language family $\mathcal{L}(\text{NFPDA}_k)$, for $k \geq 1$, immediately follows. Moreover, since $L_{k+1} = \# \cdot \{w\$w\# \mid w \in \{a, b\}^*\}^{k+1}$, the language $L_\infty = \bigcup_{k=0}^\infty L_k$ equals $\# \cdot \{w\$w\# \mid w \in \{a, b\}^*\}^*$. Thus, if L_∞ would belong to some family $\mathcal{L}(\text{NFPDA}_k)$, for some $k \geq 1$, then language $L_{k+1} = L_\infty \cap \#(\{a, b\}^*\$ \{a, b\}^*\#)^{k+1}$ is a member of $\mathcal{L}(\text{NFPDA}_k)$, which contradicts the proof of Theorem 7 due to the closure of this language family under intersection with regular sets and concatenation with a regular set to the left—the latter closure property follows from the closure under TRIO operations. Hence, $\mathcal{L}(\text{NFPDA}_k)$, for $k \geq 1$, and $\mathcal{L}(\text{NFPDA}_{fin})$ are both not closed under Kleene star. $\square$

Finally, in Table 1 we summarize our results on closure and non-closure properties for flip-pushdown language families. Observe, that $\mathcal{L}(\text{CFL}) = \mathcal{L}(\text{NFPDA}_0)$ is the lowest level in the flip-pushdown hierarchy, while unbounded pushdown reversals are the other end, i.e., $\mathcal{L}(\text{RE}) = \mathcal{L}(\text{NFPDA})$.

5 Computational Complexity of Flip-Pushdown Languages

We consider some computational complexity problems of flip-pushdown languages in more detail. Firstly, we improve the upper bound on the $\mathcal{L}(\text{NFPDA}_k)$ language families given in Theorem 4.

Table 1. Closure properties of flip-pushdown languages.

Operation	Language family			
	$\mathcal{L}(\text{CFL})$	$\mathcal{L}(\text{NFPDA}_k)$ with $k \geq 1$	$\mathcal{L}(\text{NFPDA}_{fin})$	$\mathcal{L}(\text{NFPDA})$
Union	Yes	Yes	Yes	Yes
Intersection	No	No	No	Yes
Complementation	No	No	No	No
Homomorphism	Yes	Yes	Yes	Yes
Inverse homomorphism	Yes	Yes	Yes	Yes
Intersection with regular sets	Yes	Yes	Yes	Yes
Concatenation	Yes	No	Yes	Yes
Kleene star	Yes	No	No	Yes
Quotient with regular sets	Yes	Yes	Yes	Yes

Theorem 10. $\mathcal{L}(\text{CFL}) \subset \mathcal{L}(\text{NFPDA}_k) \subset \mathcal{L}(\text{CS})$ for $k \geq 1$.

Proof. The first inclusion is straight forward and its strictness follows from Example 2. The containment of $\mathcal{L}(\text{NFPDA}_k)$ in $\mathcal{L}(\text{CS})$ is seen as follows: Let A be a flip-pushdown automaton making exactly k pushdown reversals. According to Theorem 5 we construct a context-free language L . In order to check membership in $T_k(A)$ a linear bounded automaton guesses a length k sequence of flip-pushdown input-reversals and applies it to the input w to transform it into an instance of the context-free language L . Since context-free membership can be decided by a linear bounded Turing machine, the second inclusion follows. Strictness is seen by Corollary 6, because, e.g., language $\{a^p \mid p \text{ is prim}\}$ is a context-sensitive language, which is not regular. $\square$

Now the question arises, how complicated is it to decide membership for flip-pushdown languages.

Theorem 11. *The following problems are complete w.r.t. deterministic logspace many-one reductions: (1) The fixed membership problem for k -flip-pushdown languages is $\text{LOG}(\text{CFL})$ -complete and (2) the general membership problem for k -flip-pushdown automata languages is P-complete.*

Proof. In both cases, the hardness results immediately follow from the inclusion $\mathcal{L}(\text{CFL}) \subseteq \mathcal{L}(\text{NFPDA}_k)$ for any $k \geq 0$, and the $\text{LOG}(\text{CFL})$ -completeness of fixed membership for context-free languages [14] and the P-completeness for general membership [11]. For the upper bounds we argue as in the proof of Theorem 10. The main difference in the proof is, that we can not guess a length k sequence of flip-pushdown input-reversals. Nevertheless, a deterministic logspace machine can enumerate all possible outcomes of length k sequences of flip-pushdown

input-reversals separated by \$ symbols. This suffices to prove the upper bounds—the details are left to the reader. $\square$

The above given theorem can be restated in terms of auxiliary flip-pushdown automata. It shows that auxiliary flip-pushdown automata with exactly k pushdown reversal and a logarithmically space bounded work-tape capture P , and when additionally their time is polynomially bounded the class $\text{LOG(CFL)} \subseteq P$.

6 Conclusions

We have investigated flip-pushdown automata, which were recently introduced by Sarkar [13]. The major contribution of this paper is a positive answer to Sarkar's conjecture on the strictness of the flip-pushdown hierarchy w.r.t. the number of pushdown reversals for both deterministic and nondeterministic flip-pushdown automata. Moreover, we also considered closure and non-closure properties, as well as some computational complexity problems of these language families. In most cases, flip-pushdown languages share similar properties than context-free languages. In Figure 1 the inclusion relations among the classes considered and their computational complexities (completeness) are depicted.

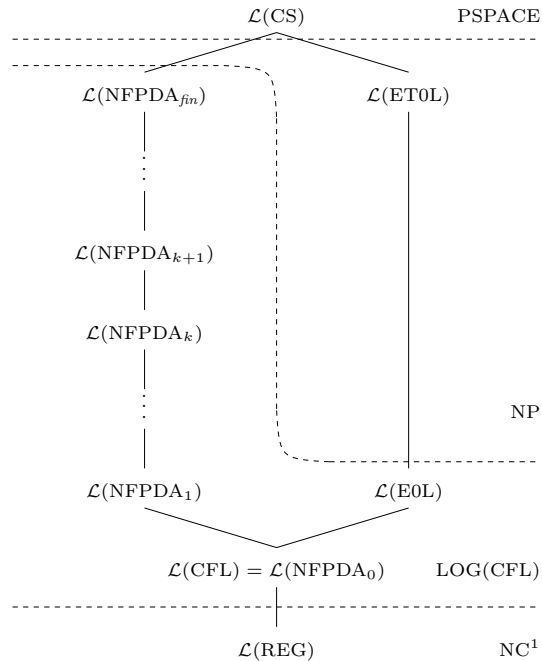


Fig. 1. Inclusion structure.

The results presented imply that flip-pushdown languages accepted by flip-pushdown automata with a constant number of pushdown reversals are almost *mildly context-sensitive*, i.e., each language is semi-linear, each language has a deterministic polynomial time solvable membership problem, and the language family contains the following non-context-free languages: Multiple agreements $L_1 = \{a^n b^n c^n \mid n \geq 0\}$, crossed agreements $L_2 = \{a^n b^m c^n d^m \mid n, m \geq 0\}$, and duplication $L_3 = \{ww \mid w \in \{a, b\}^*\}$. Except for the non-containment of L_1 , all properties of mildly context-sensitive languages are fulfilled.

Nevertheless, several questions for flip-pushdown languages remain unanswered. We mention two of these questions: (1) How does the deterministic and nondeterministic flip-pushdown language hierarchies w.r.t. the number of pushdown reversals relate to each other? (2) What is the relationship between these language families and other well known formal language classes? Especially, the latter question is of some interest, because we were not even able to clear the relationship between the family of flip-pushdown languages and some Lindenmayer families like, e.g. E0L or ET0L languages—see Rozenberg and Salomaa [12]. We conjecture incomparability, but have no proof yet.

References

1. Ch. Bader and A. Moura. A generalization of Ogden's lemma. *Journal of the ACM*, 29(2):404–407, 1982.
2. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.
3. J. Berstel. *Transductions and Context-Free Languages*, volume 38 of *Leitfäden der angewandten Mathematik und Mechanik LAMM*. Teubner, 1979.
4. N. Chomsky. *Handbook of Mathematic Psychology*, volume 2, chapter Formal Properties of Grammars, pages 323–418. Wiley & Sons, New York, 1962.
5. S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, January 1971.
6. R. J. Evey. *The Theory and Applications of Pushdown Store Machines*. Ph.D thesis, Harvard University, Massachusetts, May 1963.
7. S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland, Amsterdam, 1975.
8. S. Ginsburg, S. A. Greibach, and M. A. Harrison. One-way stack automata. *Journal of the ACM*, 14(2):389–418, April 1967.
9. S. Ginsburg and E. H. Spanier. Finite-turn pushdown automata. *SIAM Journal on Computing*, 4(3):429–453, 1966.
10. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
11. N. D. Jones and W. T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3:105–117, 1977.
12. G. Rozenberg and A. Salomaa. *The Mathematical Theory of L Systems*, volume 90 of *Pure and Applied Mathematics*. Academic Press, 1980.
13. P. Sarkar. Pushdown automaton with the ability to flip its stack. Report TR01-081, Electronic Colloquium on Computational Complexity (ECCC), November 2001.
14. I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, July 1978.

Convergence Time to Nash Equilibria

Eyal Even-Dar*, Alex Kesselman, and Yishay Mansour**

School of Computer Science, Tel-Aviv University,
{evend, alex, mansour}@cs.tau.ac.il.

Abstract. We study the number of steps required to reach a pure Nash Equilibrium in a load balancing scenario where each job behaves selfishly and attempts to migrate to a machine which will minimize its cost. We consider a variety of load balancing models, including identical, restricted, related and unrelated machines. Our results have a crucial dependence on the weights assigned to jobs. We consider arbitrary weights, integer weights, K distinct weights and identical (unit) weights. We look both at an arbitrary schedule (where the only restriction is that a job migrates to a machine which lowers its cost) and specific efficient schedulers (such as allowing the largest weight job to move first).

1 Introduction

As the users population accessing Internet services grows in size and dispersion, it is necessary to improve performance and scalability by deploying multiple, distributed server sites. Distributing services has the benefit reducing access latency, and improving service scalability by distributing the load among several sites. One important issue in such a scenario is how the user chooses the appropriate server. Similar problem occurs in the context of routing where the user has to select one of a few parallel links. For instance, many enterprise networks are connected to multiple Internet service providers (ISPs) for redundant connectivity, and backbones often have multiple parallel trunks.

Users are likely to behave “selfishly” in such cases, that is each user makes decisions so as to optimize its own performance, without coordination with the other users. Basically, each user would like to either maximize the resources allocated to it or, alternatively, minimize its cost. Load balancing and other resource allocation problems are prime candidates for such a “selfish” behavior.

A natural framework to analyze this class of problems is that of non-cooperative games, and an appropriate solution concept is that of Nash Equilibrium [22]. A strategy for the users is at a Nash Equilibrium if no user can gain by unilaterally deviating from its own policy. In this paper we focus on the load balancing problem. An interesting class of non-cooperative games, which is related to load balancing, is congestion games [24] and its equivalent model exact potential games [21].

* Supported by the Deutsch Institute

** Supported in part by a grant from the Israel Science Foundation.

Traditionally in Computer Science research has been focused on finding a global optimum. With the emerging interest in computational issues in game theory, the *coordination ratio* [17] has received considerable attention [2,7,8,13,17,25]. The coordination ratio is the ratio between the worst possible Nash equilibrium (the one with maximum social cost) and the social optimum (an optimal solution with the minimal social cost). One motivation is to show that the gap between a Nash Equilibrium and the optimal solution is in some cases not significant, thus good performance can be achieved even without a centralized control.

In this work we are concerned with the time it takes for the system to converge to a Nash equilibrium, rather than the quality of the resulting allocation. The question of convergence to a Nash equilibrium has received significant attention in the Game Theory literature (see [12]). Our approach is different from most of that line of research in a few crucial aspects. First, we are interested in quantitative bounds, rather than showing a convergence in the limit. Second, we consider games with many players (jobs) and actions (machines) and study their asymptotic behavior. Third, We limit ourselves in this work to a subclass of games that arise from load balancing, for which there always exists a pure Nash equilibrium, and thus we can allow ourselves to study only deterministic policies.

Our Model. This paper deals with load balancing (see, [3]). Jobs (players) are allowed to select a machine to minimize their own cost. The cost that a job observes from the use of a machine is determined by the load on that machine. We consider weighted load functions, where each job has a corresponding weight and the load on a machine is sum of the weights of the jobs running on it. Until a Nash Equilibrium is reached, at least one job wishes to change its machine. In our model, similarly to the Elementary Stepwise System (see [23]), at every time step only one job is allowed to move, and a centralized controller decides which job would move in the current time step. By strategy we mean the algorithm used by the centralized controller for selecting which of the competing jobs would move. Due to the selfish nature of jobs, we assume that when a job migrates its observed load is strictly reduced, which we refer to as an *improvement* policy. We also consider the well known case of *best reply* policy, where each job moves to a machine in which its observed load is minimal.

Our Results. We assume that there are n jobs and m machines. We assume that K is the number of different weights, W is the total weight of all the jobs and w_{max} is the maximum weight assigned to a job.

For the general case of unrelated machines we show that the system always converges to a Nash equilibrium. This is done by introducing an order between the different configurations and showing that when a job migrates we move to a “lower” configuration in the order. Bounding the number of configurations by $\min\{[O(\frac{n}{Km} + 1)]^{Km}, m^n\}$ derives a general bound. Using a potential base argument we derive a bound of $O(4^W)$ for integer weights, where W is the worse case sum of the weights of the jobs. For the specific strategy that first

selects jobs from the most loaded machine we can show an improved bound of $O(mW + 4^{W/m+w_{max}})$.

In the simple case of identical machines and unrestricted assignments we show that if one moves the minimum weight job, the convergence may take an exponential number of steps. Specifically, the number of steps is at least,

$$\frac{(\frac{n}{K})^K}{2(K!)} = \Omega\left(\left(\frac{n}{K^2}\right)^K\right)$$

for $K = m - 1$. In contrast, we show that if one moves the maximum weight job, and the jobs follow the best reply policy, a Nash Equilibrium is reached in at most n steps. This shows the importance of selecting of the “right” scheduling strategy. We also show that selecting the minimal weight job is “almost” the worst case for identical machines, by demonstrating that any strategy converges in $(\frac{n}{K} + 1)^K$ time steps. We also show that any strategy converges in $O(W + n)$ steps for integer weights. For the Random and FIFO strategies we show that they converge in $O(n^2)$ steps.

For restricted assignment and related machines we bound by $O((W^2 S_{max}^2)/\epsilon)$ the convergence time to ϵ -Nash, where no job can benefit more than ϵ from unilaterally migrating to another machine. Using the strategy that schedules first jobs from the most loaded machine we can derive an improved convergence bound. Note that in our setting there always exists an ϵ_{min} such that for any $\epsilon < \epsilon_{min}$ we have that any ϵ -Nash equilibrium is a Nash equilibrium. For example, in the case of identical machine with integer weights $\epsilon_{min} = 1$.

For K integer weights, we are able to derive an interesting connection between W and K , for the case of identical and related machines. We show that for any set V of K integer weights there is an equivalent set V' of K integer weights such that the maximum weight in V' is at most $O(K(cS_{max}n)^{4K})$ for some positive constant c . The equivalence guarantees that the relative cost of different machines is maintained in all configurations. (In addition, we never need to compute V' , but rather it is only used in the convergence proofs.) The equivalence implies that $W = O(Kn(cS_{max}n)^{4K})$. Thus, all bounds that depend on W can depend on $O(Kn(cS_{max}n)^{4K})$.

Related Work. Milchtaich [20] describes a class of non-cooperative games, which is related to load balancing. (In order to make the relations between the models clearer we use the load balancing terminology to describe his work.) The jobs (players) share a common set of machines (strategies). The cost of a job when selecting a particular machine depends only on the total number of jobs mapped to the machine (implicitly, all the weights are identical). However, each job has a different cost function for each machine, this is in contrast to the load balancing model where the cost of all the jobs that map to the same machine is identical. It is shown that these games always possess at least one pure (deterministic) Nash Equilibrium and there exists a best reply improvement strategy that converges in polynomial time. However, for the weighted version of these games there are cases where a pure Nash Equilibrium does not exist. In contrast, we show that any improvement policy converges to a pure Nash Equilibrium in the load balancing setting.

Our model is related to the makespan minimization problem since job moves can be viewed as a sequence of local improvements. The analysis of the approximation ratio of the local optima obtained by iterative improvement appears in [5,6,26]. The approximation ratio of a jump (one job moves at a time) iterative improvement has been studied in [10]. In [6] it has been shown that for two identical machines this heuristic requires at most n^2 iterations, which immediately translates to an n^2 upper bound for two identical machines with general weight setting in our model. In [26] they observe that the improvement strategy that moves the maximum weight job converges in n steps.

Some interesting related learning models are stochastic fictitious play [12], graphical games [19], and large population games [14]. Uniqueness of Nash Equilibria in communication networks with selfish users has been investigated in [23]. An analysis of the convergence to a Nash Equilibrium in the limit appears in [1, 4].

Paper organization: The rest of the paper is organized as follows. In Section 2 we present our model. The analysis of unrelated, related and identical machines appears in Section 3, Section 4 and Section 5, respectively. We conclude with Section 6. Due to space limitations some proofs are omitted and can be found in [9].

2 Model Description

In our load balancing scenario there are m parallel machines and n independent jobs. Each job selects exactly one machine.

Machines Model. We consider identical, related and unrelated machines. We denote by S_i the speed of M_i . Let S_{min} and S_{max} denote the minimal and maximal speed, respectively. WLOG, we assume that $S_{min} = 1$. For identical and unrelated machines we have $S_i = 1$ for $1 \leq i \leq m$.

Jobs Model. We consider both restricted and unrestricted assignments of jobs to machines. In the unrestricted assignment case each job can select any machine while in the restricted assignment case each job J can only select a machine from a pre-defined subset of machines denoted by $R(J)$.

For a job J , we denote by $w_i(J)$ the weight of J on machine M_i (where $i \in R(J)$) and by $M(J, t)$ the index of the machine on which J runs at time t . When considering identical machines, each job J has a weight $w(J) = w_i(J)$. We denote by W the maximal total weight of the jobs, that is $W = \sum_{i=1}^n \max_{j \in R(J_i)} \{w_j(J_i)\}$ and by $w_{max} = \max_i \max_{j \in R(J_i)} \{w_j(J_i)\}$ the maximum weight of a job.

We consider the following weight settings: *General weight setting* – the weights may be arbitrary real numbers. *Discrete weight setting* – there are K different integer weights $w_1 \leq \dots \leq w_K = w_{max}$. *Integer weight setting* – the weights are integers.

Load Model. We denote by $B_i(t)$ the set of jobs on machine M_i at time t . The load of a machine M_i at time t is the sum of the weights of the jobs that chose M_i , that is $L_i(t) = \sum_{J \in B_i(t)} w(J)$, and its normalized load is $T_i(t) = L_i(t)/S_i$.

We also define $L_{max}(t) = \max_i \{L_i(t)\}$ and $T_{max}(t) = \max_i \{T_i(t)\}$. The cost of job J at time t is the normalized load on the machine $M(J, t)$, i.e., $T_{M(J, t)}(t)$. We define the *marginal* load with respect to a job to be the load in the system when this job is removed.

System Model. The *system state* consists of the current assignment of the jobs to the machines. The system starts in an arbitrary state and each job has a full knowledge of the system state. A job wishes to migrate to another machine, if and only if, after the migration its cost is strictly reduced. Before migrating between machines, a job needs to receive a grant from the centralized controller. The controller has no influence on the selection of the target machine by a migrating job, it just gives the job a permission to migrate. The above is known in the literature as an Elementary Stepwise System (ESWS) (see [4,23]). Essentially, the controller serves as a critical section control. The execution is modeled as a sequence of steps and in each step one job changes its machine. Notice that if all jobs are allowed to move simultaneously, the system might oscillate and never ever reach a Nash Equilibrium.

Let $A(t)$ be the set of jobs that may decrease the experienced load at time t by migrating to another machine. When a migrating job selects a machine which minimizes its cost (after the migration), we call to this *best-reply* policy. Otherwise, we call to this *improvement* policy.

The system is said to reach a *pure* (or deterministic) Nash Equilibrium if no job can benefit from unilaterally migrating to another machine. The system is said to reach an ϵ -Nash Equilibrium if no job can benefit more than ϵ from unilaterally migrating to another machine. We study the number of time steps it takes to reach a Nash Equilibrium (or ϵ -Nash equilibrium) for different strategies of ESWS job scheduling.

Scheduling Strategies: We define a few natural strategies for the centralized controller. The input at time t is always a set of jobs $A(t)$ and the output is a job $J \in A(t)$ which would migrate at time t . (For simplicity we assume each job has a unique weight, extension for unrelated machines is possible.) The specific strategies that we consider are:

Random: Selects $J \in A(t)$ with probability $1/|A(t)|$.

Max Weight Job: Selects $J \in A(t)$ such that $w(J) = \max_{J' \in A(t)} \{w(J')\}$.

Min Weight Job: Selects $J \in A(t)$ such that $w(J) = \min_{J' \in A(t)} \{w(J')\}$.

FIFO: Let $E(J)$ be the smallest time t' such that $J \in A(t'')$ for every $t'' \in [t', t]$.

FIFO selects $J \in A(t)$ such that $E(J) = \min_{J' \in A(t)} \{E(J')\}$.

Max Load Machine: Selects $J \in A(t)$ such that $T_{M(J, t)}$ is maximal.

3 Unrelated Machines

In this section we consider the unrelated machines case with the restricted assignment. To show the convergence we define a sorted lexicographic order of the vectors describing the machine loads as follows. Consider the sorted vector of the machine loads. One vector is called “larger” than another if its first (after the common beginning of the two vectors) load component is larger than the

corresponding load component of the second vector. Formally, given two load vectors ℓ_1 and ℓ_2 , let $s_1 = \text{sort}(\ell_1)$ and $s_2 = \text{sort}(\ell_2)$ where $\text{sort}()$ returns a vector in the sorted order. We define $\ell_1 \succ \ell_2$ if $s_1 \succ s_2$ using a lexicographic ordering, i.e., $s_1[i] = s_2[i]$ for $i < k$ and $s_1[k] > s_2[k]$.

We demonstrate that the sorted lexicographic order of the load vector always decreases when a job migrates. To observe this one should note that only two machine are influenced by the migration of the job J at time t , $M_i = M(J, t)$, where job J was before the migration and $M_j = M(J, t + 1)$, the machine J migrated to. Furthermore $L_i(t) > L_j(t + 1)$, otherwise job J would not have migrated. Also note that $L_i(t) > L_i(t + 1)$ since job J has left M_i . Let $L = \max\{L_i(t + 1), L_j(t + 1)\}$. Since $L < L_i(t)$ one can show that the new machine loads vector is smaller in the sorted lexicographic order than the old machine loads vector. This is summarized in the following claim.

Claim. The sorted lexicographic order of the machine loads vector decreases when a job migrates.

The above argument shows that any improvement policy converges to a Nash equilibrium, and gives us an upper bound on the convergence time equal to the number different sorted machine loads vectors (which is trivially bounded by the number of different system configurations).

General Weights. In the general case, the number of different system configurations is at most m^n , which derives the following corollary.

Corollary 1. *For any ESWS strategy with an improvement policy, the system of multiple unrelated machines with restricted assignment reaches a Nash Equilibrium in at most m^n steps.*

Discrete Weights. For the discrete weight setting, the number of different weights is K . Let n_i be the number of jobs with weight w_i . The number of different configurations of jobs with weight w_i is bounded by $\binom{m+n_i}{m}$. Multiplying the number of configurations for the different weights bounds the number of different system configurations. Since, by definition, $\sum_{i=1}^K n_i = n$, we can derive the following.

Corollary 2. *For any ESWS strategy with an improvement policy, the system of multiple unrelated machines with restricted assignment under the discrete weight setting reaches a Nash Equilibrium in at most*

$$\prod_{i=1}^K \binom{m+n_i}{m} \leq (c \frac{n}{Km} + c)^{Km},$$

steps for some constant $c > 0$.

Integer Weights. To bound the convergence time for the integer weight setting, we introduce a potential function and demonstrate that it decreases when a job migrates. We define the potential of the system at time t , as $P(t) = \sum_{i=1}^m 4^{L_i(t)}$. After job J migrates from M_i to M_j then we have that $L_i(t) - 1 \geq$

$L_j(t+1)$, since J migrated. Also, since we have integer weights, $L_i(t+1) \leq L_i(t) - 1$. Therefore, the reduction in the potential is at least,

$$P(t) - P(t+1) = 4^{L_i(t)} + 4^{L_j(t)} - [4^{L_i(t+1)} + 4^{L_j(t+1)}] \geq 4^{L_i(t)}/2 \geq 2. \quad (1)$$

Since in the initial configuration we have that $P(0) \leq 4^W$ we derive the following theorem.

Theorem 1. *For any ESWS strategy with an improvement policy, the system of multiple machines under the integer weight setting reaches a Nash Equilibrium in $4^W/2$ steps.*

Next we show that this bound can be reduced to $O(mW + m4^{W/m+w_{max}})$ when using the Max Load Machine strategy.

Theorem 2. *For Max Load Machine strategy with an improvement policy, the system of multiple machines under the integer weight setting reaches a Nash Equilibrium in at most $4mW + m4^{W/m+w_{max}}/2$ steps.*

Proof. We divide the schedule into two phases with respect to the maximum load among the machines. The first phase continues until $L_{max}(t) \leq W/m + w_{max}$, and then the second phase starts. At the start of the second phase, at time T , the potential is at most $m4^{L_{max}(T)} \leq m4^{W/m+w_{max}}$. By (1), at every step the potential drops by at least two, therefore the length of the second phase is bounded by $m4^{W/m+w_{max}}/2$. Thus, it remains to bound the length of the first phase, namely T . At any time $t < T$ we have $L_{max}(t) > W/m + w_{max}$, which implies that $L_{min}(t) \leq W/m$. Therefore every job in the maximum loaded machine can benefit by migrating to the least loaded machine. The Max Load Machine strategy will choose one of those jobs. By (1), the decrease in the potential is at least $4^{L_{max}(t)}/2 \geq P(t)/2m$. Therefore, after T steps we have $P(T) \leq P(0)(1 - 1/2m)^T$. Since $P(0) \leq 4^W$ and $P(T) \geq 1$, it follows that $T \leq 4mW$, which establishes the theorem. $\square$

Two Weights. It is worth to note that for the special case of two different weights there exists an efficient ESWS strategy the converges in linear time.

4 Related Machines

In this section we consider the related machines. We first consider restricted assignments and assume that all jobs follow an *improvement* policy. We define the potential of the system as follows:

$$P(t) = \sum_{i=1}^m \frac{(L_i(t))^2}{S_i} + \sum_{j=1}^n \frac{w_j^2}{S_{M(j,t)}} = \sum_{i=1}^m S_i (T_i(t))^2 + \sum_{j=1}^n \frac{w_j^2}{S_{M(j,t)}}$$

The following lemma shows that the potential drops after each improvement step.

Lemma 1. *When a job of size w migrates from machine i to machine j at time t then $P(t+1) - P(t) = 2w(T_j(t+1) - T_i(t)) < 0$.*

We now like to bound the drop in the potential in each step. Clearly, if we are interested in ϵ -Nash equilibrium, then the drop is at least $2w\epsilon > \epsilon$. Considering a Nash equilibrium, for integer weights and speeds the drop is at least $(S_{max})^{-2}$. Since the initial potential is bounded by W^2 , we can derive the following Theorem.

Theorem 3. *For any ESWS strategy with an improvement policy, the system of multiple related machines with restricted assignment reaches an ϵ -Nash Equilibrium in at most $O(\frac{W^2}{\epsilon})$ steps, and reaches a Nash Equilibrium, assuming both integer weights and speeds, in at most $O(W^2 S_{max}^2)$ steps.*

For unrestricted assignment, by forcing to move the job from the most loaded machine we can improve the bound as follows.

Theorem 4. *Max Load Machine strategy with best reply policy reaches an ϵ -Nash Equilibrium in at most*

$$O(W \sqrt{m S_{max} + \frac{nw_{max}^2}{\epsilon}})$$

steps.

Discrete Weights. We show that for any K integer weight there is an equivalent model in which w_{max} is bounded by $O(K(S_{max}n)^{4K})$, and therefore $W = O(Kn(S_{max}n)^{4K})$. This allows us to translate the results using W to the discrete weight model by replacing W by $O(Kn(S_{max}n)^{4K})$. (We do not need to calculate the equivalent weights, since they are only used for the convergence time analysis.) We first define what we mean by an equivalent set of weights.

Definition 1. *Two discrete set of weights $w_1, \dots, w_K$ and $\alpha_1, \dots, \alpha_K$ are equivalent if for any two assignments, $n_1, \dots, n_K$ and $\ell_1, \dots, \ell_K$ we have $\sum_{i=1}^K n_i w_i > \sum_{i=1}^K \ell_i w_i$ if and only if $\sum_{i=1}^K n_i \alpha_i > \sum_{i=1}^K \ell_i \alpha_i$, and $\sum_{i=1}^K n_i w_i = \sum_{i=1}^K \ell_i w_i$ if and only if $\sum_{i=1}^K n_i \alpha_i = \sum_{i=1}^K \ell_i \alpha_i$. (We require that both $\sum_{i=1}^K n_i \leq n$ and $\sum_{i=1}^K \ell_i \leq n$.)*

Intuitively, the above definition says that as long as we use only comparisons, we can replace $w_1, \dots, w_K$ by $\alpha_1, \dots, \alpha_K$. Most important for us is that we can use in the potential the α 's rather than the w 's. From the definition of an equivalent set of weights we can derive the following. Any strategy based on comparisons of job weights and machine loads and an improvement policy based on comparisons of machine loads (e.g. best reply) would produce the same sequence of job migrations starting from any initial configuration.

The following theorem, which is proven using standard linear integer programming techniques, bounds the size of the equivalent weights.

Theorem 5. *For any discrete set of weights $w_1, \dots, w_K$ there exist an equivalent set of weights $\alpha_1, \dots, \alpha_K$ such that $\alpha_K \leq K(cS_{max}n)^{4K}$ for some constant $c > 0$.*

Unit Weight Jobs. We show that for unit weight jobs, there exists a strategy that converges in mn steps. The unit weight jobs is a special case of [20] with a symmetric cost function, where was derived an upper bound of $O(mn^2)$ on the convergence time of a specific strategy. We follow the proof of [20] and obtain a better bound in our model.

Theorem 6. *There exists an ESWS strategy with an improvement policy such that the system of multiple related machines with restricted assignment reaches a Nash Equilibrium in at most mn steps in the case of unit weight jobs.*

The next theorem presents a lower bound of $\Omega(mn)$ on the convergence time of some ESWS strategy (different from that of Theorem 6).

Theorem 7. *There exists an ESWS strategy with an improvement policy such that for the system of multiple related machines with unrestricted assignment, there exists a system configuration that requires at least $\Omega(mn)$ steps to reach a Nash Equilibrium in the case of unit weight jobs.*

5 Identical Machines

In this section we will show improved upper bounds that apply to identical machines with unrestricted assignment. We also show a lower bound for K weights which is exponential in K . The lower bound is presented for the Min Weight Job policy. Clearly, this lower bound also implies a lower bound in all the other models. First we derive some general properties. The next observation states the minimal load cannot decrease.

Observation 1. At every time step the minimal load among the machines either remains the same or increases.

Now we show that when a job moves to a new machine, this machine still remains a minimal marginal load machine for all jobs at that machine which have greater weight.

Observation 2. If job J has migrated to its best response machine M_i at time t then M_i is a minimal marginal load machine with regard to any job $J' \in B_i(t)$ such that $w(J') \geq w(J)$.

Next we show that once a job has migrated to a new machine, it will not leave it unless a larger job arrives.

Claim. Suppose that job J has migrated to machine M at time t . If $J \in A(t')$ for $t' > t$ then another job J' such that $w(J') > w(J)$ switched to M at time t'' , and $t < t'' \leq t'$.

Next we present an upper bound on the convergence time of Max Weight Job strategy. (A similar claim (without proof) appears in [26].)

Theorem 8. *The Max Weight Job strategy with best response policy, for the system of multiple identical machines with unrestricted assignment reaches a Nash Equilibrium in at most n steps.*

Proof. By Claim 5, once the job has migrated to a new machine, it will not leave it unless a larger job arrives. But under Max Weight Job strategy only smaller jobs can arrive in the subsequent time steps, so each job stabilizes after the first migration, and the theorem follows. $\square$

Now we present a lower bound for the Min Weight Job strategy.

Theorem 9. *For the Min Weight Job strategy with best response policy, for the system of multiple identical machines with unrestricted assignment, there exists a system configuration that requires at least $(\frac{n}{K})^K / (2(K!))$ steps to reach a Nash Equilibrium, where $K = m - 1$.*

We also present a lower bound of $n^2/4$ on the convergence time of Min Weight Job and FIFO strategies for the case of two machines.

Theorem 10. *For the Min Weight Job and FIFO strategies with best response policy, for the system of two identical machines with unrestricted assignment, there exists a system configuration that requires at least $n^2/4$ steps to reach a Nash Equilibrium.*

Proof. Consider the following scenario. There are $n/2$ classes of jobs $C_1, \dots, C_{n/2}$ and each class contains exactly 2 jobs and has weight $w_i = 3^{i-1}$. Notice that a job in C_i with weight $w_i = 3^{i-1}$ has weight equal to the total weight of all the jobs in the first $i - 1$ classes plus 1.

Initially, all jobs are located at the same machine. We divide the schedule into phases. Let C_j^i we denote all jobs from classes $C_j, \dots, C_i$. A k -phase is defined as follows. Initially, all jobs from classes C_1^k are located at one machine. During the phase these jobs, except one job from C_k , migrate to the other machine. Thus, the duration of a k -phase is $2k - 1$. It is easy to see that the schedule consists of the phases $n/2, \dots, 1$ for Min Weight Job strategy. One can observe that FIFO can generate the same schedule, if ties are broken using minimal weight. $\square$

The following theorem shows a tight upper bound of $\Theta(n^2)$ on the convergence time of FIFO strategy.

Theorem 11. *For FIFO strategy with best response policy, the system of multiple identical machines with unrestricted assignment reaches a Nash Equilibrium in at most $n(n + 1)/2$ steps.*

Similarly to FIFO, we bound the expected convergence time of Random strategy by $O(n^2)$.

Theorem 12. *For Random strategy with best response policy, the system of multiple identical machines with unrestricted assignment reaches a Nash Equilibrium in expected time of at most $n(n + 1)/2$ steps.*

Discrete Weights. For the discrete weight case, we demonstrate an upper bound of $O((n/K+1)^K)$ on the convergence time of any ESWS strategy, showing that the bound of Theorem 9 for the Min Weight Job is not far from the worst convergence time.

Theorem 13. *For any ESWS strategy with best response policy, the system of multiple identical machines with unrestricted assignment under the discrete weight setting reaches a Nash Equilibrium in $O((n/K+1)^K)$ steps.*

Integer Weights. For the integer weight case, we show that the convergence time of any ESWS strategy is proportional to the sum of weights.

Theorem 14. *For any ESWS strategy with best response policy, the system of multiple identical machines with unrestricted assignment under the integer weight setting reaches a Nash Equilibrium in $W+n$ steps.*

Unit Weight Jobs. For the unit weight jobs, we present a lower bound on the convergence time of a specific strategy.

Theorem 15. *There exists an ESWS strategy with the improvement policy for which the worst case number of steps for the system of multiple identical machines with unrestricted assignment and unit weight jobs to reach a Nash Equilibrium is at least $\Omega(\min\{mn, n \log n \frac{\log m}{\log \log n}\})$ steps.*

6 Concluding Remarks

In this paper we have studied the online load balancing problem that involves selfish jobs (users). We have focused on the number of steps required to reach a Nash Equilibrium and established the convergence time for different strategies. While some strategies provably converge in polynomial time, for the others the convergence time might require an exponential number steps.

In the real world, the convergence time is of high importance, since even if the system starts operation at a Nash Equilibrium, the users may join or leave dynamically. Thus, when designing distributed control algorithms for systems like the Internet, the convergence time should be taken into account.

References

1. E. Altman, T. Basar, T. Jimenez and N. Shimkin, "Routing into two parallel links: Game-Theoretic Distributed Algorithms," *Journal of Parallel and Distributed Computing*, pp. 1367–1381, Vol. 61, No. 9, 2001.
2. B. Awerbuh, Y. Azar, and Y. Richter, "Analysis of worse case Nash equilibria for restricted assignment," unpublished manuscript.
3. Y. Azar, "On-line Load Balancing Online Algorithms – The State of the Art," chapter 8, 178–195, Springer, 1998.
4. T. Boulogne, E. Altman and O. Pourtallier, "On the convergence to Nash equilibrium in problems of distributed computing," *Annals of Operation research*, 2002.

5. P. Brucker, J. Hurink, and F. Werner, "Improving Local Search Heuristics for Some Scheduling Problems, Part I," *Discrete Applied Mathematics*, 65, pp. 97–122, 1996.
6. P. Brucker, J. Hurink, and F. Werner, "Improving Local Search Heuristics for Some Scheduling Problems, Part II," *Discrete Applied Mathematics*, 72, pp. 47–69, 1997.
7. A. Czumaj, P. Krysta and B. Vocking, "Selfish traffic allocation for server farms," STOC 2002.
8. A. Czumaj and B. Vocking, "Tight bounds on worse case equilibria," SODA 2002.
9. E. Even-Dar, A. Kesselman and Y. Mansour, "Convergence Time To Nash Equilibria", Technical Report, available at <http://www.cs.tau.ac.il/~evend/papers.html>
10. G. Finn and E. Horowitz, "Linear Time Approximation Algorithm for Multiprocessor Scheduling," *BIT*, vol. 19, no. 3, 1979, pp. 312–320.
11. Florian, M. and D. Hearn, "Network Equilibrium Models and Algorithms", *Network Routing. Handbooks in RO and MS*, M.O. Ball et al. Editors, Elsevier, pp. 485–550. 1995.
12. D. Fudenberg and D. Levine, "The theory of learning in games," *MIT Press*, 1998.
13. D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis, "The Structure and Complexity of Nash Equilibria for a Selfish Routing Game," *In Proceedings of the 29th ICALP*, Malaga, Spain, July 2002.
14. M. Kearns and Y. Mansour, "Efficient Nash Computation in Large Population Games with Bounded Influence," *In Proceedings of UAI*, 2002.
15. Y. A. Korilis and A. A. Lazar, "On the Existence of Equilibria in Noncooperative Optimal Flow Control," *Journal of the ACM*, Vol. 42, pp. 584–613, 1995.
16. Y.A. Korilis, A.A. Lazar, and A. Orda. Architecting Noncooperative Networks. *IEEE J. on Selected Areas in Communications*, Vol. 13, pp. 1241–1251, 1995.
17. E. Koutsoupias, C. H. Papadimitriou, "Worst-case equilibria," *STACS 99*.
18. R. J. La and V. Anantharam, "Optimal Routing Control: Game Theoretic Approach," *Proceedings of the 36rd IEEE Conference on Decision and Control*, San Diego, CA, pp. 2910–2915, Dec. 1997.
19. M. Littman, M. Kearns, and S. Singh, "An efficient exact algorithm for singly connected graphical games," *In Proceedings of NIPS*, 2002.
20. I. Milchtaich, "Congestion Games with Player-Specific Payoff Functions," *Games and Economic Behavior*, vol. 13, pp. 111–124, 1996.
21. D. Monderer and L. S. Shapley, "Potential games," *Games and Economic Behavior*, 14, pp. 124–143, 1996.
22. J. F. Nash, "Non-cooperative games," *Annals of Mathematics*, Vol. 54, pp. 286–295, 1951.
23. A. Orda, N. Rom and N. Shimkin, "Competitive routing in multi-user communication networks," *IEEE/ACM Transaction on Networking*, Vol 1, pp. 614–627, 1993.
24. R. W. Rosenthal, "A class of games possessing pure-strategy Nash equilibria," *International Journal of Game Theory*, 2, pp. 65–67, 1973.
25. T. Roughgarden and E. Tardos, "How Bad is Selfish Routing?," *In the Proceedings of the 41st Annual IEEE Symposium on the Foundations of Computer Science*, 2000.
26. P. Schuurman and T. Vredeveld, "Performance guarantees of local search for multiprocessor scheduling," *Proceedings IPCO*, pp. 370–382, 2001.
27. S. Shenker, "Making greed work in networks a game-theoretic analysis of switch service disciplines," *IEEE/ACM Transactions on Networking*, Vol. 3, pp. 819–831, 1995.

Nashification and the Coordination Ratio for a Selfish Routing Game^{*}

Rainer Feldmann, Martin Gairing, Thomas Lücking, Burkhard Monien, and
Manuel Rode^{**}

Department of Computer Science, Electrical Engineering and Mathematics
University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany
{obelix,gairing,luck,bm,rode}@uni-paderborn.de

Abstract. We study the problem of n users selfishly routing traffic through a network consisting of m parallel related links. Users route their traffic by choosing private probability distributions over the links with the aim of minimizing their private latency. In such an environment Nash equilibria represent stable states of the system: no user can improve its private latency by unilaterally changing its strategy.

Nashification is the problem of converting any given non-equilibrium routing into a Nash equilibrium without increasing the social cost. Our first result is an $O(nm^2)$ time algorithm for Nashification. This algorithm can be used in combination with any approximation algorithm for the routing problem to compute a Nash equilibrium of the same quality. In particular, this approach yields a PTAS for the computation of a best Nash equilibrium. Furthermore, we prove a lower bound of $\Omega(2^{\sqrt{n}})$ and an upper bound of $O(2^n)$ for the number of greedy selfish steps for identical link capacities in the worst case.

In the second part of the paper we introduce a new structural parameter which allows us to slightly improve the upper bound on the coordination ratio for pure Nash equilibria in [3]. The new bound holds for the individual coordination ratio and is asymptotically tight. Additionally, we prove that the known upper bound of $\frac{1+\sqrt{4m-3}}{2}$ on the coordination ratio for pure Nash equilibria also holds for the individual coordination ratio in case of mixed Nash equilibria, and we determine the range of m for which this bound is tight.

1 Introduction

Motivation-Framework. We study a routing problem in a communication network where n sources of traffic, called users, are going to route their traffic through a shared network. Traffic is routed through links of the network at a certain rate depending on the link, and different users may have different objectives, e.g. speed, quality of service, etc. The users choose routing strategies in order to minimize their private costs in terms of their private objectives

^{*} Partly supported by the DFG-SFB 376 and by the IST Program of the EU under contract numbers IST-1999-14186 (ALCOM-FT), and IST-2001-33116 (FLAGS).

^{**} International Graduate School of Dynamic Intelligent Systems

without cooperating with other users. Such networks are called non-cooperative networks [10]. A famous example of such a network is the internet. Motivated by non-cooperative systems like the internet, combining ideas from game theory and computer science has become increasingly important [4,9,15,16,18].

Such an environment, which lacks a central control unit due to its size or operational mode, can be modeled as a non-cooperative game [17]. Users selfishly choose their private strategies, which in our environment correspond to probability distributions over the paths from their sources to their destinations. When routing their traffic according to the probability distributions chosen, the users will experience an expected latency caused by the traffic of all users sharing edges. Each user tries to minimize its expected individual latency without taking the global performance of the whole network into account. The theory of Nash equilibria [14] provides us with an important solution concept for environments of this kind: a Nash equilibrium is a state of the system such that no user can decrease its individual cost by unilaterally changing its strategy.

The concept of Nash equilibria has become an important mathematical tool in analyzing the behavior of selfish users in non-cooperative systems [18]. Many algorithms have been developed to compute a Nash equilibrium in a general game (see [13] for an overview). The computational complexity of computing a Nash equilibrium in general games is open [18]. The problem becomes even more challenging when global objective functions have to be optimized over the set of all Nash equilibria.

In this work, we concentrate on a special non-cooperative network consisting of a single source and a single destination which are connected by m parallel related links of capacities $c_1, \dots, c_m$. Users $1, \dots, n$ are going to selfishly route their traffics $w_1, \dots, w_n$ from the source to the destination. This model has been introduced by Koutsoupias and Papadimitriou [11]. The individual cost of a user is defined as the maximum expected latency of any link it has chosen with positive probability. Depending on how the latency of a link is defined we distinguish between three variations of the model: In the identical link model all links have equal capacity. In the model of related links the latency for a link j is defined to be the quotient of the sum of the traffics through j and the capacity c_j . In the general case of unrelated links a traffic i induces load w_{ij} on link j . In this work we concentrate on the models of related and identical links.

In our model the social cost is defined to be the expected maximum latency on a link, where the expectation is taken over all random choices of the users. It is well known that, due to the lack of coordination, the users may get to a solution, i.e. a Nash equilibrium, that is suboptimal in terms of the social cost. Koutsoupias and Papadimitriou [11] defined the coordination ratio as the ratio of the social cost of a worst Nash equilibrium and the social cost of the global optimal solution. Results on the coordination ratio depend on the definition of the individual cost and the social cost. A model which uses the sum of the edge latencies as a cost function was considered by Roughgarden and Tardos [19].

In the case that the users are not allowed to randomize their strategies, the set of solutions of the routing problem consists of the set of all pure Nash equilibria. When restricted to pure strategies, the problem of computing a routing (not necessarily an equilibrium one) with minimum social cost is equivalent to

the problem of scheduling n independent jobs on m related parallel machines with minimum makespan [7]. In this environment the problem of Nashification becomes important. The problem of Nashification is to compute an equilibrium routing from a given non-equilibrium one without increasing the social cost. An efficient algorithm for the Nashification problem allows to compute a Nash equilibrium with low social cost by first computing an appropriate non-equilibrium routing with known algorithms for the scheduling problem and then converting this routing into a Nash equilibrium. Here, the intention to centrally nashify a non-equilibrium solution is to provide a routing from which no user has an incentive to deviate.

One way to nashify an assignment is to perform a sequence of greedy selfish steps. A greedy selfish step is a user's change of its current pure strategy to its best pure strategy with respect to the current strategies of all other users. Any sequence of greedy selfish steps leads to a pure Nash equilibrium. However, the length of such a sequence may be exponential in n .

Related work. The selfish routing problem considered in this paper was first introduced by Koutsoupias and Papadimitriou in [11]. The problem was later studied by Mavronicolas and Spirakis [12], who introduced and analyzed fully mixed equilibria of the problem. These works were aimed at analyzing the coordination ratio of the routing game. Czumaj and Vöcking [3] gave two upper bounds of $\Gamma^{-1}(m) + 1 = O(\frac{\log m}{\log \log m})$ and $O(\log \frac{c_{max}}{c_{min}})$, respectively, for the coordination ratio when restricted to pure Nash equilibria and showed that these bounds are tight up to a constant factor. For mixed Nash equilibria they showed an upper bound of $O(\frac{\log m}{\log \log \log m})$ for the coordination ratio.

It has been shown by Fotakis et al. [6] that in our model a pure Nash equilibrium can be computed in polynomial time. In the same work it was proved that the problem of computing a pure Nash equilibrium with minimum (or maximum, respectively) social cost is NP -hard.

In Gairing et al. [7] it was shown that it is NP -hard to decide whether a given routing can be transformed into an equilibrium in k greedy selfish steps, even if the number of links is 2. In the same work a polynomial time algorithm was given which, in the case of identical capacities, nashifies any non-equilibrium assignment. For identical link capacities it was shown that a PTAS exists for approximating the best social cost of a Nash equilibrium within a factor of $1 + \varepsilon$.

The routing problem considered in this paper is equivalent to the multiprocessor scheduling problem. Here, pure Nash equilibria and Nashification translate to local optima and sequences of local improvements. A schedule is said to be *jump optimal* if no job on a processor with maximum load can improve by moving to another processor [20]. Obviously, the set of pure Nash equilibria is a subset of the set of jump optimal schedules. Thus, the strict upper bound of $\frac{1+\sqrt{4m-3}}{2}$ on the ratio between best and worst makespan of jump optimal schedules [2,20] also holds for pure Nash equilibria.

In the model of identical processors every jump optimal schedule can be transformed into a pure Nash equilibrium without changing the makespan. Algorithms for computing a jump optimal schedule on identical processors from any given schedule have been proposed in [1,5,20]. The fastest algorithm is given

by Schuurman and Vredeveld [20]. However, in all algorithms the resulting jump optimal schedule is not necessarily a Nash equilibrium.

Results. In the first part of this work we study the problem of Nashification. Given any pure routing, the goal is to compute a Nash equilibrium with less or equal social cost. We present an $O(nm^2)$ time algorithm which nashifies any pure routing in the model of related link capacities, generalizing the result of Gairing et al. [7]. The routing problem considered here is equivalent to the scheduling problem for related machines. As an immediate consequence of our result, we get a PTAS for computing a Nash equilibrium with minimum social cost by applying the PTAS of Hochbaum and Shmoys [8] to the scheduling problem and nashifying the schedule. Moreover, our algorithm efficiently computes a jump optimal schedule in the model of related processors.

One approach to nashify a routing would be to let the users, in some order, make greedy selfish steps until a Nash equilibrium is reached. We prove that for our routing problem there exists an instance of size polynomial in n such that the maximum length of a sequence of greedy selfish steps is at least $\Omega(2^{\sqrt{n}})$. This result is followed by an $O(2^n)$ upper bound for the length of any sequence of greedy selfish steps in the model of identical capacities. As a consequence we have shown that nashifying a solution using the above mentioned naive approach may take time exponential in n .

Czumaj and Vöcking [3] consider upper bounds on the maximum expected load Λ of any mixed Nash equilibrium in order to get bounds on the coordination ratio. Their two bounds on Λ depend on the number of links m and the fraction of the largest and the smallest link capacity, respectively. However, not only the capacities, but the relation between the sizes of the traffics and capacities determine the individual coordination ratio. We introduce a new structural parameter p that considers the relation between the largest traffic of a user and the capacities of the links. We denote by p the fraction of the sum of all link capacities of links, to which the largest traffic can be assigned causing latency at most the maximum latency $\text{OPT}(w)$ of an optimal assignment.

In the last part of the paper, using the parameter p and similar techniques as in [3], we show the upper bound $\Gamma^{-1}(\frac{1}{p})$ on the coordination ratio for pure Nash equilibria, which is asymptotically tight for all p . Here, Γ^{-1} is the inverse of the Gamma function. Since $p \geq \frac{1}{m}$, our result also shows an asymptotically tight upper bound of $\Gamma^{-1}(m)$ for the coordination ratio, which is a slight improvement of the result in [3]. We prove our results for the individual coordination ratio, that is, the ratio between the maximum expected individual cost $\text{IC}(w, P)$ and the social cost of a globally optimal solution $\text{OPT}(w)$. For every Nash equilibrium P , $\text{IC}(w, P)$ is at most the social cost $\text{SC}(w, P)$, which is defined to be the expected maximum latency. $\text{SC}(w, P)$ equals $\text{IC}(w, P)$ if P is a pure Nash equilibrium. Additionally, we prove an upper bound of $\text{IC}(w, P) \leq \frac{1+\sqrt{4m-3}}{2} \cdot \text{OPT}(w)$. For small m , namely $m \leq 19$, this bound improves on the $\Gamma^{-1}(\frac{1}{p})$ bound, and for $m \leq 5$ it is tight.

2 Notation

Mathematical Preliminaries. For an integer $i \geq 1$, denote $[i] = \{1, \dots, i\}$. Denote Γ the *Gamma function*; that is, for any natural number i , $\Gamma(i+1) = i!$, while for any arbitrary real number $x > 0$, $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$. We will use the fact that $\Gamma(x+1) = x \cdot \Gamma(x)$. The Gamma function is invertible; both Γ and its inverse Γ^{-1} are increasing.

General. We consider a *network* consisting of a set of m parallel *links* $1, 2, \dots, m$ from a *source* node to a *destination* node. Each of n *network users* $1, 2, \dots, n$, or *users* for short, wishes to route a particular amount of traffic along a (non-fixed) link from source to destination. Denote w_i the *traffic* of user $i \in [n]$. Define the $n \times 1$ *traffic vector* w in the natural way. Assume, without loss of generality, that $w_1 \geq w_2 \geq \dots \geq w_n$, and denote $W = \sum_{i=1}^n w_i$ the *total traffic*. 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. A *pure strategy profile* L is represented by an n -tuple $\langle l_1, l_2, \dots, l_n \rangle \in [m]^n$; a *mixed strategy profile* P is represented by an $n \times m$ *probability matrix* of nm probabilities p_{ij} , $i \in [n]$ and $j \in [m]$, where p_{ij} is the probability that user i chooses link j . The *support* of the mixed strategy for user $i \in [n]$, denoted $\text{support}(i)$, is the set of those pure strategies (links) to which i assigns positive probability; so, $\text{support}(i) = \{j \in [m] \mid p_{ij} > 0\}$. For pure strategies we denote $\text{link}(i) = l_i$.

System, Models and Cost Measures. Denote $c_j > 0$ the *capacity* of link $j \in [m]$, representing the rate at which the link processes traffic. In the model of *identical capacities*, all link capacities are equal. Link capacities may vary arbitrarily in the model of *related capacities*. Without loss of generality assume $c_1 \geq \dots \geq c_m$, and denote $C = \sum_{j=1}^m c_j$ the *total capacity*. So, the *latency* for traffic w_i through link j equals $\frac{w_i}{c_j}$. Let P be an arbitrary mixed strategy profile. The *expected latency* of user i on link j is $\lambda_{ij} = \frac{w_i + \sum_{k \in [n], k \neq i} p_{kj} w_k}{c_j}$. The *minimum expected latency* of user i is $\lambda_i = \min_{j \in [m]} \lambda_{ij}$. Denote $\text{IC}(w, P)$ the *maximum expected individual latency*, that is, the maximum, over all users, of the minimum expected latency. Thus, $\text{IC}(w, P) = \max_{i \in [n]} \lambda_i$. The *expected traffic* on link j is defined by $\delta_j = \sum_{i \in [n]} p_{ij} w_i$. We denote the *expected traffic on link j without user i* by $\tau_{ij} = \sum_{k \in [n], k \neq i} p_{kj} w_k = \delta_j - p_{ij} w_i$. The *expected load* A_j on link j is the ratio between the expected traffic on link j and the capacity of link j . Thus, $A_j = \frac{\delta_j}{c_j}$. The *maximum expected load* $\Lambda = \max_{j \in [m]} A_j$ is the maximum (over all links) of the expected load A_j on a link j . Associated with a traffic vector w and a mixed strategy profile P is the *social cost* [11, Section 2], denoted $\text{SC}(w, P)$, which is the expected maximum latency on a link, where the expectation is taken over all random choices of the users. Thus,

$$\text{SC}(w, P) = \sum_{\langle l_1, l_2, \dots, l_n \rangle \in [m]^n} \left(\prod_{k=1}^n p_{kl_k} \cdot \max_{j \in [m]} \frac{\sum_{k: l_k=j} w_k}{c_j} \right).$$

Note that $\text{SC}(w, P)$ reduces to the maximum latency through a link in the case of pure strategies. On the other hand, the *social optimum* [11, Section 2] associated

with a traffic vector w , denoted $\text{OPT}(w)$, is the *least possible* maximum (over all links) latency through a link; thus,

$$\text{OPT}(w) = \min_{\langle l_1, l_2, \dots, l_n \rangle \in [m]^n} \max_{j \in [m]} \frac{\sum_{k: l_k = j} w_k}{c_j}.$$

Nash Equilibria and Coordination Ratio. Say that a user $i \in [n]$ is *satisfied* for the probability matrix P if $\lambda_{ij} = \lambda_i$ for all links $j \in \text{support}(i)$, and $\lambda_{ij} \geq \lambda_i$ for all $j \notin \text{support}(i)$. Otherwise, user i is *unsatisfied*. Thus, a satisfied user has no incentive to unilaterally deviate from its mixed strategy. P is a *Nash equilibrium* [11, Section 2] iff all users $i \in [n]$ are satisfied for P .

Fix any traffic vector w . A *best (worst) Nash equilibrium* is a Nash equilibrium that minimizes (maximizes) $\text{SC}(w, P)$. The *best social cost* is the social cost of a best Nash equilibrium and equals $\text{OPT}(w)$. The *worst social cost* is the social cost of a worst Nash equilibrium and is denoted by $\text{WC}(w)$. Fotakis et al. [6, Theorem 1] consider sequences of selfish steps starting from any arbitrary pure strategy profile. In a *selfish step*, exactly one unsatisfied user is allowed to change its pure strategy. A selfish step is a *greedy selfish step* if the user chooses its best strategy. Selfish steps do not increase the social cost of the initial pure strategy profile. The *coordination ratio* [11] is the *maximum* of $\text{WC}(w)/\text{OPT}(w)$, over all traffic vectors w . Correspondingly, we denote the maximum of $\text{IC}(w, P)/\text{OPT}(w)$ the *individual coordination ratio*.

3 Nashification

In this section, we consider the problem of converting a given pure strategy profile on related links into a Nash equilibrium without increasing the social cost. Every sequence of (greedy) selfish steps yields a Nash equilibrium eventually. However, in Section 3.2 we show that this approach can lead to an exponential number of steps, even on identical links. We present an algorithm which nashifies any pure routing by performing a polynomial number of (not necessarily selfish) moves without increasing the maximum latency.

3.1 A Polynomial Time Algorithm for Nashification

Figure 1 shows the algorithm *Nashify* which converts a pure strategy profile into a Nash equilibrium. A crucial observation for proving the correctness of the algorithm is stated in the following lemma:

Lemma 1. *If user i with traffic w_i performs a greedy selfish step from link j to link k with $c_j \leq c_k$, then no user s with traffic $w_s \geq w_i$ becomes unsatisfied.*

Proof. Let user s be located on link $q = \text{link}(s)$. Since only the loads on link j and k change due to the greedy selfish step of user i we have to show that user s cannot improve by moving to link j . Also we have to show that, if user s is located on link k , s does not become unsatisfied due to the arrival of user

i. Assume first, $q \neq k$. As user s is satisfied, $\frac{\delta_k + w_s}{c_k} \geq \frac{\delta_q}{c_q}$. User i improves by moving to link k , thus $\frac{\delta_j}{c_j} > \frac{\delta_k + w_i}{c_k}$, and we can estimate

$$\begin{aligned} \frac{\delta_j - w_i + w_s}{c_j} &> \frac{\delta_k + w_i}{c_k} + \frac{w_s - w_i}{c_j} \geq \frac{\delta_q}{c_q} - \frac{w_s}{c_k} + \frac{w_i}{c_k} + \frac{w_s - w_i}{c_j} \\ &\geq \frac{\delta_q}{c_q} + (w_s - w_i) \left(\frac{1}{c_j} - \frac{1}{c_k} \right) \geq \frac{\delta_q}{c_q}. \end{aligned}$$

The last inequality holds since $c_k \geq c_j$ and $w_s \geq w_i$. Thus, s cannot improve by moving to link j after i moved. It remains to prove that user s cannot become unsatisfied, if $q = k$. Because of $\frac{\delta_j - w_i + w_s}{c_j} \geq \frac{\delta_j}{c_j} > \frac{\delta_k + w_i}{c_k}$, user s cannot improve by moving to link j . Since user i performed a greedy selfish step, we have

$$\frac{\delta_r + w_s}{c_r} \geq \frac{\delta_r + w_i}{c_r} \geq \frac{\delta_k + w_i}{c_k} \quad \forall r \in [m] \setminus \{j\},$$

and therefore user s cannot improve by moving to any link $r \neq j$. $\square$

For identical links Lemma 1 implies that by moving a user to its best link, no user with larger or equal traffic can become unsatisfied. Thus, by successively moving each user to its best link in order of non-increasing traffic sizes, we end up in a Nash equilibrium without increasing the social cost of the initial routing. This algorithm is described in [7].

```

Nashify()
Input:  $n$  users with traffics  $w_1 \geq \dots \geq w_n$ 
         $m$  links with capacities  $c_1 \geq \dots \geq c_m$ 
        Assignment of users to links
Output: Assignment of users to links with less
        or equal maximum latency, which is a NE
{
  // phase 1:
   $i := n$ ;  $S := \{n\}$ ;
  while  $i \geq 1$  {
    move user  $i$  to link with highest possible index
    without increasing overall maximum latency;
    if  $i$  was moved or  $i \in S$  or  $\text{link}(i) \leq \text{link}(i+1)$ 
    then  $S := S \cup \{i\}$ ;  $i := i - 1$ ;
    else {
      move user  $i$  to link with smallest possible index
      without increasing overall maximum latency;
      if  $i$  was moved then  $S := S \cup \{i\}$ ;  $i := n$ ;
      else break;
    }
  }
  // phase 2:
  while  $\exists i \in S$  {
    make greedy selfish step for user  $i = \min(S)$ ;
     $S := S \setminus \{i\}$ ;
  }
}

```

Fig. 1. Algorithm Nashify

With algorithm Nashify in Figure 1 we generalize this idea to non-identical links. The algorithm works in two phases. At every time $\text{link}(i)$ denotes the link user i is currently assigned to. The main idea is to fill up slow links with users with small traffic as close to the maximum latency as possible in the first phase (but without increasing the maximum latency) and to perform greedy selfish steps for unsatisfied users in the second phase. During the first phase, the set S is used to collect all those users with small traffics, who have been used to fill up slow links. Throughout the whole algorithm, each user in S is located

on a link with non-greater index than any smaller user in S . In other words, the smaller the traffic of a user in S , the slower the link it is assigned to. We may start with $S = \{n\}$, because the above property is trivially fulfilled if S contains only one user. When no further user is added to S by the algorithm, the first phase terminates. In the second phase we successively perform greedy selfish steps for all unsatisfied users, starting with the largest one. That is, we move each user that can improve by changing its link to its best link. Because of the special conditions that have been established by phase 1, and by Lemma 1, these greedy selfish steps do not cause other users with larger traffic to become unsatisfied.

Lemma 2. *After phase 1 the following holds:*

- (1) *All unsatisfied users are in S .*
- (2) *$S = \{n, (n-1), \dots, (n+1-|S|)\}$, that is, S contains the $|S|$ users with smallest traffics.*
- (3) *$i, i+1 \in S \Rightarrow \text{link}(i) \leq \text{link}(i+1)$.*
- (4) *Every user $i \in S$ can only improve by moving to a link with smaller index.*

Proof. The while-loop in phase 1 can only be terminated if either i becomes 0 (the while condition does not hold) or some user $i \notin S$ on a link $\text{link}(i) > \text{link}(i+1)$ cannot be moved to any other link $j < \text{link}(i)$ (the break-command is executed). In the first case all users are in S , which implies (1). In the second case we know that user $(i+1)$ does not fit on any link $j > \text{link}(i+1)$, as user $(i+1)$ was put on the link with maximal index without increasing the maximum latency in the previous run of the loop. In particular, as $\text{link}(i+1) < \text{link}(i)$, user $(i+1)$ cannot be moved to any link $j \geq \text{link}(i)$. Thus, no user $k \notin S$ would fit on any link $j \neq \text{link}(k)$, as $w_k \geq w_i \geq w_{i+1}$, $\forall k \notin S$. This again implies (1).

To see that (2) holds at any time, notice first that a user which is included in S will never be removed from S . Second, whenever a user is added, it is the user with smallest traffic which is not contained in S so far. So S is always a consecutive set of the users with smallest traffics.

(3) is an invariant which holds before and after every run of the while-loop in phase 1. Before the first run it holds because $S = \{n\}$. Whenever a user $i \in S$ is moved, it is moved to a link $j > \text{link}(i)$ with capacity $c_j \leq c_{\text{link}(i)}$. As the traffic of user $(i+1)$ is not larger than the traffic of user i , it would fit on link j , too. But user $(i+1)$ has been considered before user i in the previous run of the while-loop. Thus, user $(i+1)$ is located on some link $\text{link}(i+1) \geq j$, because otherwise it would have been moved to link j . Therefore, $\text{link}(i+1) \geq j$ and (3) remains true after moving user i to link j .

To show (4), consider the last $|S|$ runs of the while-loop, not counting the run which executes the break command (in which no user is moved). (2) implies, that these runs establish a sweep over all users in S , beginning with user n and ending up with the user having the smallest index in S . Each user $i \in S$ is moved to the link with highest index it fits on (without increasing the maximum latency). After user i is assigned, only users with larger or equal traffics are considered. They are located on links $j \leq \text{link}(i)$, which follows from (3). Thus, by moving the remaining users, the maximum latency on any link $j > \text{link}(i)$ is

not decreased, which implies that user i cannot be moved to a link $j > \text{link}(i)$ after the sweep either. As this holds for all users $i \in S$, (4) is valid. $\square$

Theorem 1. *Given any pure strategy profile, algorithm Nashify computes a Nash equilibrium with non-increased social cost, performing at most $(m + 1)n$ moves in sequential running time $\mathcal{O}(m^2n)$.*

Proof. We first prove the correctness of the algorithm Nashify. After phase 1 the conditions from Lemma 2 hold. We now show that these conditions still hold after each run of the while-loop in phase 2.

Consider any run of the while-loop and assume that the conditions of Lemma 2 hold. Let i be the user with smallest index in S , and suppose it is moved from link $j = \text{link}(i)$ to its best link k . Because of (2), we have $i = n + 1 - |S|$. (4) implies $k < \text{link}(i)$ and therefore, $c_k \geq c_j$.

Now let $s \notin S$ be any user on some link $q = \text{link}(s)$.

Due to Lemma 1, user s is satisfied after user i has been moved. Thus, (1) still holds after moving i . As i is removed from S and $i = n + 1 - |S|$, (2) still holds. As i was the user with largest traffic in S , (3) still holds. (3) and the fact that i was moved to a link $j \leq \text{link}(i)$ imply that (4) remains true after the run. At the end of the algorithm, because S is empty and condition (1) still holds, there are no unsatisfied users, i.e., we have a Nash equilibrium. As in no step of the algorithm the overall maximum latency is increased, the algorithm correctly computes a Nash equilibrium with non-increased social cost.

Now we show the bound on the running time. In phase 1, each user is shifted at most once to a link with smaller index. Afterwards it can be shifted at most $m - 1$ times to a link with higher index. So we have at most m moves per user. In phase 2 we have at most $|S| \leq n$ moves. Thus, at most $mn + n$ moves are required altogether. Using appropriate data structures in phase 1, it takes time $\mathcal{O}(1)$ to determine whether a user has to be moved or not. One possibility to do this is to maintain two arrays (x_j) and (y_j) during phase 1, both containing one entry for each link. x_j is the maximal size of a user on link j that would fit on some link $k > j$ without increasing the overall maximum latency. Analogously, y_j is the maximal size of a user on link j that would fit on some link $k < j$. Certainly, $x_m = 0$ and $y_1 = 0$. For each move the algorithm may have to consider m links to find an appropriate target link. It then must update the data structures. Finding the target link and updating the data structures can be done in time $\mathcal{O}(m)$. This yields time complexity $\mathcal{O}(m^2n)$ for phase 1. Phase 2 requires time $\mathcal{O}(mn)$. $\square$

Combining any approximation algorithm for the computation of good routings with the Nashify algorithm yields a method for approximating the best Nash equilibrium. Particularly, using the PTAS for the Scheduling Problem from Hochbaum and Shmoys [8], we get:

Corollary 1. *There is a PTAS for computing a best pure Nash equilibrium.*

This is optimal in the sense that the development of an FPTAS is not possible since the exact computation of the best Nash equilibrium is $\mathcal{NP}$ -complete in the strong sense [6].

Remark 1. Apart from the routing model with related links as considered here, the algorithm can also cope with a slightly relaxed setting. All we need is an order of the users and links, such that $\forall i \in [n-1], j \in [m] : w_{ij} \geq w_{i+1,j}$ and $\forall i \in [n-1], j \in [m-1] : w_{i,j+1} - w_{ij} \geq w_{i+1,j+1} - w_{i+1,j}$. Recall that w_{ij} denotes the contribution of user i to the latency on link j in the model of unrelated links. In the related link model we have the special case $w_{ij} = \frac{w_i}{c_j}$.

3.2 Sequences of Greedy Selfish Steps

Performing greedy selfish steps will eventually convert any routing into a pure Nash equilibrium. However, this may take exponential time even if the links have identical capacities, as shown in the following two theorems. Due to lack of space the proofs are omitted here.

Theorem 2. *There exists an instance of n users with traffics whose bitlength is polynomial in n on $m = \sqrt{n+7} - 1$ identical links for which the maximum length of a sequence of greedy selfish steps is at least $2^{\sqrt{n+7}-3}$.*

Theorem 3. *For any instance with n users on identical links, the length of any sequence of greedy selfish steps is at most $2^n - 1$.*

4 Coordination Ratio

In this section we introduce a structural parameter p . We denote $\mathcal{M}_1 = \{j \in [m] \mid w_1 \leq c_j \cdot \text{OPT}(w)\}$ and $p = \sum_{j \in \mathcal{M}_1} c_j / C$. In other words, p is the ratio between the sum of link capacities of links to which the largest traffic can be assigned causing latency at most $\text{OPT}(w)$ and the sum of all link capacities. With the help of p we are able to prove an upper bound on the individual coordination ratio.

Theorem 4. *For any mixed Nash equilibrium P the ratio between the maximum expected individual latency $\text{IC}(w, P) = \max_{i \in [n]} \lambda_i$ and $\text{OPT}(w)$ is bounded by*

$$\frac{\text{IC}(w, P)}{\text{OPT}(w)} < \begin{cases} \frac{3}{2} + \sqrt{\frac{1}{p} - \frac{3}{4}} & \text{if } \frac{1}{3} \leq p \leq 1, \\ 2 + \sqrt[3]{\frac{1}{p} - 2} & \text{if } \frac{1}{37} \leq p < \frac{1}{3}, \\ \Gamma^{-1}\left(\frac{1}{p}\right) & \text{if } p < \frac{1}{37}. \end{cases}$$

Since $\frac{w_1}{c_1} \leq \text{OPT}(w)$, we have $p \geq \frac{c_1}{C} \geq \frac{1}{m}$. Furthermore, $\text{IC}(w, P) \geq \Lambda$ holds for every assignment. Thus, from Theorem 4 we get the following corollaries:

Corollary 2. *The maximum expected load Λ is bounded from above by $\Lambda \leq \Gamma^{-1}\left(\frac{1}{p}\right) \text{OPT}(w) \leq \Gamma^{-1}(m) \text{OPT}(w)$.*

Corollary 3. *The individual coordination ratio $\text{IC}(w, P)$ is bounded from above by $\text{IC}(w, P) \leq \Gamma^{-1}(m)\text{OPT}(w)$.*

Corollary 2 shows that the generalized upper bound is an improvement of the upper bound $\Gamma^{-1}(m) + 1$ on the maximum expected load Λ in [3]. This leads to an improvement of the upper bound on the coordination ratio [3, Lemma 2.1].

We now introduce a pure Nash equilibrium in Example 1. This can be used to prove that the upper bounds of $\Gamma^{-1}(\frac{1}{p})$ and $\Gamma^{-1}(m)$ are tight.

Example 1 *Let $k \in \mathbb{N}$, and consider the following instance with k different classes of users:*

- Class U_1 : $|U_1| = k$ users with traffic 2^{k-1}
- Class U_i : $|U_i| = 2^{i-1} \cdot (k-1) \prod_{j=1, \dots, i-1} (k-j)$ users with traffic 2^{k-i} for all $2 \leq i \leq k$.

In the same way we define $k+1$ different classes of links:

- Class P_0 : One link with capacity 2^{k-1} .
- Class P_1 : $|P_1| = |U_1| - 1$ links with capacity 2^{k-1} .
- Class P_i : $|P_i| = |U_i|$ links with capacity 2^{k-i} for all $2 \leq i \leq k$.

Consider the following assignment:

- Class P_0 : All users in U_1 are assigned to this link.
- Class P_i : On each link in P_i there are $2(k-i)$ users from U_{i+1} , respectively, for all $1 \leq i \leq k-1$.
- Class P_k : The links from P_k remain empty.

The above assignment is a pure Nash equilibrium L with social cost $\text{SC}(w, L) = k$ and $\text{OPT}(w) = 1$.

Lemma 3. *For each $k \in \mathbb{N}$ there exists an instance with a pure Nash equilibrium L with*

$$k = \frac{\text{SC}(w, L)}{\text{OPT}(w)} \geq \Gamma^{-1}\left(\frac{1}{3p}\right).$$

Lemma 4. *For each $k \in \mathbb{N}$ there exists an instance with a pure Nash equilibrium L with*

$$k = \frac{\text{SC}(w, L)}{\text{OPT}(w)} \geq \Gamma^{-1}(m) \cdot (1 + o(1)).$$

Note that we can prove $k \geq \Gamma^{-1}(\frac{1}{p}) - 1$ in a similar way as in Lemma 3. This shows that the generalized upper bound is tight up to an additive constant for all m whereas due to Lemma 4, $\Gamma^{-1}(m)$ is tight only for large m .

We conclude this section by giving an upper bound on the maximal expected individual latency of a mixed Nash equilibrium, which depends on the number of links m . The same bound also applies to the social cost of a pure Nash equilibrium. This bound improves on Corollary 3 for small m .

Theorem 5. *For any mixed Nash equilibrium P on m links, $IC(w, P)$ is bounded by $IC(w, P) \leq \frac{1+\sqrt{4m-3}}{2} \text{OPT}(w)$. This bound is not tight if $m \geq 6$. For $m \geq 4$, there is no pure Nash equilibrium matching the bound. For $m \geq 2$, there is no fully mixed Nash equilibrium matching the bound.*

Lemma 5. *The bound from Theorem 5 is tight for up to five links. For pure Nash equilibria, the bound is tight for up to three links.*

Theorem 5 slightly extends this result to the maximum expected individual latency $IC(w, P)$ of mixed Nash equilibria. Furthermore, we have shown that the bound on $IC(w, P)$ is tight if and only if $1 \leq m \leq 5$ for mixed Nash equilibria, and if and only if $1 \leq m \leq 3$ for pure Nash equilibria. The bound of $\Gamma^{-1}(m)$ from Corollary 3 is asymptotically tight, but for small numbers of links ($m \leq 19$) the bound from Theorem 5 is better.

References

1. P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems. part ii. *Discrete Applied Mathematics*, 72:47–69, 1997.
2. Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.
3. A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. In *Proc. of SODA 2002*, pp 413–420, 2002.
4. J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. In *Proc. of STOC 2000*, pp 218–227, 2000.
5. G. Finn and E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT*, 19:312–320, 1979.
6. 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 ICALP 2002*, pp 123–134, 2002.
7. M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and P. Spirakis. Extreme nash equilibria. Technical report, FLAGS-TR-03-10, 2002.
8. 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.
9. K. Jain and V. Vazirani. Applications of approximation algorithms to cooperative games. In *Proc. of STOC 2001*, pp 364–372, 2001.
10. Y.A. Korilis, A.A. Lazar, and A. Orda. Architecting noncooperative networks. *IEEE Journal on Selected Areas in Communications*, 13(7):1241–1251, 1995.
11. E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proc. of STACS 1999*, pp 404–413, 1999.
12. M. Mavronicolas and P. Spirakis. The price of selfish routing. In *Proc. of STOC 2001*, pp 510–519, 2001.
13. R.D. McKelvey and A. McLennan. Computation of equilibria in finite games. In H. Amman, D. Kendrick, and J. Rust, editors, *Handbook of Computational Economics*, 1996.
14. J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.
15. N. Nisan. Algorithms for selfish agents. In *Proc. of STACS 1999*, pp 1–15, 1999.
16. N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. of STOC 1999*, pp 129–140, 1999.

17. M.J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.
18. C.H. Papadimitriou. Algorithms, games, and the internet. In *Proc. of STOC 2001*, pp 749–753, 2001.
19. T. Roughgarden and E. Tardos. How bad is selfish routing? In *Proc. of FOCS 2000*, pp 93–102, 2000.
20. P. Schuurman and T. Vredeveld. Performance guarantees of load search for multi-processor scheduling. In *Proc. of IPCO 2001*, pp 370–382, 2001.

Stable Marriages with Multiple Partners: Efficient Search for an Optimal Solution

Vipul Bansal^{*1}, Aseem Agrawal², and Varun S. Malhotra^{*3}

¹ Adobe Systems, I-1A Sector 25A, Noida 201301 India
vbansal@adobe.com

² IBM India Research Lab., IIT Campus Hauz Khas, New Delhi 110016 India
aagrawal@in.ibm.com

³ Stanford University, Electrical Engineering Dept., CA 94305 USA
vsagar@stanford.edu

Abstract. This paper considers the many-to-many version of the original stable marriage problem posed by Gale and Shapley [1]. Each man and woman has a strict preference ordering on the members of the opposite sex and wishes to be matched with upto his or her specified number of partners. In this setup, a polynomial time algorithm for finding a stable matching that minimizes the sum of partner ranks across all men and women is provided. It is argued that this sum can be used as an optimality criterion for minimizing total dissatisfaction if the preferences over partner-combinations satisfy a no-complementarities condition. The results in this paper extend those already known for the one-to-one version of the problem.

1 Introduction

The stable assignment problem, first described by Gale and Shapley [1] as the stable marriage problem, involves an equal number of men and women each seeking one partner of the opposite sex. Each person ranks all members of the opposite sex in strict order of preference. A matching is defined to be stable if no man and woman, who are not matched to each other, prefer each other to their current partners. Gale and Shapley showed the existence of at least one stable matching for any instance of the problem by giving an algorithm for finding it. An introductory discussion of the problem is given by Polya et al. [2] and an elaborate study is presented by Knuth [3]. Variants of this problem have been studied by Gusfield and Irving [4] amongst others, including cases where the orderings are over partial lists or contain ties. It is known that a stable matching can be found in each of these cases individually in polynomial time (Gale and Sotomayor [5], Gusfield and Irving [4]). However in the case of simultaneous occurrence of incomplete lists and ties, the problem becomes NP-hard (Iwama et al. [6]).

^{*} The work was done while the authors were with IBM Research.

While the search version of the problem is shown to be polynomially solvable in most situations, the problem of counting the number of possible stable matchings for a given problem instance is exponential. Irving and Leather [9] showed that the corresponding enumeration problem is $\#P$ -complete (Valiant [7], [8]).

McVitie and Wilson [10] pointed out that the algorithm by Gale and Shapley [1], in which men propose to women, generates a *male-optimal* solution in which every man gets the best partner he can in any stable matching and every woman gets the worst partner she can in any stable matching. They suggested an egalitarian measure of optimality under which the sum of the ranks of partners for all men and women was to be minimized. Irving et al. [11] provided an efficient algorithm to find a stable matching satisfying the optimality criterion of McVitie and Wilson [10].

The present work extends the work of Irving and Leather [9] and Irving et al. [11] to the many-to-many version of the problem, where each man or woman may have multiple partners. Such a situation may arise in the context of matching hospitals with doctors (consultants), buyers with sellers and similar other cases.

In a general many-to-many matching problem, a person may have preferences defined over subsets of the members of the other set. The literature on this can be grouped into two categories based on the assumptions placed on the preference function of a person over the members of the other set. One approach, coming from the economics domain, assumes that each person (or firm) specifies a strict preference ordering on all possible subsets of the set of acceptable partners (or workers). Workers and firms regard each other as *substitutes*, that is, if a worker is a desirable employee to a firm amongst a subset of workers, then he continues to be so even amongst a less desirable subset of workers. Under these assumptions, many results developed for the one-to-one stable matching have their counterparts for one-to-many (Roth and Sotomayor [12]) and many-to-many situations (Roth [13], Sotomayor [14], Martinez et al. [15] and Alkan [16], [17]). The approach has a computational limitation due to exponential nature of the preference function which puts a lower bound on what any algorithm can achieve.

The other approach comes from the computer science domain and is closely related to the original *stable marriage* problem of Gale and Shapley [1]. Here, each man and woman has an upper limit on the number of partners and specifies a preference ordering on acceptable *individuals* of the opposite sex (and not on combinations of them). This approach is simpler, computationally attractive and well suited for situations where it is feasible to rank only the individual items. Taking this approach, Baiou and Balinski [18] showed the existence of male and female optimal assignments and provided characterizations for them. However, determining the stable matchings under an equitable measure of optimality remains an open problem.

In this paper, we work with the second approach to the many-to-many stable matching problem and generalize the notion of optimality proposed for one-to-one matching by McVitie and Wilson [10]. We show that the optimality criterion make sense provided that we include a no-complementarities condition for pref-

erences on combinations of partners. Although this may seem to resemble the first approach, it differs significantly in not requiring the specification of preference functions of exponential size and makes do with only preference ordering on individuals of the opposite set. We then generalize the methodology described by Irving and Leather [9] and Irving et al. [11] and show the existence of the corresponding results for the many-to-many stable marriage problem. In particular, we obtain a polynomial time algorithm for finding an optimal assignment and show how all the stable matchings for the problem can be enumerated. The algorithm and the other results are generic and are not dependent upon the non-complementarities assumption. In addition to the results themselves, the paper also reveals a novel concept (which we call *meta-rotations*, extending the concept of *rotations* by Irving and Leather [9]), which is a generic technique with potential use in solving search problems.

In the next section, we formally describe our model of the multiple partner stable marriage problem. We then introduce our optimality criterion and discuss its usefulness. We then propose a methodology for search space reduction and show that it leads to a polynomial time algorithm for finding an ‘optimal’ matching. Finally, some concluding remarks are presented.

2 Multiple Partner Stable Marriage Model

Let $M = \{m_1, \dots, m_{|M|}\}$ and $F = \{f_1, \dots, f_{|F|}\}$ respectively denote the sets of $|M|$ males and $|F|$ females. Every person has a strict preference order over those members of the opposite sex that he or she considers acceptable. Let L_m be the preference list of male m with the most preferred partner being the first in the list. Similarly, L_f represents the preference ordering of female f . Incomplete lists are allowed so that $|L_m| \leq |F|$, $\forall m \in M$ and $|L_f| \leq |M|$, $\forall f \in F$. Each person also has a quota on the total number of partners with whom he or she may be matched. A person prefers to be matched to a person in his or her preference list than to be matched to fewer than the number of persons specified by his or her quota. Let q_m and q_f denote the respective quotas of male m and female f . The many-to-many stable marriage problem can thus be specified by $P = (M, F, L_M, L_F, Q_M, Q_F)$ where L_M and L_F respectively denote the $|M| \times 1$ and $|N| \times 1$ vectors of male and female preference lists, and Q_M and Q_F represent the $|M| \times 1$ and $|N| \times 1$ vectors of male and female quotas respectively.

Example 1. Consider an instance P specified by: $|M| = 6$, $|N| = 6$

$$\begin{array}{ll}
 L_{m_1} = [f_2, f_6, f_1, f_3, f_5, f_4] & L_{f_1} = [m_5, m_1, m_2, m_3, m_4, m_6] \\
 L_{m_2} = [f_1, f_2, f_4, f_5, f_6, f_3] & L_{f_2} = [m_1, m_2, m_3, m_4, m_5] \\
 L_{m_3} = [f_5, f_3, f_1, f_4, f_2, f_6] & L_{f_3} = [m_3, m_4, m_5, m_1, m_2, m_6] \\
 L_{m_4} = [f_4, f_1, f_2, f_3, f_6, f_5] & L_{f_4} = [m_6, m_3, m_4, m_1, m_5, m_2] \\
 L_{m_5} = [f_1, f_3, f_4, f_5, f_6] & L_{f_5} = [m_4, m_2, m_1, m_6, m_5, m_3] \\
 L_{m_6} = [f_3, f_5, f_6, f_1, f_4] & L_{f_6} = [m_3, m_6, m_2, m_5, m_1, m_4] \\
 Q_M = [2, 3, 1, 2, 1, 3]^T & Q_F = [2, 2, 2, 2, 2, 2]^T
 \end{array}$$

□

A male-female pair, (m, f) is considered a *feasible* pair if m and f are in each other's preference lists. That is, $m \in L_f$ and $f \in L_m$. A *matching* is defined to be a set of feasible male-female pairs $\{(m, f)\}$ such that $\forall m \in M$, m appears in at most q_m pairs and $\forall f \in F$, f appears in at most q_f pairs. A matching γ is *stable* if any feasible pair $(m, f) \notin \gamma$ implies that at least one of the two persons, m or f , is matched to his or her full quota of partners all of whom he or she considers better. This implies that for any stable matching γ , there cannot be any unmatched feasible pair (m, f) that can be paired with both m and f becoming better off. Let γ_m denote the set of partners of male m under the stable matching γ . The set of partners of female f is denoted by γ_f .

Let Γ denote the set of all stable matchings γ for a given instance $P = (M, F, L_M, L_F, Q_M, Q_F)$. Accordingly, Γ_m is the set of all possible sets of partners γ_m that the male m can have in different stable matchings in Γ . For a female f , Γ_f is similarly defined.

3 Notion of Optimality

The optimality criterion for one-to-one matching (by McVitie and Wilson [10]) minimizes the sum of ranks of partners for all males and females. In a multiple partner context, a person's rankings of individuals may not be sufficient to determine his or her preference orderings over combinations of them. For example, (i) a person who wants upto 2 matches need not be indifferent to the partner combinations $(1, 6)$, $(2, 5)$ and $(3, 4)$ (where the numbers denote the partners' ranks in the person's preference list), (ii) he or she may actually prefer $(1, 6)$ over $(2, 4)$ due to an overbearing preference for the partner ranked 1 even though $(1, 6)$ has a greater sum of ranks than $(2, 4)$, and (iii) the person may prefer $(2, 6)$ over $(2, 5)$ if partners ranked 2 and 6 are complementary and their combination is of greater value to him or her than the combination $(2, 5)$.

Out of the three complications presented in the above illustration, the first two can be avoided by using weighted preference lists. However, such an approach would come at a significant cost because (i) an individual needs to compute mutually consistent weights that capture his or her preference ordering over all acceptable partner-combinations, and (ii) the weights need to be normalized across individuals because the optimality criterion would consider the sum of weights across all individuals in the matching.

We show below that the many-to-many stable matching has the structure due to which one can explicitly rule out all such complications provided that a *no-complementarities* condition is imposed on preference orderings of males and females over combinations of partners. The no-complementarities condition (for males) states: Given two sets of partners, A_1 and A_2 ($A_1, A_2 \subset F$), if a male m prefers A_1 at least as much as A_2 , and m strictly prefers f_1 over f_2 ($f_1, f_2 \in F \setminus (A_1 \cup A_2)$), then m strictly prefers $A_1 \cup \{f_1\}$ over $A_2 \cup \{f_2\}$. This assumption is similar to the substitutability assumption widely used in the literature (For example: Roth [13] and Martinez et al. [15]) and can be derived

from it. The assumption is intuitive and is plausible in most situations except where specific partners are strong complements of one-another.

Given a set of partners γ_m in a stable matching γ , we define the *dissatisfaction score* $DS(\gamma_m)$ of male m to be the sum of position numbers (or ranks) of the females in γ_m in his preference list L_m . Thus, $DS(\gamma_m) = \sum_{f \in \gamma_m} R_m(f)$ where $R_m(f)$ is the rank given by male m to the female f . The dissatisfaction score $DS(\gamma_f)$ for a female f is similarly defined. The dissatisfaction score of a matching γ is defined as the sum of the dissatisfaction scores of all persons involved. We show that the dissatisfaction score as defined above and the no-complementarities condition stated earlier impose a strict ordering on a person's preferences over all possible set of partners that he or she may have in any stable marriage for any given instance of the problem $P = (M, F, L_M, L_F, Q_M, Q_F)$.

First, we note some useful properties of the many-to-many stable marriage problem (due to Baiou and Balinski [18]). The results below are stated for males. They are also true for females due to the symmetrical nature of the problem.

Property 1. A male $m \in M$ is assigned the same number of partners, $N_P(m)$, in all stable matchings. Further, if $N_P(m) < q_m$, then m has the same set of partners in all stable matchings.

Property 2. Suppose γ and γ^* are stable matchings that assign different sets of partners to a male m . Then there is one (say γ) such that if $(m, f) \in \gamma$ and $(m, f^*) \in \gamma^* \setminus \gamma$, then $R_m(f) < R_m(f^*)$.

A useful corollary of Property 2 is that if a person is assigned different sets of partners in different stable matchings, then his or her least preferred partner in each of them must be different. Let the function *min* specify the least preferred partner of a person amongst his or her given set of partners. Accordingly, $\min(\gamma_m)$ and $\min(\gamma_f)$ specify the least preferred partners of male m and female f in the stable matching γ .

Property 3. Suppose γ and γ^* are stable matchings that assign different sets of partners to a male m . Then, $R_m(\min(\gamma_m)) < R_m(\min(\gamma_m^*))$ implies $R_f(\min(\gamma_f)) > R_f(\min(\gamma_f^*))$ for all f such that $(m, f) \in (\gamma \setminus \gamma^*) \cup (\gamma^* \setminus \gamma)$.

Property 3 leads to the conclusion that if any pair $(m, f) \in \gamma$, for some stable matching $\gamma \in \Gamma$, it is not possible for both m and f to be simultaneously worse off (or simultaneously better off) in any stable matching γ' ($(m, f) \notin \gamma'$) than they are in γ . The properties 1 and 2 stated above and the no-complementarities condition lead us to the following important result (stated for males):

Theorem 1. Suppose γ_m and γ_m^* are two distinct sets of partners of the male m under the stable matchings γ and γ^* respectively. Then, (i) $DS(\gamma_m) \neq DS(\gamma_m^*)$, and (ii) If $DS(\gamma_m) < DS(\gamma_m^*)$ then m prefers γ_m over γ_m^* and vice versa.

Proof. By Property 1, m must be matched to the same number of females in all stable matchings. Without loss of generality, Property 2 allows us to assume

that $R_m(\min(\gamma_m)) < R_m(\min(\gamma_m^*))$ (implying that $\min(\gamma_m^*)$ is to the right of $\min(\gamma_m)$ in male m 's preference list L_m). Each female in L_m to the right of $\min(\gamma_m)$ corresponds to at most one set of partners for m (amongst whom she is the least preferred partner).

We consider the first female to the right of $\min(\gamma_m)$ in L_m who is the least preferred partner of m in some stable marriage, say $\gamma^{**} \in \Gamma$. By Property 2, we note that any female $f \in \gamma_m^{**} \setminus \gamma_m$ must be to the right of any female $f' \in \gamma_m$ in L_m . Further, since $|\gamma_m| = |\gamma_m^{**}|$, each such female f is a replacement of some other female $f' \in \gamma_m$ who was to the left of f in the preference list L_m . Each replacement of f' with f leads to an increase in the dissatisfaction score of m so that $DS(\gamma_m) < DS(\gamma_m^{**})$. Further, the no-complementarities assumption applied successively to each such replacement implies that m strictly prefers γ_m over γ_m^{**} .

If γ_m^{**} is identical to γ_m^* , it completes the proof. Else continue the above step until $\gamma_m^{**} = \gamma_m^*$. $\square$

Theorem 1 is significant because it gives us a strict preference ordering over all possible sets of acceptable stable marriage partners for any person using only the preference orderings on individuals and a no-complementarities assumption on the preferences over combinations. This obviates the need for preference functions of exponential size which *a priori* specify the orderings over all possible subsets of members of the opposite sex. We can now use the terms *better* or *worse* unambiguously to compare any two sets of stable marriage partners of a person and we can do so by comparing either the least preferred partners or the dissatisfaction scores.

Theorem 1 allows us to propose that the minimization of the sum of dissatisfaction scores across all persons can be used as an egalitarian measure of optimality for the many-to-many stable marriage problem specified by $P = (M, F, L_M, L_F, Q_M, Q_F)$ for which the preferences over combinations of partners additionally satisfy the no-complementarities condition. Formally,

$$\text{Minimize } \gamma \in \Gamma : \sum_{m \in M} DS(\gamma_m) + \sum_{f \in F} DS(\gamma_f). \quad (1)$$

The optimality criterion can be restated as:

$$\text{Minimize } \gamma \in \Gamma : \sum_{(m,f) \in \gamma} [R_m(f) + R_f(m)]. \quad (2)$$

We note that the above optimality measure is also the natural generalization of the one proposed for the one-to-one marriage problem by McVitie and Wilson [10] for which a polynomial time algorithm was later provided by Irving et al. [11]. The treatment of optimality using dissatisfaction score can be generalized to include weighted ranks which allows persons to specify their preferences more accurately. The results presented in this paper are also true for the weighted rank preferences. The only requirement is that the preference orderings on individuals be strict. We will henceforth refer only to the case with unity weights for simplicity of exposition.

4 Reduction of Search Space

Irving and Leather [9] and Irving et al. [11] describe a methodology for the single partner stable marriage problem in which starting with the male-optimal solution, all the stable matchings can be generated by successive elimination of what they call *rotations*. They conclude that the enumeration problem is #P-complete and provide a polynomial time algorithm for finding a matching which satisfies the egalitarian optimality criterion.

The basis of their methodology lies in the process by which the *rotations* get exposed and eliminated. A rotation is a cycle comprising r male-female pairs (m_i, f_i) such that f_i is m_i 's current match and f_{i+1} ($i+1$ is taken modulo r) is the second in m_i 's list. Since every individual is matched to only one partner, it is convenient to eliminate all females to the left to the current match of a male and all males to the right of the current match of a female from the male-optimal solution (By definition of the male-optimal solution, these cannot occur in any stable matching). This step is sufficient to ensure that at least one rotation gets exposed as long as the female-optimal solution is not reached. The exposed rotation is then eliminated (m_i gets paired with f_{i+1}) and the process continued till all rotations are eliminated.

It may be tempting to apply the above methodology to the many-to-many context - it is known (due to Baiou and Balinski [18]) that the corresponding male-optimal stable matching γ^M always exists and can be found in $O(n^2)$ steps. Similar to its one-to-one counterpart, γ^M has the property that there is no other stable matching in which any of the males is better off (has a lower dissatisfaction score) or any of the females worse off (has a greater dissatisfaction score).

Beyond obtaining the male-optimal solution, applying the methodology to the many-to-many case is far from straightforward. Consider γ^M : a male m may be matched to multiple and non-contiguous females in his preference list L_m . The fate of the females (not matched to m) in L_m who fall between the matched ones in L_m is not immediately clear. Indeed, we show later that all such females can be deleted from the list L_m from further consideration; on the other hand the males not matched to a female f but who is lying between the matched ones in L_f need to be retained as they can occur in stable marriages yet to be identified. This illustrates why the pruning (or elimination) step is tricky - not removing the necessary entries may result in no rotations being exposed while unnecessary deletion may lead to some stable marriages not being found.

Another problem is in defining a meaningful rotation. In the single partner case, a male getting matched to his second-most-preferred partner constitutes logical *atomic* progress from the male-optimal towards the female-optimal solution. In the multiple partner setup, such a progress may be possible in multiple ways - a subset of a male's current partners may be swapped with another subset such that the dissatisfaction score of the male increases by one. It is easy to see that the number of such choices is exponential.

A key contribution of this paper is to define a generalized concept of rotation and show how they can be exposed and eliminated. This is achieved through the definitions that follow.

Definition 1 (Initial Pruning). Given the male-optimal solution γ^M of an instance P , the initial pruning step consists of: (a) removing the females $f \notin \gamma_m^M$ from each male list L_m for which $R_m(f) < R_m(\min(\gamma_m^M))$, (b) removing the males m from a female list L_f for which $R_f(m) > R_f(\min(\gamma_f^M))$, (c) removing f from m 's list if the (so far) reduced list of f does not contain m , and (d) removing m from f 's list if the (so far) reduced list of m does not contain f .

Next, we introduce the concept of a *meta-rotation*, which is a many-to-many generalization of the concept of a *rotation*. The definition of a meta-rotation is motivated by the observation that if a person becomes better off or worse off, his or her least preferred partner (or *min*) must change.

Definition 2 (Meta-rotation). Given a problem instance P , a meta-rotation ρ is defined as an ordered sequence of feasible male-female pairs $\{(m_0, f_0), \dots, (m_i, f_i), \dots, (m_{r-1}, f_{r-1})\}$, $r \geq 2$ such that $f_{(i+1) \bmod r} = \text{sm}(\gamma_{m_i}^M)$ and $m_i = \min(\gamma_{f_i}^M)$. Here, $\text{sm}(\gamma_{m_i}^M)$ denotes the female to the immediate right of $\min(\gamma_{m_i}^M)$ in his current list. Such a meta-rotation is said to be exposed in P relative to its male-optimal stable matching, γ^M .

Definition 3 (Meta-rotation Elimination). A meta-rotation ρ exposed in P , is said to be eliminated when for each $(m_i, f_i) \in \rho$, m_i gets matched to $f_{(i+1) \bmod r}$ in place of f_i and for each female f_i , her preference list L_{f_i} is modified to delete all males to the right of her new least preferred partner, and the preference lists of the males are correspondingly modified to remove females that no longer have them in their preference lists.

Definition 4 (R-instance). A problem instance $\hat{P} = (M, F, \hat{L}_M, \hat{L}_F, Q_M, Q_F)$ is defined to be a reduced instance (or R-instance) of the problem instance $P = (M, F, L_M, L_F, Q_M, Q_F)$ if it is obtained either by applying initial pruning on P or by the elimination of a meta-rotation from another R-instance $\hat{P}^*$ of P .

Example 2. For the problem instance in Example 1, the male-optimal solution is: $\gamma^M = \{(m_1, f_2), (m_1, f_6), (m_2, f_1), (m_2, f_2), (m_2, f_4), (m_3, f_5), (m_4, f_4), (m_4, f_3), (m_5, f_1), (m_6, f_3), (m_6, f_5), (m_6, f_6)\}$. Application of initial pruning yields an R-instance $\hat{P}$ with the following male and female lists (the matched pairs are in boldface):

$$\begin{array}{ll} \hat{L}_{m_1} = [\mathbf{f_2}, \mathbf{f_6}, f_1, f_3, f_5, f_4] & \hat{L}_{f_1} = [\mathbf{m_5}, m_1, \mathbf{m_2}] \\ \hat{L}_{m_2} = [\mathbf{f_1}, \mathbf{f_2}, \mathbf{f_4}, f_5, f_6, f_3] & \hat{L}_{f_2} = [\mathbf{m_1}, \mathbf{m_2}] \\ \hat{L}_{m_3} = [\mathbf{f_5}, f_3, f_4, f_6] & \hat{L}_{f_3} = [m_3, \mathbf{m_4}, m_5, m_1, m_2, \mathbf{m_6}] \\ \hat{L}_{m_4} = [\mathbf{f_4}, \mathbf{f_3}, f_5] & \hat{L}_{f_4} = [m_6, m_3, \mathbf{m_4}, m_1, m_5, \mathbf{m_2},] \\ \hat{L}_{m_5} = [\mathbf{f_1}, f_3, f_4, f_5, f_6] & \hat{L}_{f_5} = [m_4, m_2, m_1, \mathbf{m_6}, m_5, \mathbf{m_3}] \\ \hat{L}_{m_6} = [\mathbf{f_3}, \mathbf{f_5}, \mathbf{f_6}, f_4] & \hat{L}_{f_6} = [m_3, \mathbf{m_6}, m_2, m_5, \mathbf{m_1}] \end{array}$$

The meta-rotation $\rho_1 = \{(m_3, f_5), (m_6, f_3), (m_2, f_4)\}$ is seen to be exposed in $\hat{P}$. $\square$

Note that every R-instance $\hat{P}$ of P is a stable marriage problem instance in its own right and therefore has meta-rotations defined on it unless the male optimal matching coincides with the female optimal matching. Since the preference lists in $\hat{P}$ are a subset of those in P , it is clear from the definition of stability that a stable matching in P , if it can be defined in $\hat{P}$, will also be stable in $\hat{P}$. With the help of the foregoing definitions, we can now show that if we start with the male-optimal stable matching γ^M of P and successively identify and eliminate the meta-rotations, we would finally reach the female-optimal stable matching γ^F of P . If the process is carried out exhaustively for all possible sequences of meta-rotation eliminations, it would generate all possible stable marriages for the problem instance P .

The following lemma states that the male-female pairs eliminated by initial pruning do not occur in any stable matching.

Lemma 1. *The sets of all possible stable matchings for the R-instance obtained by initial pruning of P is the same as Γ , the set of stable matchings for P .*

Proof. For step (a) of initial pruning, we note by Property 2 that if $(m, f) \in \gamma \setminus \gamma^M$, then $R_m(f) > R_m(\min(\gamma_m^M))$. Therefore, a female f who is not paired with m in γ^M and is preferred over $\min(\gamma_m^M)$ cannot be paired with m in any stable matching. For males deleted from a female f 's list in (b), we note that they are preferred less by f than her least preferred partner in her worst possible set of partners and hence cannot be paired with f in any stable matching. Steps (c) and (d) only remove infeasible pairs. We have thus shown that pairs removed from consideration by initial pruning cannot occur in any stable matching for the problem instance P . Therefore, all stable matchings of P can also be found in $\hat{P}$. Further, removal of some pairs cannot introduce any new stable matching. Hence the result. $\square$

Next, we show that every R-instance of P has a corresponding male-optimal stable matching which is also a stable matching for original instance P .

Lemma 2. *An R-instance $\hat{P}$ of the stable marriage problem P has a male-optimal stable matching $\hat{\gamma}^M$ in which each male m is matched to the first $N_P(m)$ females in his list $\hat{L}_m$ and for each female f , her least preferred partner is the right-most in her list $\hat{L}_f$. Further, $\hat{\gamma}^M \in \Gamma$, the set of stable matchings for P .*

Proof. The proof is by induction. An R-instance $\hat{P}$ can be generated from P by initial pruning and then by successive meta-rotation eliminations. For the $\hat{P}$ obtained by initial pruning, by Lemma 1 the male-optimal stable matching γ^M for P is the required stable matching.

Let $\hat{P}^*$ be an R-instance of P which satisfies Lemma 2. Therefore, $\hat{\gamma}^{*M} \in \Gamma$. Consider an R-instance $\hat{P}$ obtained by eliminating an exposed meta-rotation ρ from $\hat{P}^*$. Clearly each female f 's least preferred partner occurs last in her list $\hat{L}_f$ by the definition of meta-rotation elimination. A male m who is not in the meta-rotation ρ continues to be matched to the first $N_P(m)$ females in his list $\hat{L}_m$. A male m who is a part of ρ ends up pairing with the female immediately

to the right of his least preferred match in lieu of a female whom he preferred more. The latter is removed from his list. Thus, m continues to be matched to the first $N_P(m)$ females in his new list $\hat{L}_m$.

Note that the number of partners of each male and female does not change in a meta-rotation elimination. Therefore, the newly updated set of matched pairs obtained by eliminating ρ continues to be a matching for P . We next show that it constitutes a stable matching for the R-instance $\hat{P}^{**}$, obtained by initial pruning of P .

For contradiction, suppose that it is not a stable matching for $\hat{P}^{**}$. Then there exists a pair (m, f) not in this matching, where $m \in \hat{L}_f^{**}$ and $f \in \hat{L}_m^{**}$, such that each of m and f is either not matched to his or her full quota or prefers the other person (f or m as the case may be) over his or her least preferred partner in this matching. Consider the male m . In the male-optimal stable matching for $\hat{P}^{**}$, m is matched to the first $\min(q_m, |\hat{L}_m^{**}|)$ females in his list $\hat{L}_m^{**}$. Therefore, $N_P(m) = \min(q_m, |\hat{L}_m^{**}|)$. Similarly, for the stable matching $\hat{\gamma}_M^{**}$, $N_P(m) = \min(q_m, |\hat{L}_m^*|)$. For $N_P(m) < q_m$ to be true, we need to have $|\hat{L}_m^{**}| = |\hat{L}_m^*|$. But $N_P(m) < q_m$ would imply that f is not in $\hat{L}_m^*$ so that $|\hat{L}_m^{**}| \geq |\hat{L}_m^*| + 1$. Therefore, it follows that $N_P(m) = q_m$. Our assumption on the existence of the pair (m, f) now implies that m must prefer f over his current least preferred partner. If this is true, (m, f) must belong to some stable matching for P and f must have been deleted from the list of m in a meta-rotation elimination step. At that time, f considered m worse than her least preferred partner. Since meta-rotation eliminations can only improve the least preferred partners of females, f must therefore consider m worse than her current least preferred partner. Also, when (m, f) was unpaired by a meta-rotation elimination, m became worse off. So if f were not to be matched to her full quota, she could have retained her pairing with m . Since f did not do so, $N_P(f) = q_f$. We now have a violation of the conditions of our assumption which required f to either be matched to less than her full quota or prefer m over her current least preferred partner. Therefore, the matching (say $\hat{\gamma}^{**}$) is a stable matching for $\hat{P}^{**}$.

By Lemma 1, $\hat{\gamma}^{**}$ is also a stable matching for P . Further, since $\hat{\gamma}^{**}$ is contained within the preference lists of $\hat{P}$, it is also a stable matching for $\hat{P}$. This gives us the required male optimal stable matching $\hat{\gamma}^M$ for $\hat{P}$. $\square$

Example 3. Lemma 2 can be easily verified in Example 2. The elimination of meta-rotation $\rho_1 = \{(m_3, f_5), (m_6, f_3), (m_2, f_4)\}$ gives an R-instance for which the male optimal solution $\hat{\gamma}^M$ is: $\{(m_1, f_2), (m_1, f_6), (m_2, f_1), (m_2, f_2), (m_2, f_5), (m_3, f_3), (m_4, f_4), (m_4, f_3), (m_5, f_1), (m_6, f_5), (m_6, f_6), (m_6, f_4)\}$. It can be checked that this also constitutes a stable matching for P . $\square$

We now show that if there is a stable matching for P in which a male is worse off than in the male-optimal stable matching corresponding to an R-instance of P , then there must exist a meta-rotation exposed in that R-instance.

Lemma 3. *If m and f are deleted from each other's preference lists on the elimination of meta-rotation ρ exposed in $\hat{P}$ relative to its male-optimal stable*

matching $\hat{\gamma}^M$, and $(m, f) \notin \hat{\gamma}^M$, then (m, f) cannot occur in any stable matching γ of P .

Proof. Let $\hat{P}^*$ be the R-instance obtained by eliminating ρ from $\hat{P}$. Let m_1 be the least preferred partner of f in $\hat{\gamma}^*$. Then, f prefers m_1 to m . Let f_1 and f_2 be the least preferred partners of m in $\hat{\gamma}^M$ and $\hat{\gamma}^{*M}$ respectively. Since m is matched to the first $N_P(m)$ females in $\hat{L}_m$, therefore $R_m(f) > R_m(f_2) > R_m(f_1)$. If (m, f) is a pair in some stable matching γ of P , then both m and f are better off in the stable matching $\hat{\gamma}^{*M}$ than they are in γ which cannot be true. $\square$

Lemma 4. *Given an R-instance $\hat{P}$ of the stable marriage problem P , with its male-optimal stable matching $\hat{\gamma}^M$, if there is a male m for whom $R_m(\min(\hat{\gamma}_m^M)) < R_m(\min(\gamma_m))$ for some stable marriage $\gamma \in \Gamma$ of P , then there is at least one meta-rotation exposed in $\hat{P}$.*

Proof. Suppose that there is a male m for whom $R_m(\min(\hat{\gamma}_m^M)) < R_m(\min(\gamma_m))$ for some stable marriage $\gamma \in \Gamma$ of P . We construct a sequence $\{(m_i, f_i)\}$ as follows: Let $m_0 = m$. By Lemma 1 and Lemma 3, no females to the right of $\min(\hat{\gamma}_{m_0}^M)$ who can be paired with m_0 in some stable matching get deleted during initial pruning and meta-rotation elimination, therefore, m_0 is worse off in γ implies that $smin(\hat{\gamma}_{m_0}^M)$ is defined in $\hat{L}_{m_0}$. Let $f_1 = smin(\hat{\gamma}_{m_0}^M)$. In γ , m_0 either gets matched to f_1 or to someone further right in his list. In the former case, f_1 becomes better off in γ because m_0 is to the left of her least preferred partner in $\hat{\gamma}^M$. In the latter case, f_1 must be matched to her full quota of partners, each of whom she prefers over m_0 otherwise the matching γ would not be stable. In this case too, f_1 becomes better off in γ than in $\hat{\gamma}^M$.

Now, f_1 is better off in γ implies that $R_{f_1}(\min(\gamma_{f_1})) < R_{f_1}(\min(\hat{\gamma}_{f_1}^M))$. Let $m_1 = \min(\hat{\gamma}_{f_1}^M)$. Clearly, $(m_1, f_1) \notin \gamma$ as all of f_1 's partners in γ are preferred by her over m_1 . Since f_1 is better off in γ , and (m_1, f_1) are partners in $\hat{\gamma}^M$ which by Lemma 2 is a stable matching of P , therefore, m_1 must be worse off in γ using Property 3.

We continue to build the chain where $f_{i+1} = smin(\hat{\gamma}_{m_i}^M)$ and $m_i = \min(\hat{\gamma}_{f_i}^M)$. We cannot progress indefinitely, so the sequence $\{(m_i, f_i)\}$ must cycle. Thus, we have constructively shown the existence of a meta-rotation in $\hat{P}$ (relative to the matching $\hat{\gamma}^M$). $\square$

We can designate this meta-rotation as the meta-rotation generated in $\hat{P}$ by a male m who gets a worse set of partners in the new stable matching after eliminating the meta-rotation.

Lemma 5. *If $\{(m_0, f_0), \dots, (m_{r-1}, f_{r-1})\}$, $r \geq 2$ is a meta-rotation exposed in some R-instance $\hat{P}$ of P relative to its male-optimal stable matching $\hat{\gamma}^M$ and in some stable matching $\gamma \in \Gamma$, $R_{m_k}(\min(\gamma_{m_k})) > R_{m_k}(\min(\hat{\gamma}_{m_k}^M))$ for a particular male m_k , then for each male $m_i, i \in 0 \dots r-1$ in the meta-rotation, $R_{m_i}(\min(\gamma_{m_i})) > R_{m_i}(\min(\hat{\gamma}_{m_i}^M))$.*

Proof. On the lines of the proof for Lemma 4, we note that if m_k becomes worse off in γ than in $\hat{\gamma}$, then the female $f_{(k+1) \bmod r}$ should become better off. Continuing further, this should make the male $m_{(k+1) \bmod r}$ worse off. The Lemma 5 therefore follows. $\square$

Corollary 1. *If $\{(m_0, f_0), \dots, (m_{r-1}, f_{r-1})\}$, $r \geq 2$ is a meta-rotation exposed in some R-instance $\hat{P}$ of P relative to its male-optimal matching $\hat{\gamma}^M$ and for some stable matching $\gamma \in \Gamma$, $R_{f_k}(\min(\gamma_{f_k})) < R_{f_k}(\min(\hat{\gamma}_{f_k}^M))$ for a particular female f_k , then for each female f_i , $i \in 0 \dots r - 1$ in the meta-rotation, $R_{f_i}(\min(\gamma_{f_i})) < R_{f_i}(\min(\hat{\gamma}_{f_i}^M))$.* $\square$

We are now in a position to show that every stable marriage for the problem instance P can be obtained as the male-optimal solution for some R-instance $\hat{P}$ of P .

Theorem 2. *Given a stable matching $\gamma \in \Gamma$ for P , γ is identical to the male-optimal stable matching $\hat{\gamma}^M$ for some R-instance $\hat{P}$ of P .*

Proof. Consider the R-instance $\hat{P}$ obtained by applying initial pruning to P . If $\gamma = \gamma^M$ then by Lemma 1, $\gamma = \hat{\gamma}^M$ also. Therefore, $\hat{P}$ is the required R-instance.

Suppose $\gamma \neq \hat{\gamma}^M$. This implies that there is at least one male m who is worse off in γ than in $\hat{\gamma}^M$. By Lemma 4, there exists a meta-rotation ρ exposed in $\hat{P}$. We also note (by proof methodology of Lemma 4) that ρ can be identified by starting at m so that m is included in the meta-rotation. We can eliminate ρ from $\hat{P}$ to yield a new R-instance $\hat{P}^*$. By Lemma 5, we note that the least preferred partner of every male included in ρ must also be worse than in $\hat{\gamma}^M$. By Lemma 3, a meta-rotation elimination does not remove any female to the right of the least preferred partner of a male from his list unless they cannot be paired in any stable marriage, therefore, it is ensured that for any male $m_i \in M$, $R_{m_i}(\min(\hat{\gamma}_{m_i}^{*M})) \leq R_{m_i}(\min(\gamma_{m_i}))$ which states that m_i prefers his least preferred partner in γ^{*M} at least as much as the least preferred partner in γ .

Suppose now, $R_{m_i}(\min(\hat{\gamma}_{m_i}^{*M})) = R_{m_i}(\min(\gamma_{m_i}))$ for all $m_i \in M$. By Lemma 2, we know that $\hat{\gamma}^{*M} \in \Gamma$, the set of stable matchings for P . Since the least preferred partners uniquely define a person's set of partners, it follows that $\hat{\gamma}^{*M} = \gamma$ and $\hat{P}^*$ is the required R-instance.

Otherwise, there is at least one male m' who is worse off in γ than in $\hat{\gamma}^{*M}$. We can again apply Lemma 4 to find a meta-rotation exposed in $\hat{P}^*$. This process terminates only when we get a R-instance such that for any $m_i \in M$, $R_{m_i}(\min(\hat{\gamma}_{m_i}^{*M})) = R_{m_i}(\min(\gamma_{m_i}))$. At that point, the male-optimal stable matching for the R-instance is identical to γ . $\square$

Therefore, every stable matching $\gamma \in \Gamma$ for P can be obtained by successive application of meta-rotation eliminations on the R-instance obtained by initial pruning of P . Since the male-optimal stable matching $\hat{\gamma}^M$ is unique for a given R-instance $\hat{P}$, we have also established a one-to-one correspondence between the stable matchings γ for P and the R-instances $\hat{P}$ of P .

Example 4. In Example 2, the elimination of meta-rotation $\rho_1 = \{(m_3, f_5), (m_6, f_3), (m_2, f_4)\}$ leads to the meta-rotation $\rho_2 = \{(m_1, f_6), (m_2, f_1)\}$ becoming exposed. Elimination of ρ_2 gives the female-optimal solution $\gamma^F = \{(m_1, f_2), (m_1, f_1), (m_2, f_2), (m_2, f_5), (m_2, f_6), (m_3, f_3), (m_4, f_4), (m_4, f_3), (m_5, f_1), (m_6, f_5), (m_6, f_6), (m_6, f_4)\}$. $\square$

Define Ω to be the set of meta-rotations for the problem instance P . $\rho \in \Omega$ if and only if ρ is a meta-rotation exposed in some R-instance $\hat{P}$ of the problem instance P . We note that Ω is obtained by successively eliminating meta-rotations from the R-instance obtained by initial pruning of P .

Using the results obtained thus far, it is now straightforward to show that no pair (m, f) can belong to two different meta-rotations and thereby establish that there exists a one-to-one correspondence between the set of stable matchings for P and the closed subsets of a meta-rotation poset $\Psi, \leq$ defined over its set of meta-rotations Ω . (The partially ordered set $\Psi, \leq$ is defined using the predecessor relationship $\leq$: a meta-rotation ρ_1 is a predecessor of a meta-rotation ρ_2 if ρ_1 must be eliminated for ρ_2 to become exposed). The required proofs flow trivially from the description by Irving and Leather [9] for one-to-one matching and using the results obtained in this paper.

Example 5. In Example 4, the meta-rotation $\rho_1 = \{(m_3, f_5), (m_6, f_3), (m_2, f_4)\}$ is a predecessor of meta-rotation $\rho_2 = \{(m_1, f_6), (m_2, f_1)\}$. $\square$

5 An Efficient Algorithm for ‘Optimal’ Stable Matching

Given a meta-rotation $\rho = \{(m_0, f_0), \dots, (m_{r-1}, f_{r-1})\}$, $r \geq 2$ define its weight w_ρ in the following manner: $w_\rho = \sum_{i=0}^{r-1} [R_{m_i}(f_i) + R_{f_i}(m_i) - R_{m_i}(f_{i+1}) - R_{f_i}(m_{i-1})]$ where $(i+1)$ and $(i-1)$ are taken modulo r . The weight of a closed subset A of the meta-rotation poset $\Psi, \leq$ is defined to be the sum of the meta-rotations in A .

Given an instance $P = (M, F, L_M, L_F, Q_M, Q_F)$ of the multi-partner stable marriage problem, a stable matching γ^* which minimizes the sum of dissatisfaction scores over all persons in P can be found in polynomial time by the following steps:

1. Obtain the male-optimal stable matching γ^M .
2. Apply initial pruning to P to get an R-instance $\hat{P}$.
3. Find a meta-rotation ρ exposed in $\hat{P}$ (if one exists); eliminate ρ .
4. Repeat previous step until no such ρ can be found.
5. Construct the weighted meta-rotation poset $\Psi, \leq$.
6. Identify maximum-weight closed subset A of $\Psi, \leq$.
7. Eliminate the meta-rotations of A from the R-instance obtained by initial pruning of P to get the optimal stable matching γ^* .

Let the dissatisfaction score for the male-optimal stable matching γ^M be DS_0 . The dissatisfaction score of γ^* is given by $DS_0 - \sum_{i \in 1 \dots k} w_{\rho_i}$ where $\rho_1, \rho_2, \dots, \rho_k$ are the meta-rotations eliminated during the process. Since we have chosen the maximum weight closed subset of $\Psi, \leq$; γ^* has the minimum dissatisfaction score over all stable matchings for P .

Example 6. In Example 4, $w(\rho_1) = 8$ and $w(\rho_2) = -2$. The maximum weight closed subset is $A = \{\rho_1\}$. Therefore, the ‘optimal’ stable matching is: $\gamma = \{(m_1, f_2), (m_1, f_6), (m_2, f_1), (m_2, f_2), (m_2, f_5), (m_3, f_3), (m_4, f_4), (m_4, f_3), (m_5, f_1), (m_6, f_5), (m_6, f_6), (m_6, f_4)\}$. It is obtained by eliminating ρ_1 from the R-instance obtained by initial pruning of P . $\square$

We now determine the complexity of the algorithm. Let $n = \max(|M|, |F|)$. Generating the male-optimal matching (Step 1) takes $O(n^2)$ time (Baiou and Balinski [18]). Since $\forall m \in M, q_m \leq |F|$ and $\forall f \in F, q_f \leq |M|$ the initial pruning (Step 2) requires at most $O(|M|*|F|^2 + |F|*|M|^2)$ steps which is bounded by $O(n^3)$. A meta-rotation, if it exists, can be identified and eliminated in $O(n^2)$ steps in an iteration of Step 3. Note that as pointed out earlier, a pair (m, f) can occur in only one meta-rotation. Since a meta-rotation must have at least 2 pairs and there are at most $(|M|*|F|)$ pairs to be eliminated, the number of iterations of Step 3 is bounded by $O(n^2)$. Hence Steps 3 and 4 together take $O(n^4)$ steps to eliminate all possible meta-rotations. The $\Psi, \leq$ can be constructed (Step 5) and its maximum weight closed subset found in $O(n^6)$ time (Picard [19], Rhys [20] and Picard and Queyranne [21]). The algorithm complexity is thus bounded by $O(n^6)$.

Before closing, we examine the complexity of counting the number of stable matchings for a given instance of the multiple partner stable marriage problem P . Any given matching can be checked for stability in polynomial time which implies that the enumeration problem is clearly in $\#P$. The problem instance P where $q_m = 1, \forall m \in M$ and $q_f = 1, \forall f \in F$ is also an instance of the single partner stable marriage problem which has been shown to be $\#P$ -complete (Irving and Leather [9]). Therefore, determining the number of stable matchings for an instance of the multiple partner stable marriage problem is also $\#P$ -complete.

6 Concluding Remarks

In this paper, we considered an egalitarian measure of optimality for the multiple partner stable marriage problem (with incomplete lists) and provided a polynomial time algorithm for obtaining a stable matching which satisfied the optimality criterion. By doing so, we generalized some of the results known for the corresponding one-to-one problem. The polynomial complexity is significant because the problem of determining the number of all stable matchings for the problem is $\#P$ -complete. In the process of solving the problem at hand, we proposed a novel concept of *meta-rotations* which extends the concept of *rotations* (Irving and Leather [9]) and makes it useful as a search space reduction technique for search problems.

We also showed that a useful property of the multiple partner stable marriage problem is that under a no-complementarities assumption on the preferences over combinations of partners, specifying the preference ordering over individuals alone is sufficient to ensure that the preferences of males and females turn out to be strictly ordered over all possible sets of partners that they can get in any stable matching. The results presented in this paper can accommodate weighted preferences of males and females so that the optimality criterion is not restrictive.

We note that the methodology to map stable matchings of the many-to-many problem to the antichains of a poset as well as the polynomial time algorithm to find the matching which minimizes the dissatisfaction score is generic and does not require the no-complementarities assumption. At the same time, the use of dissatisfaction scores as an egalitarian measure of optimality makes best sense when the assumption is used.

Acknowledgements. We thank Lloyd Shapley for sharing his insights on the problem. Our special thanks are due to Robert Irving for reviewing our work and providing encouraging feedback for taking it forward. We also thank David Manlove for sharing some of his current work on the stable marriage problem.

References

1. Gale, D., Shapley, L.S.: College Admissions and the Stability of Marriage. *American Mathematical Monthly*, Vol 69, (1962) 9–15
2. Polya, G., Tarjan, R.E., Woods, D.R.: *Notes on Introductory Combinatorics*. Birkhauser Verlag, Boston, Massachusetts, 1983
3. Knuth, D.E.: *Mariages Stables*. Les Presses de l'Universite de Montreal, Montreal, 1976
4. Gusfield, D., Irving, R.W.: *The Stable Marriage Problem: Structure and Algorithms*. The MIT Press, Cambridge, 1989
5. Gale, D., Sotomayor, M.: Some Remarks on the Stable Matching Problem. *Discrete Applied Mathematics*, Vol 11, (1985) 223–232
6. Iwama, K., Manlove, D., Miyazaki, S., Morita, Y.: Stable Marriage with Incomplete Lists and Ties. *Proceedings of ICALP*, (1999) 443–452
7. Valiant, L.G.: The Complexity of Computing the Permanent. *Theoretical Computer Science*, Vol 8, (1979) 189–201
8. Valiant, L.G.: The Complexity of Enumeration and Reliability problems. *SIAM Journal of Computing*, Vol 8, (1979) 410–421
9. Irving, R.W., Leather, P.: The Complexity of Counting Stable Marriages, *SIAM Journal of Computing*, Vol 15(3), (Aug 1986) 655–667
10. McVitie, D., Wilson, L.B.: The Stable Marriage Problem. *Communications of the ACM*, Vol 114, (1971) 486–492
11. Irving, R.W., Leather, P., Gusfield, D.: An Efficient Algorithm for the “Optimal” Stable Marriage. *Journal of the ACM*, Vol 34(3), (Jul 1987) 532–543
12. Roth, A. and Sotomayor, M.: *Two-sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Econometrica Society Monographs, Vol. 18, Cambridge University Press, 1990

13. Roth, A.: Stability and Polarization of Interests in Job Matching. *Econometrica*, Vol 52, (1984) 47–57
14. Sotomayor, M.: The Lattice Structure of the Set of Stable Outcomes of the Multiple Partners Assignment Game. *International Journal of Game Theory*, Vol 28, (1999) 567–583
15. Martinez, R., Masso, J., Neme, A., Oviedo, J.: An Algorithm to Compute the Set of Many-to-many Stable Matchings, UAB.IAE Working papers 436, 2001
16. Alkan, A.: On Preferences over Subsets and the Lattice Structure of Stable Matchings. *Review of Economic Design*, Vol 6, (2001) 99–111
17. Alkan, A.: A class of Multipartner Matching Markets with a Strong Lattice Structure. *Economic Theory*, Vol 19(4), (2002) 737–746
18. Baiou, M., Balinski, M.: Many-to-many Matching: Stable Polyandrous Polygamy (or Polygamous Polyandry). *Discrete Applied Mathematics*, Vol 101, (2000) 1–12
19. Picard, J.: Maximum Closure of a Graph and Applications to Combinatorial Problems. *Management Science*, Vol 22, (1976) 1268–1272
20. Rhys, J.: A Selection Problem of Shared Fixed Costs and Network Flows. *Management Science*, Vol 17, (1970) 200–207
21. Picard, J., Queyranne, M.: Selected Applications of Minimum Cuts in Networks. *INFOR - Canadian Journal of Operations Research and Information Processing*, Vol 20, (1982) 394–422

An Intersection Inequality for Discrete Distributions and Related Generation Problems^{*}

Endre Boros¹, Khaled Elbassioni¹, Vladimir Gurvich¹, Leonid Khachiyan², and Kazuhisa Makino³

¹ RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003; {boros,elbassio,gurvich}@rutcor.rutgers.edu

² Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway NJ 08854-8003; leonid@cs.rutgers.edu

³ Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, 560-8531, Japan; makino@sys.es.osaka-u.ac.jp

Abstract. Given two finite sets of points $\mathcal{X}, \mathcal{Y}$ in $\mathbb{R}^n$ which can be separated by a nonnegative linear function, and such that the componentwise minimum of any two distinct points in $\mathcal{X}$ is dominated by some point in $\mathcal{Y}$, we show that $|\mathcal{X}| \leq n|\mathcal{Y}|$. As a consequence of this result, we obtain quasi-polynomial time algorithms for generating all maximal integer feasible solutions for a given monotone system of separable inequalities, for generating all p-inefficient points of a given discrete probability distribution, and for generating all maximal empty hyper-rectangles for a given set of points in $\mathbb{R}^n$. This provides a substantial improvement over previously known exponential algorithms for these generation problems related to Integer and Stochastic Programming, and Data Mining. Furthermore, we give an incremental polynomial time generation algorithm for monotone systems with fixed number of separable inequalities, which, for the very special case of one inequality, implies that for discrete probability distributions with independent coordinates, both p-efficient and p-inefficient points can be separately generated in incremental polynomial time.

1 Introduction

Let $\mathcal{X}$ and $\mathcal{Y}$ be two finite sets of points in $\mathbb{R}^n$ such that

(P1) $\mathcal{X}$ and $\mathcal{Y}$ can be separated by a nonnegative linear function: $w(x) > t \geq w(y)$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, where $w(x) = \sum_{i=1}^n w_i x_i$, $w_1, \dots, w_n \in \mathbb{R}_+$ are nonnegative weights, and $t \in \mathbb{R}$ is a real threshold,

^{*} The research of the first four authors was supported in part by the National Science Foundation Grant IIS-0118635. The research of the first and third authors was also supported in part by the Office of Naval Research Grant N00014-92-J-1375. The second and third authors are also grateful for the partial support by DIMACS, the National Science Foundation's Center for Discrete Mathematics and Theoretical Computer Science. The fifth author was supported in part by the Scientific Grant in Aid of the Ministry of Education, Science, Sports, Culture and Technology of Japan.

(P2) For any two distinct points $x, x' \in \mathcal{X}$, their componentwise minimum $x \wedge x'$ is dominated by some $y \in \mathcal{Y}$, i.e. $x \wedge x' \leq y$.

Given $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^n$ satisfying properties (P1) and (P2), one may ask the question of how large the size of $\mathcal{X}$ can be in terms of the size of $\mathcal{Y}$. For instance, if $\mathcal{X}$ is the set of the n -dimensional unit vectors, and $\mathcal{Y} = \{\mathbf{0}\}$ is the set containing only the origin, then $\mathcal{X}$ and $\mathcal{Y}$ satisfy properties (P1), (P2), and the ratio between their cardinalities is n . We shall show that this is actually an extremal case:

Lemma 1 (Intersection Lemma). *If $\mathcal{X}$ and $\mathcal{Y} \neq \emptyset$ are two finite sets of points in $\mathbb{R}^n$ satisfying properties (P1) and (P2) above, then*

$$|\mathcal{X}| \leq n|\mathcal{Y}|. \quad (1)$$

An analogous statement for binary sets $\mathcal{X}, \mathcal{Y} \subseteq \{0, 1\}^n$ was shown in [6]. Let us also recall from [6] that condition (P1) is important, since without that $|\mathcal{X}|$ could be exponentially larger than $|\mathcal{Y}|$, already in the binary case.

Let us also remark that the nonnegativity of the weight vector w is also important. Consider for instance $\mathcal{Y} = \{(1, 1, \dots, 1)\}$ and an arbitrary number of points in the set $\mathcal{X}$ such that $0 \leq x_i < 1$ for all $x \in \mathcal{X}$ and $i = 1, \dots, n$. Then clearly (P2) holds, and (P1) is satisfied with $w = (-1, 0, \dots, 0)$ and $t = -1$. However, it is impossible to bound the cardinality of $\mathcal{X}$ in terms of n and $|\mathcal{Y}| = 1$.

Let us further note that, due to the strict separation in (P1), we may assume without loss of generality that all weights are positive $w > 0$. In fact, it would be even enough to prove the lemma with $w = (1, 1, \dots, 1)$, since scaling the i th coordinates of all points in $\mathcal{X} \cup \mathcal{Y}$ by $w_i \geq 0$ for $i = 1, \dots, n$ always transforms the input into one satisfying (P1) with $w = (1, 1, \dots, 1)$. Clearly, such scaling preserves the relative order of the i th coordinates of the points, and scales properly their componentwise minimum, thus the transformed point sets will satisfy (P2) as well.

As a consequence of the above lemma, we obtain new results on the complexity of several generation problems, including:

Monotone systems of separable inequalities: Given a system of inequalities on sums of single-variable monotone functions, generate all maximal feasible integer solutions of the system.

p-Efficient and p-inefficient points of discrete probability distributions: Given a random variable $\xi \in \mathbb{Z}^n$, generate all p -inefficient points, i.e. maximal vectors $x \in \mathbb{Z}^n$ whose cumulative probability $\Pr[\xi \leq x]$ does not exceed a certain threshold p , and/or generate all p -efficient points, i.e. minimal vectors $x \in \mathbb{Z}^n$ for which $\Pr[\xi \leq x] \geq p$. This problem has applications in Stochastic Programming [10,22].

Maximal k-boxes: Given a set of points in $\mathbb{R}^n$ and a nonnegative integer k , generate all maximal n -dimensional intervals (*boxes*), which contain at most k of the given points in their interior. Such intervals are called empty boxes or empty rectangles, when $k = 0$. This problem has applications in computational geometry, data mining and machine learning [1,2,8,11,16,17,20,21].

These problems are described in more details in the following sections. What they have in common is that each can be modelled by a property π over a set of vectors $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_n$, where $\mathcal{C}_i, i = 1, \dots, n$ are finite subsets of the reals, and π is anti-monotone, i.e. if $x, y \in \mathcal{C}$, $x \geq y$, and x satisfies property π , then y also satisfies π . Each problem in turn can be stated as of incrementally generating the family $\mathcal{F}_\pi$ of all *maximal* elements of $\mathcal{C}$ satisfying an anti-monotone property π :

GEN($\mathcal{F}_\pi, \mathcal{E}$): *Given an anti-monotone property π , and a subfamily $\mathcal{E} \subseteq \mathcal{F}_\pi$ of the maximal elements satisfying π , either find a new maximal element $x \in \mathcal{F}_\pi \setminus \mathcal{E}$, or prove that $\mathcal{E} = \mathcal{F}_\pi$.*

Clearly, the entire family $\mathcal{F}_\pi$ can be generated by initializing $\mathcal{E} = \emptyset$ and iteratively solving the above problem $|\mathcal{F}_\pi| + 1$ times.

For a subset $\mathcal{A} \subseteq \mathcal{C}$, denote by $\mathcal{I}(\mathcal{A})$ the set of maximal independent elements of $\mathcal{A}$, i.e. the set of those elements $x \in \mathcal{C}$ that are maximal with respect to the property that $x \not\geq a$ for all $a \in \mathcal{A}$. Then $\mathcal{I}^{-1}(\mathcal{A})$ is the set of elements $x \in \mathcal{C}$ that are minimal with the property that $x \not\leq a$ for all $a \in \mathcal{A}$. In particular, $\mathcal{I}^{-1}(\mathcal{F}_\pi)$ denotes the family of *minimal* elements of $\mathcal{C}$ which *do not* satisfy property π .

Following [6], let us call $\mathcal{F}_\pi$ *uniformly dual-bounded*, if for every subfamily $\mathcal{E} \subseteq \mathcal{F}_\pi$ we have

$$|\mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_\pi)| \leq p(|\pi|, n, |\mathcal{E}|) \quad (2)$$

for some polynomial $p(\cdot)$, where $|\pi|$ denotes the length of the description of property π . It is known that for uniformly dual-bounded families $\mathcal{F}_\pi$ of subsets of a discrete box $\mathcal{C}$ problem GEN($\mathcal{F}_\pi, \mathcal{E}$) can be reduced in polynomial time to the following *dualization* problem on boxes (see [4] and also [3,13]):

DUAL($\mathcal{C}, \mathcal{A}, \mathcal{B}$): *Given an integer box $\mathcal{C}$, a family of vectors $\mathcal{A} \subseteq \mathcal{C}$ and a subset $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$ of its maximal independent vectors, either find a new maximal independent vector $x \in \mathcal{I}(\mathcal{A}) \setminus \mathcal{B}$, or prove that no such vector exists, i.e., $\mathcal{B} = \mathcal{I}(\mathcal{A})$.*

It is furthermore known that problem DUAL($\mathcal{C}, \mathcal{A}, \mathcal{B}$) can be solved in $\text{poly}(n) + m^{o(\log m)}$ time, where $m = |\mathcal{A}| + |\mathcal{B}|$ (see [4,12]). However, it is still open whether DUAL($\mathcal{C}, \mathcal{A}, \mathcal{B}$) has a polynomial time algorithm (e.g., [3,12,19]).

For each of the problems described above, it will be shown that the families $\mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_\pi)$ and $\mathcal{E} \subseteq \mathcal{F}_\pi$ are, respectively, in one to one correspondence with two sets of points $\mathcal{X}, \mathcal{Y}$ satisfying the conditions of Lemma 1. Thus, by Lemma 1 we can derive (2), which in its turn is sufficient for the efficient generation of the family $\mathcal{F}_\pi$ (see [4]).

In particular, it will follow that each of the above generation problems can be solved incrementally in *quasi-polynomial time*. Furthermore, we give incremental *polynomial-time* algorithms for generating

- all maximal feasible, and separately, all minimal infeasible integer vectors for systems with fixed number of monotone separable inequalities, and
- all p -efficient, and separately, all p -inefficient points of discrete probability distributions with independent coordinates

2 Systems of Monotone Separable Inequalities

For $i = 1, 2, \dots, n$, let l_i, u_i be given integers, $l_i \leq u_i$, and let $\mathcal{C}_i \stackrel{\text{def}}{=} \{l_i, l_i + 1, \dots, u_i\}$. A function $f : \mathcal{C}_i \mapsto \mathbb{R}$ is called *monotone* if, for $x, y \in \mathcal{C}_i$, $f(x) \geq f(y)$ whenever $x \geq y$. Let $f_{ij} : \mathcal{C}_i \mapsto \mathbb{R}$, $i = 1, 2, \dots, n$, $j = 1, \dots, r$ be polynomially-computable monotone functions, and consider the system of inequalities

$$\sum_{i=1}^n f_{ij}(x_i) \leq t_j, \quad j = 1, \dots, r, \quad (3)$$

over the elements $x \in \mathcal{C} = \{x \in \mathbb{Z}^n \mid l \leq x \leq u\}$, where $l = (l_1, \dots, l_n)$, $u = (u_1, \dots, u_n)$, and $t = (t_1, \dots, t_r)$ is a given r -dimensional real vector.

Let us denote by $\mathcal{F}_t$ the set of maximal feasible solutions for (3), and thus $\mathcal{I}^{-1}(\mathcal{F}_t)$ represents the set of minimal infeasible vectors for (3).

Generalizing results on monotone systems of *linear* inequalities from [4], we will now use Lemma 1 to prove the following:

Theorem 1. *If $\mathcal{F}_t$ is the family of maximal feasible solutions of (3), and $\mathcal{E} \subseteq \mathcal{F}_t$ is non-empty, then*

$$|\mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_t)| \leq rn|\mathcal{E}|. \quad (4)$$

In particular, $|\mathcal{I}^{-1}(\mathcal{F}_t)| \leq rn|\mathcal{F}_t|$.

Proof. Let us define a monotonic mapping $\phi : \mathcal{C} \mapsto \mathbb{R}^n$ by setting $\phi(x) = (f_{1j}(x_1), \dots, f_{nj}(x_n))$ for $x \in \mathcal{C}$. Let $\mathcal{Y} \stackrel{\text{def}}{=} \{\phi(x) \mid x \in \mathcal{E}\}$, and let $\mathcal{X}_j \stackrel{\text{def}}{=} \{\phi(x) \mid x \in \mathcal{I}^{-1}(\mathcal{E}), \sum_{i=1}^n f_{ij}(x_i) > t_j\}$, for $j = 1, \dots, r$. In other words, $\mathcal{X}_j$ is the ϕ -mapping of those minimal infeasible solutions of (3) in $\mathcal{I}^{-1}(\mathcal{E})$ which violate the j th inequality. Since the functions f_{ij} are monotone, and since we consider only maximal feasible or minimal infeasible vectors for (3), the mappings $\mathcal{E} \rightarrow \mathcal{Y}$ and $\mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_t) \rightarrow \mathcal{X}_1 \cup \dots \cup \mathcal{X}_r$ are one-to-one.

It is also easy to see that the sets $\mathcal{X}_j$ and $\mathcal{Y}$ satisfy the conditions of Lemma 1 with $w = (1, 1, \dots, 1)$ and $t = t_j$, for $j = 1, \dots, r$, and thus (4) follows readily by Lemma 1. $\square$

Since by (4) the family $\mathcal{F}_t$ is uniformly dual-bounded, the results of [4], as we cited earlier, directly imply the following.

Corollary 1. *Problem $\text{GEN}(\mathcal{F}_t, \mathcal{X})$ of incrementally generating maximal feasible solutions for (3) can be solved in $k^{o(\log k)}$ time, where $k = \max\{n, r, |\mathcal{X}|\}$, $\log(\|u - l\|_\infty + 1)$.*

It should be mentioned that in contrast to (4), the size of $\mathcal{F}_t$ cannot be bounded by a polynomial in n , r , and $|\mathcal{I}^{-1}(\mathcal{F}_t)|$, even for monotone systems of linear inequalities. However, for systems (3) with constant r , we shall show that such a bound exists, and further that the generation problem can be solved in polynomial time:

Theorem 2. *If $\mathcal{F}_t$ is the family of maximal feasible solutions of (3), and $\mathcal{E} \subseteq \mathcal{I}^{-1}(\mathcal{F}_t)$ is non-empty, then*

$$|\mathcal{I}(\mathcal{E}) \cap \mathcal{F}_t| \leq (n|\mathcal{E}|)^r. \quad (5)$$

In particular, $|\mathcal{F}_t| \leq \left(n|\mathcal{I}^{-1}(\mathcal{F}_t)|\right)^r$.

Theorem 3. *If the number of inequalities in (3) is bounded, then both the maximal feasible and minimal infeasible vectors can be generated in incremental polynomial time, in n , r and $\sum_{i=1}^n |\mathcal{C}_i|$.*

The proofs of Theorem 2 and 3 will be given in Section 6. In the next section, we consider an application of Theorem 3 for the case of $r = 1$.

3 p -Efficient and p -Inefficient Points of Probability Distributions

Let ξ be an n -dimensional random variable on $\mathbb{Z}^n$, with a finite support $\mathcal{S} \subseteq \mathbb{Z}^n$, i.e., $\sum_{q \in \mathcal{S}} \Pr[\xi = q] = 1$, and $\Pr[\xi = q] > 0$ for $q \in \mathcal{S}$. Given a threshold probability $p \in [0, 1]$, a point $x \in \mathbb{Z}^n$ is said to be p -efficient if it is minimal with the property that $\Pr[\xi \leq x] > p$. Let us conversely say that $x \in \mathbb{Z}^n$ is p -inefficient if it is maximal with the property that $\Pr[\xi \leq x] \leq p$. Denote respectively by $\mathcal{F}_{\mathcal{S},p}$, $\mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},p})$ the sets of p -inefficient, and p -efficient points for ξ . Clearly, these sets are finite since, in each dimension $i \in [n] \stackrel{\text{def}}{=} \{1, \dots, n\}$, we need to consider only the projections $\mathcal{C}_i \stackrel{\text{def}}{=} \{q_i, q_i - 1 \mid q \in \mathcal{S}\} \subseteq \mathbb{Z}$. In other words, the sets $\mathcal{F}_{\mathcal{S},p}$ and $\mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},p})$ can be regarded as subsets of a finite integral box $\mathcal{C} = \mathcal{C}_1 \times \dots \times \mathcal{C}_n$ of size at most $2|\mathcal{S}|$ along each dimension.

Theorem 4. *Given a partial list $\mathcal{E} \subseteq \mathcal{F}_{\mathcal{S},p}$ of p -inefficient points, problem $\text{GEN}(\mathcal{F}_{\mathcal{S},p}, \mathcal{E})$ can be solved in $k^{o(\log k)}$ time, where $k \stackrel{\text{def}}{=} \max\{n, |\mathcal{S}|, |\mathcal{E}|\}$.*

Proof. This statement is again a consequence of the fact that the set $\mathcal{F}_{\mathcal{S},p}$ is uniformly dual-bounded, i.e. that

$$|\mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},p})| \leq |\mathcal{S}||\mathcal{E}|, \quad (6)$$

for any non-empty subset $\mathcal{E} \subseteq \mathcal{F}_{\mathcal{S},p}$. To see (6), let $\mathcal{X} = \{\phi(x) \mid x \in \mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},p})\}$ and $\mathcal{Y} = \{\phi(y) \mid y \in \mathcal{E}\}$, where $\phi : \mathbb{Z}^n \mapsto \mathbb{R}^{|\mathcal{S}|}$ is the mapping defined by: $\phi(x) = (\Pr[\xi = q] : q \in \mathcal{S}, q \leq x)$ for $x \in \mathbb{Z}^n$. One can easily check that the mapping ϕ is one-to-one between $\mathcal{X}$ and $\mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},p})$, and that the families $\mathcal{X}$ and $\mathcal{Y}$ satisfy properties (P1) and (P2) with $w = (1, 1, \dots, 1)$ and $t = p$. Therefore, (6) follows from the intersection lemma. $\square$

In particular, all p -inefficient points of a discrete probability distribution can be enumerated incrementally in quasi-polynomial time. In general, a result analogous to that for p -efficient points is highly unlikely to hold, as there exist examples for which the corresponding problem is NP-hard:

Proposition 1. *Given a discrete random variable ξ on a finite support set $\mathcal{S} \subseteq \mathbb{R}^n$, a threshold probability $p \in [0, 1]$, and a partial list $\mathcal{E} \subseteq \mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},p})$ of p -efficient points for ξ , it is NP-complete to decide if $\mathcal{E} \neq \mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},p})$.*

Proof. Consider the well-known NP-complete problem of deciding whether a given graph $G = (V, E)$ contains an independent set of size t , where $t \geq 2$ is a given threshold. Let $\mathcal{S} \subseteq \{0, 1\}^V$ be the set of points consisting of the $|V|$ incidence vectors of the vertices of G , and $t-2$ copies of the $|E|$ incidence vectors of the edges. Let ξ be an n -dimensional integer-valued random variable having uniform distribution on $\mathcal{S}$, i.e., $\Pr[\xi = q] = 1/|\mathcal{S}|$ if and only if $q \in \mathcal{S}$. Then, for $p = t/|\mathcal{S}|$, the incidence vector of each edge is a p -efficient point for ξ , and it is easy to see that there is another p -efficient point if and only if there is an independent set of G of size t . $\square$

Finally we observe that if ξ is an integer-valued finite random variable with *independent* coordinates $\xi_1, \dots, \xi_n$, then the generation of both $\mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},p})$ and $\mathcal{F}_{\mathcal{S},p}$ can be done in polynomial time, even if the number of points $\mathcal{S}$, defining the distribution of ξ , is *exponential* in n (but provided that the distribution function for each component ξ_i is computable in polynomial-time). Indeed, by independence we have $\Pr[\xi \leq x] = \prod_{j=1}^n \Pr[\xi_j \leq x_j]$. Defining $f(x) = \log \Pr[\xi \leq x] = \sum_{j=1}^n \log \Pr[\xi_j \leq x_j]$, we can write $f(x)$ as the sum of single-variable monotone functions $f_1, \dots, f_n$, where $f_i = \log \Pr[\xi_i \leq x_i]$, for $i = 1, \dots, n$. Let $l_i = \min\{x_i \in \mathbb{Z} \mid \Pr[\xi_i \leq x_i] > 0\} - 1$, $u_i = \min\{x_i \in \mathbb{Z} \mid \Pr[\xi_i \leq x_i] = 1\}$, and $C_i = \{z \in \mathbb{Z} \mid l_i \leq z \leq u_i\}$. Then the p -inefficient (p -efficient) points are the maximal feasible (respectively, minimal infeasible) solutions of the *monotone separable* inequality $\sum_{i=1}^n f_i(x_i) \leq t \stackrel{\text{def}}{=} \log p$ over the product $\mathcal{C} \stackrel{\text{def}}{=} C_1 \times \dots \times C_n$. Consequently, Theorem 3 immediately gives the following:

Corollary 2. *If the coordinates of a random variable ξ over $\mathbb{Z}^n$ are independent, then both the p -efficient and the p -inefficient points for ξ can be enumerated in incremental polynomial time.*

4 Maximal k -Boxes

Let $\mathcal{S}$ be a set of points in $\mathbb{R}^n$, and k be a given integer, $k \leq |\mathcal{S}|$. A *maximal k -box* is an n -dimensional interval which does not contain more than k points of $\mathcal{S}$ in its interior, and which is maximal with respect to this property (i.e. cannot be extended in any direction without strictly enclosing more points of $\mathcal{S}$). Let $\mathcal{F}_{\mathcal{S},k}$ be the set of all maximal k -boxes. The problem of generating all elements of $\mathcal{F}_{\mathcal{S},k}$ has been studied in the machine learning and computational geometry literatures (see [2,8,11,20,21]), and is motivated by the discovery of missing associations or “holes” in data mining applications (see [1,16,17]). All

known algorithms that solve this problem have running time complexity which is exponential in the dimension n of the given point set. In contrast, we show in this paper that the problem can be solved in quasi-polynomial time:

Theorem 5. *Given a point set $\mathcal{S} \subseteq \mathbb{R}^n$, an integer k , and a partial list of maximal empty boxes $\mathcal{E} \subseteq \mathcal{F}_{\mathcal{S},k}$, problem $\text{GEN}(\mathcal{F}_{\mathcal{S},k}, \mathcal{E})$ can be solved in $m^{o(\log m)}$ time, where $m \stackrel{\text{def}}{=} \max\{n, |\mathcal{S}|, |\mathcal{E}|\}$.*

Proof. Let us define $\mathcal{C}_i \stackrel{\text{def}}{=} \{p_i - \epsilon, p_i, p_i + \epsilon \mid p \in \mathcal{S}\}$ for $i = 1, \dots, n$, where $\epsilon > 0$ is small enough, and let us consider the family of boxes $\mathcal{B} = \{[a, b] \subseteq \mathbb{R}^n \mid a, b \in \mathcal{C}_1 \times \dots \times \mathcal{C}_n, a \leq b\}$. Then $\mathcal{F}_{\mathcal{S},k} \subseteq \mathcal{B}$, and $\mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},k})$ corresponds to minimal boxes of $\mathcal{B}$ containing at least $k+1$ points of $\mathcal{S}$ in their interior. Then, to prove the theorem it is enough to show that, for any non-empty subset $\emptyset \neq \mathcal{E} \subseteq \mathcal{F}_{\mathcal{S},k}$, we have

$$|\mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},k})| \leq |\mathcal{S}| |\mathcal{E}|. \quad (7)$$

Let us note first that for $k = 0$ we have $|\mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},0})| = |\mathcal{S}|$, implying (7) readily, thus we assume $k > 0$ in the sequel. Let $u = (u_1, \dots, u_n)$ where $u_i = \max \mathcal{C}_i$ for $i = 1, \dots, n$, let $\mathcal{C}_i^* \stackrel{\text{def}}{=} \{u_i - p \mid p \in \mathcal{C}_i\}$ for $i = 1, \dots, n$, and let us consider the $2n$ -dimensional box $\mathcal{C} = \mathcal{C}_1^* \times \dots \times \mathcal{C}_n^* \times \mathcal{C}_1 \times \dots \times \mathcal{C}_n$. Let us further represent every n -dimensional interval $[a, b]$ in $\mathcal{F}_{\mathcal{S},k} \cup \mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},k})$ as a $2n$ -dimensional vector $(u - a, b) \in \mathcal{C}$. It is now easy to see that if $x, y \in \mathcal{C}$ are two boxes, $x \leq y$ (componentwise, as usual), and x defines a box, then indeed y also defines a box which contains x (though not all elements of $\mathcal{C}$ define a box, since $a_i > b_i$ is possible for some $(u - a, b) \in \mathcal{C}$).

Let us now define the anti-monotone property π to be satisfied by an $x \in \mathcal{C}$ if and only if it contains at most k points in its interior, where x contains no point in its interior if it does not define a box. Clearly, $\mathcal{F}_\pi$ for this property and $\mathcal{F}_{\mathcal{S},k}$ differ by at most $\sum_{i=1}^n |\mathcal{C}_i| - 1$ elements, in which $a_i > b_i$ for exactly one of the indices i , and the values a_i and b_i are consecutive in $\mathcal{C}_i$.

Finally, consider the sets $\mathcal{X} = \{\phi(x) \mid x \in \mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},k})\}$ and $\mathcal{Y} = \{\phi(y) \mid y \in \mathcal{E}\}$, where $\phi(x) \in \{0, 1\}^{\mathcal{S}}$ is the characteristic vector of the subset of $\mathcal{S}$ contained in the interior of box $x \in \mathcal{C}$. It is easy to see now that the mapping ϕ is one-to-one between $\mathcal{X}$ and $\mathcal{I}^{-1}(\mathcal{E}) \cap \mathcal{I}^{-1}(\mathcal{F}_{\mathcal{S},k})$, and that the sets $\mathcal{X}$ and $\mathcal{Y}$ satisfy properties (P1) and (P2) with $w = (1, 1, \dots, 1)$ and $t = k$. Thus, inequality (7) follows by applying the intersection lemma. $\square$

5 Proof of the Intersection Lemma

As mentioned in the introduction, we may assume without loss of generality that all the weights are 1's. We can further assume that $\mathcal{Y}$ is a minimal family for properties (P1) and (P2). For $i = 1, \dots, n$, let $l_i \stackrel{\text{def}}{=} \min\{x_i \mid x \in \mathcal{X}\}$, and $u_i \stackrel{\text{def}}{=} \max\{x_i \mid x \in \mathcal{X}\}$.

To prove the lemma, we shall show by induction on $|\mathcal{X}|$ that

$$|\mathcal{X}| \leq \sum_{y \in \mathcal{Y}} q(y), \quad (8)$$

where $q(y)$ is the number of components y_i such that $y_i < u_i$.

Clearly, for $|\mathcal{X}| \leq 1$ the statement is true since $\mathcal{Y}$ is non-empty and $q(y) = 0$ for $y \in \mathcal{Y}$ implies by (P1) that $\mathcal{X} = \emptyset$. Let us assume therefore that $|\mathcal{X}| \geq 2$, and define for every $i = 1, \dots, n$ and $z \in \mathbb{R}$ the families

$$\mathcal{X}(i, z) = \{x \in \mathcal{X} \mid x_i \geq z\}, \quad \mathcal{Y}(i, z) = \{y \in \mathcal{Y} \mid y_i \geq z\}.$$

Clearly, these families satisfy conditions (P1) and (P2) and therefore satisfy the conclusion of the lemma whenever $\mathcal{Y}(i, z) \neq \emptyset$. Furthermore, we may assume without loss of generality that $\mathcal{Y}(i, z) = \emptyset$ implies $\mathcal{X}(i, z) = \emptyset$ for all $i \in [n]$ and $z \in \mathbb{R}$. Indeed, by (P2), if $|\mathcal{Y}(i, z)| = 0$ then $|\mathcal{X}(i, z)| \in \{0, 1\}$. If there is an $i \in [n]$ and $z \in \mathbb{R}$, such that $\mathcal{X}(i, z) = \{x\}$ and $\mathcal{Y}(i, z) = \emptyset$, then deleting the element x from $\mathcal{X}$ reduces $|\mathcal{X}|$ by 1 and reduces the sum $\sum_{y \in \mathcal{Y}} q(y)$ by at least 1.

Thus, we can assume by induction on the number of elements in $\mathcal{X}$ that

$$|\mathcal{X}(i, z)| \leq \sum_{y \in \mathcal{Y}(i, z)} q(y) \quad (9)$$

whenever

$$|\mathcal{X}(i, z)| < |\mathcal{X}|.$$

Let us now sum up inequalities (9), for all indices $i \in [n]$ and for all values $z > l_i$ (for which $|\mathcal{X}(i, z)| \neq |\mathcal{X}|$), yielding

$$\sum_{i=1}^n \int_{z>l_i} |\mathcal{X}(i, z)| dz \leq \sum_{i=1}^n \int_{z>l_i} \sum_{y \in \mathcal{Y}(i, z)} q(y) dz. \quad (10)$$

It is easily seen that the left hand side of (10) is equal to

$$L = \sum_{x \in \mathcal{X}} \sum_{i=1}^n (x_i - l_i) |\mathcal{X}|,$$

while the right hand side is equal to

$$R = \sum_{y \in \mathcal{Y}} q(y) \sum_{i=1}^n (y_i - l_i).$$

Thus, we get by (P1) and (10) that

$$(t - \sum_{i=1}^n l_i) |\mathcal{X}| < L \leq R \leq (t - \sum_{i=1}^n l_i) \sum_{y \in \mathcal{Y}} q(y). \quad (11)$$

Note that $t - \sum_{i=1}^n l_i > 0$ can be assumed without loss of generality. Indeed, if $t \leq \sum_{i=1}^n l_i$ then for an arbitrary $y \in \mathcal{Y}$ (and $\mathcal{Y} \neq \emptyset$) we have $\sum_{i=1}^n y_i \leq t \leq \sum_{i=1}^n l_i$ by (P1). By the minimality of $\mathcal{Y}$, we must have $y_i \geq l_i$, for all $i = 1, \dots, n$, implying that $t = \sum_{i=1}^n l_i$. But then we can replace t by $t + \epsilon$, for a sufficiently small $\epsilon > 0$, and still satisfy property (P1). Thus inequality (8) follows from (11). $\square$

Remark 1. Lemma 1 can be generalized as follows. Given two finite sets of points $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^n$ and an integer $r \geq 2$, such that $\mathcal{X}$ and $\mathcal{Y}$ can be separated by a nonnegative linear function and for any r distinct points $x^1, x^2, \dots, x^r \in \mathcal{X}$, their componentwise minimum $x^1 \wedge x^2 \wedge \dots \wedge x^r$ is dominated by some $y \in \mathcal{Y}$ (i.e. $x^1 \wedge x^2 \wedge \dots \wedge x^r \leq y$), then

$$|\mathcal{X}| \leq n(r-1)|\mathcal{Y}|.$$

6 Proof of Theorems 2 and 3

For $j = 1, 2, \dots, r$, let $f_j(x) = \sum_{i=1}^n f_{ij}(x_i)$, where $x \in \mathcal{C} = \{x \in \mathbb{Z}^n \mid l_i \leq x_i \leq u_i, i = 1, 2, \dots, n\}$. For a given real vector $t = (t_1, \dots, t_r)$, let $\mathcal{F}_t$ be the set of maximal feasible solutions of system (3).

For each $i \in [n] \stackrel{\text{def}}{=} \{1, \dots, n\}$, let $\Delta_{ij} : \{l_i - 1, l_i, \dots, u_i\} \rightarrow \mathbb{R}$ be the difference of f_{ij} defined by

$$\Delta_{ij}(x_i) = \begin{cases} f_{ij}(x_i + 1) - f_{ij}(x_i) & \text{if } x_i \in \{l_i, l_i + 1, \dots, u_i - 1\} \\ +\infty & \text{if } x_i \in \{l_i - 1, u_i\}. \end{cases} \quad (12)$$

Let us now define, for each $j \in [r]$, a mapping μ^j from pairs of a vector $x \in \mathcal{C}$ and a component $i \in [n]$ with $x_i > l_i$ to vectors $y \in \mathcal{C}$ by

$$\mu^j(x, i)_k = \begin{cases} x_k - 1 & \text{if } k = i \\ x_k + \alpha_k & \text{otherwise,} \end{cases} \quad (13)$$

where $\alpha_k = \alpha_k(x, k, j)$ is a non-negative integer such that $\Delta_{kj}(x_k + \alpha_k) \geq \Delta_{ij}(x_i - 1)$ and $\Delta_{kj}(x_k + s) < \Delta_{ij}(x_i - 1)$ for all $s = 0, 1, \dots, \alpha_k - 1$. Note that such α_k always exists by our definition (12).

Given any $x \in \mathcal{I}^{-1}(\mathcal{F}_t)$, there exists an index $j = \rho(x) \in [r]$ such that x violates the j th inequality of the system, i.e. $f_j(x) > t_j$. For $\mathcal{E} \subseteq \mathcal{I}^{-1}(\mathcal{F}_t)$ and $j \in [r]$, let $\rho_{\mathcal{E}}^{-1}(j) \stackrel{\text{def}}{=} \{x \in \mathcal{E} \mid \rho(x) = j\}$.

Proof of Theorem 2 Let us consider an arbitrary non-empty subset $\mathcal{E} \subseteq \mathcal{I}^{-1}(\mathcal{F}_t)$. Consider a vector $y \in \mathcal{I}(\mathcal{E}) \cap \mathcal{F}_t$ and let y_i be a component of y such that $y_i < u_i$ (such a component always exists since $\mathcal{E}$ is non-empty). Then, by the maximality of y , there exists a vector $x = x^i \in \mathcal{E}$ such that $x \leq y + \mathbf{e}^i$, where $\mathbf{e}^i$ is the i th unit vector. Let $j = \rho(x) \in [r]$ be an index such that x violates the j th inequality of the system.

Claim 1. $y \leq \mu^j(x, i)$.

Proof. Let us first note that $x_i = y_i + 1$, since $x_i \leq y_i + 1$ and we have $f_j(x) \leq t_j$ if $x_i \leq y_i$, contradicting the fact that $x \in \mathcal{I}^{-1}(\mathcal{F}_t)$. This means $y_i = \mu^j(x, i)_i$. Moreover, if $x_k < y_k - \alpha_k$ for some $k \neq i$, then we have

$$\begin{aligned} f_j(y) - f_j(x) &= \sum_{h \neq i, k} (f_{hj}(y_h) - f_{hj}(x_h)) \\ &\quad + (f_{kj}(y_k) - f_{kj}(x_k)) - (f_{ij}(x_i) - f_{ij}(y_i)) \\ &\geq \Delta_{kj}(x_k + \alpha_k) - \Delta_{ij}(x_i - 1), \end{aligned} \quad (14)$$

where the last inequality follows from the monotonicity of the functions f_{ij} , and the facts that $x_k \leq y_k$ for all $k \neq i$, $y_i = x_i - 1$, and $y_k \geq x_k + \alpha_k + 1$. Since $\Delta_{kj}(x_k + \alpha_k) - \Delta_{ij}(x_i - 1) \geq 0$ by the definition of $\alpha_k = \alpha_k(x, k, j)$, we get $f_j(y) \geq f_j(x) > t_j$, a contradiction to the fact that $y \in \mathcal{F}_t$. Therefore, $y_k \leq x_k + \alpha_k$ must hold for all components $k \neq i$, proving the calim. $\square$

Claim 2. $y_k = \mu^j(x, i)_k$ for all components $k \in [n]$ for which

$$\Delta_{kj}(y_k) \geq \Delta_{ij}(y_i). \quad (15)$$

Proof. Let $k \neq i$ satisfy (15), then for $s = 0, 1, \dots, \alpha_j - 1$, we have

$$\Delta_{kj}(y_k) \geq \Delta_{ij}(y_i) = \Delta_{ij}(x_i - 1) > \Delta_{kj}(x_k + s), \quad (16)$$

by definition of $\alpha_k = \alpha_k(x, k, j)$. Since $x_k \leq y_k$, it follows from (16) that $y_k \geq x_k + \alpha_k = \mu^j(x, i)_k$, and therefore the result follows from Claim 1. $\square$

Claims 1 and 2 imply that

$$y = \bigwedge_{i \in [n]: y_i < u_i} \mu^j(x^i, i), \quad (17)$$

where for vectors $v, u \in \mathcal{C}$ we let, as before, $v \wedge u$ denote the component-wise minimum of v and u .

Not all of the vectors $\mu^j(x^i, i)$ are necessary for this representation. Suppose that there exist two vectors $x^i, x^k \in \mathcal{E}$ such that $x^i_i > l_i$, $x^k_k > l_k$, $x^i \leq y + \mathbf{e}^i$, $x^k \leq y + \mathbf{e}^k$, and $\rho(x^i) = \rho(x^k) = j$. Suppose further that $\Delta_{kj}(x^k_k - 1) \leq \Delta_{ij}(x^i_i - 1)$. Then Claim 2 implies that (17) remains valid even if we drop x^k . In other words, we can identify, for each $j \in [r]$, a single vector $x^{i_j} \in \rho_{\mathcal{E}}^{-1}(j)$, and obtain consequently at most r vectors $\mu^j(x^{i_j}, i_j)$ such that

$$y = \bigwedge_{j \in [r]} \mu^j(x^{i_j}, i_j), \quad (18)$$

where we have $\mu^j(x^{i_j}, i_j) = u$ if there exists no vector x^i in $\rho_{\mathcal{E}}^{-1}(j)$. The latter representation readily implies (5). $\square$

Proof of Theorem 3 Note that for constant r , the sizes of $\mathcal{F}_t$ and $\mathcal{I}^{-1}(\mathcal{F}_t)$ are polynomially related by inequalities (4) and (5). Hence, the theorem follows

from the following lemma which gives an algorithm for generating all minimal true points and/or all maximal false points of a monotone separable system (3), with bounded number of inequalities r , in incremental polynomial time.

For $\mathcal{E} \subseteq \mathcal{C}$, denote by $\mathcal{E}^+ = \{y \in \mathcal{C} \mid y \geq x, \text{ for some } x \in \mathcal{E}\}$ and $\mathcal{E}^- = \{y \in \mathcal{C} \mid y \leq x, \text{ for some } x \in \mathcal{E}\}$.

Lemma 2. *Let $\mathcal{F}_t$ be the set of maximal feasible solutions for (3), and let $\mathcal{Y} \subseteq \mathcal{F}_t$ and $\mathcal{X} \subseteq \mathcal{I}^{-1}(\mathcal{F}_t)$, such that $\mathcal{X} \neq \emptyset$. Then $\mathcal{Y} = \mathcal{F}_t$ and $\mathcal{X} = \mathcal{I}^{-1}(\mathcal{F}_t)$ if and only if*

- (i) *For all $x \in \mathcal{X}$ and $i \in [n]$ such that $x_i > l_i$, and for all $k \neq i$ such that $\mu^j(x, i)_k < u_k$, where $j = \rho(x)$, the vector $\bar{x} = \bar{x}(x, i, k)$ given by*

$$\bar{x}_h = \begin{cases} x_h - 1 & \text{if } h = i \\ \mu^j(x, i)_h + 1 & \text{if } h = k \\ x_h & \text{otherwise,} \end{cases} \quad (19)$$

is in $\mathcal{X}^+$.

- (ii) *For every collection $(x^j \in \rho_{\mathcal{X}}^{-1}(j) \mid j \in [r])$, and for every selection of indices $(k_1, \dots, k_r)$ such that $x_{k_j}^j > l_{k_j}$, the vector $y = \bigwedge_{j \in [r]} \nu^j$ is in $\mathcal{X}^+ \cup \mathcal{Y}^-$, where ν_j is either $\mu^j(x^j, k_j)$ or u .*

Proof. Note that if $x \in \mathcal{X}$, $i, k \in [n]$ and $j \in [r]$ satisfy the conditions specified in (i), and $\bar{x} = \bar{x}(x, i, k)$ is given by (19), then $f_j(\bar{x}) - f_j(x) \geq 0$ follows, implying that both (i) and (ii) are indeed necessary conditions for duality (i.e. for $\mathcal{Y} = \mathcal{F}_t$ and $\mathcal{X} = \mathcal{I}^{-1}(\mathcal{F}_t)$).

To see the sufficiency, suppose that (i) and (ii) hold, and let y be a maximal element in $\mathcal{C} \setminus (\mathcal{X}^+ \cup \mathcal{Y}^-)$. Since $y \neq u$ by assumption, there is an $i \in [n]$ such that $y_i < u_i$. By maximality of y , there exists an $x \in \mathcal{X}$ such that $y + \mathbf{e}^i \geq x$. Let $j = \rho(x)$. If $y_k \geq \mu^j(x, i)_k + 1$, for some $k \neq i$, then $y \geq \bar{x}(x, i, k)$, and hence by (i), $y \in \mathcal{X}^+$, yielding a contradiction. We conclude therefore that $y \leq \mu^j(x, i)$, and consequently, as in the proof of Theorem 2, y is in the form given in (18). But then, by (ii), $y \in \mathcal{X}^+ \cup \mathcal{Y}^-$, another contradiction. $\square$

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, Fast discovery of association rules, in *Advances in Knowledge Discovery and Data Mining* (U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, eds.), pp. 307–328, AAAI Press, Menlo Park, California, 1996.
2. M. J. Atallah and G. N. Fredrickson, A note on finding a maximum empty rectangle, *Discrete Applied Mathematics* 13 (1986) 87–91.
3. J. C. Bioch and T. Ibaraki, Complexity of identification and dualization of positive Boolean functions, *Information and Computation* 123 (1995) 50–63.
4. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K. Makino, Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities, *SIAM Journal on Computing*, **31** (5) (2002) pp. 1624–1643.

5. E. Boros, K. Elbassioni, V. Gurvich and L. Khachiyan, An inequality for polymatroid functions and its applications, to appear in *Discrete Applied Mathematics*, 2003. (DIMACS Technical Report 2001-14, Rutgers University, ("http://dimacs.rutgers.edu/TechnicalReports/2001.html").
6. E. Boros, V. Gurvich, L. Khachiyan and K. Makino, Dual bounded generating problems: partial and multiple transversals of a hypergraph, *SIAM Journal on Computing* **30** (6) (2001) 2036–2050.
7. E. Boros, V. Gurvich, L. Khachiyan and K. Makino, On the complexity of generating maximal frequent and minimal infrequent sets in binary matrices. In: Proceedings of the 19th International Symposium on Theoretical Aspects of Computer Science (STACS 2002). (H. Alt and A. Ferreira, eds., Antibes Juan-les-Pins, France, March 14–16, 2002), Lecture Notes in Computer Science 2285 (2002) pp. 133–141, (Springer Verlag, Berlin, Heidelberg, New York).
8. B. Chazelle, R. L. (Scot) Drysdale III and D. T. Lee, Computing the largest empty rectangle, *SIAM Journal on Computing*, 15(1) (1986) 550–555.
9. Y. Crama, Dualization of regular Boolean functions, *Discrete Applied Mathematics* 16 (1987) 79–85.
10. D. Dentcheva, A. Prékopa and A. Ruszczyński, Concavity and efficient points of discrete distributions in Probabilistic Programming, *Mathematical Programming* **89** (2000) 55–77.
11. J. Edmonds, J. Gryz, D. Liang and R. J. Miller, Mining for empty rectangles in large data sets, in *Proc. 8th Int. Conf. on Database Theory (ICDT)*, Jan. 2001, *Lecture Notes in Computer Science* 1973, pp. 174–188.
12. M. L. Fredman and L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms, *Journal of Algorithms*, 21 (1996) 618–628.
13. V. Gurvich and L. Khachiyan, On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions, *Discrete Applied Mathematics*, 96–97 (1999) 363–373.
14. D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen, Data mining, hypergraph transversals and machine learning, in *Proceedings of the 16th ACM-SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, (1997) pp. 12–15.
15. E. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, *SIAM Journal on Computing*, 9 (1980) 558–565.
16. B. Liu, L.-P. Ku and W. Hsu, Discovering interesting holes in data, In *Proc. IJCAI*, pp. 930–935, Nagoya, Japan, 1997.
17. B. Liu, K. Wang, L.-F. Mun and X.-Z. Qi, Using decision tree induction for discovering holes in data, In *Proc. 5th Pacific Rim International Conference on Artificial Intelligence*, pp. 182–193, 1998.
18. K. Makino and T. Ibaraki, Interior and exterior functions of Boolean functions, *Discrete Applied Mathematics*, 69 (1996) 209–231.
19. K. Makino and T. Ibaraki, The maximum latency and identification of positive Boolean functions. *SIAM Journal on Computing*, 26:1363–1383, 1997.
20. A. Namaad, W. L. Hsu and D. T. Lee, On the maximum empty rectangle problem. *Discrete Applied Mathematics*, **8**(1984) 267–277.
21. M. Orlowski, A new algorithm for the large empty rectangle problem, *Algorithmica* 5(1) (1990) 65–73.
22. A. Prékopa, *Stochastic Programming*, (Kluwer, Dordrecht, 1995).
23. R. C. Read and R. E. Tarjan, Bounds on backtrack algorithms for listing cycles, paths, and spanning trees, *Networks* 5 (1975) 237–252.

24. R. Srikant and R. Agrawal, Mining generalized association rules. In *Proc. 21st International Conference on Very Large Data Bases*, pp. 407–419, 1995.
25. R. Srikant and R. Agrawal, Mining quantitative association rules in large relational tables. In *Proc. of the ACM-SIGMOD 1996 Conference on Management of Data*, pp. 1–12, 1996.

Higher Order Pushdown Automata, the Caucal Hierarchy of Graphs and Parity Games

Thierry Cachat

Lehrstuhl für Informatik VII, RWTH, D-52056 Aachen
Fax: (49) 241-80-22215, cachat@informatik.rwth-aachen.de

Abstract. We consider two-player parity games played on transition graphs of higher order pushdown automata. They are “game-equivalent” to a kind of model-checking game played on graphs of the infinite hierarchy introduced recently by Caucal. Then in this hierarchy we show how to reduce a game to a graph of lower level. This leads to an effective solution and a construction of the winning strategies.

1 Introduction

Games on finite graphs have been intensively studied for many years and used for modeling reactive systems. In the last years, two-player games on simple classes of infinite graphs have attracted attention. Parity games on pushdown graphs were solved by Walukiewicz in [18] using a reduction to finite graphs and a refined winning condition involving claims for one player (see also [2]). Kupferman and Vardi used two-way alternating automata in [15,17] to give a solution of parity games on the more general class of prefix recognizable graphs (see also [3]). In this framework, a solution means that given the finite description of the game, an algorithm should determine the winner and compute a winning strategy. The model checking problem is equivalent to the question of determining the winner: given a graph and a μ -calculus formula, one can construct a parity game such that the first player wins if and only if the formula is satisfied in the graph. In this framework of game also weaker logics and winning conditions have been studied, see among others [1,6,10,14].

In the present paper we consider a generalization to higher order pushdown automata for defining the game graph, where the player and the priority of a configuration are determined by the control state. We consider also the infinite hierarchy of graphs defined recently by Caucal [5] from the finite trees using inverse mapping and unfolding. The paper has two main contributions: an equivalence via game-simulation between higher order pushdown automata and the Caucal graphs, and an effective solution of parity games on both of these types of graphs. Using this game-simulation we show how to translate a game on a higher order pushdown automaton to a kind of model-checking game on a Caucal graph; one can then reduce such a game to a game on a graph from a lower level of the hierarchy and finally to a parity game on a finite graph, which gives an effective solution. It is then possible to reconstruct a winning strategy

for the original game. As far as we know this is the first result in this direction. So far only the decidability of MSO-properties of these graphs was known [5,13].

In the next section we define the different models of graphs and automata considered. Then we present in terms of game-simulation the reduction from higher order pushdown automata to the Caucal graphs and vice versa. In Section 4 we show that a game on a Caucal graph can be reduced to an equivalent game on a graph of lower level. For this we use a generalization of ideas from [17] to trees of infinite degree: the construction of an alternating one-way tree automaton equivalent to a given two-way alternating automaton. The main result that we use without proof is the positional (memoryless) determinacy of parity games of [8]: from any configuration one of the players has a positional winning strategy. We assume that the reader is familiar with the basic notions of language theory, automata, graphs and games (see [11] for an overview). The appendix (with proof of Lemma 5) is available at

<http://www-i7.informatik.rwth-aachen.de/~cachat/publi.html>

2 The Models

We note $[max] = \{0, \dots, max - 1\}$ for an integer $max > 0$. We write regular expressions in the usual way, for example $(a+b)^*c$ for letters a, b, c from a (finite) alphabet Γ . The empty word is ε and $\Gamma^{\leq 3} := \varepsilon + \Gamma + \Gamma^2 + \Gamma^3 = \bigcup_{i \leq 3} \Gamma^i$.

2.1 Parity Games

A *game structure* is a tuple (V_0, V_1, E, Ω) , where $V = V_0 \uplus V_1$ is a set of vertices partitioned into vertices of Player 0 and vertices of Player 1, $E \subseteq V \times V$ is a set of edges (directed, unlabeled), and $\Omega : V \rightarrow [max]$ is a priority function assigning to each vertex an integer between 0 and $max - 1$, with $max > 0$. Starting in a given initial vertex $\pi_0 \in V$, a play in (V_0, V_1, E, Ω) proceeds as follows: if $\pi_0 \in V_0$, Player 0 picks the first transition (move) to π_1 with $\pi_0 E \pi_1$, else Player 1 does, and so on from the new vertex π_1 . A play is a (possibly infinite) maximal sequence $\pi_0 \pi_1 \dots$ of successive vertices. For the winning condition we consider the min-parity version: Player 0 wins the play $\pi_0 \pi_1 \dots$ iff $\liminf_{k \rightarrow \infty} \Omega(\pi_k)$ is even, *i.e.*, iff the minimal priority seen infinitely often in the play is even. If the play is finite because of a deadlock, then by convention the player who should play loses immediately.

A *strategy* for Player 0 is a function associating to each prefix $\pi_0 \pi_1 \dots \pi_n$ of a play such that $\pi_n \in V_0$ a “next move” π_{n+1} with $\pi_n E \pi_{n+1}$. A strategy is *positional* (or memoryless) if it depends only on the current vertex π_n . We say that Player 0 wins the game from the initial vertex π_0 if he has a *winning strategy* for this game: a strategy such that he wins every play.

A game structure (V_0, V_1, E, Ω) is *game-simulated* by another game structure $(V'_0, V'_1, E', \Omega')$ from initial vertices $\pi_0 \in V$ and $\pi'_0 \in V'$ if

- Player 0 wins the game $(V'_0, V'_1, E', \Omega')$ from π'_0 iff Player 0 wins the game (V_0, V_1, E, Ω) from π_0 ,
- from a winning strategy of Player 0 in $(V'_0, V'_1, E', \Omega')$ one can compute a winning strategy of Player 0 in (V_0, V_1, E, Ω) .

2.2 Higher Order Pushdown System

We recall the definition from [13] (which is equivalent to the one from [9]), where we slightly change the terminology. A *level 1 store* (or *1-store*) over an alphabet Γ is an arbitrary sequence $[a_1, \dots, a_\ell]$ of elements of Γ , with $\ell \geq 0$. A *level n store* (or *n -store*), for $n \geq 2$, is a sequence $[s_1, \dots, s_\ell]$ of $(n-1)$ -stores, where $\ell \geq 0$. We allow a store to be empty. The following operations can be performed on 1-store:

$$\begin{aligned} push_1^a([a_1, \dots, a_{\ell-1}, a_\ell]) &:= [a_1, \dots, a_{\ell-1}, a_\ell, a] \text{ for all } a \in \Gamma, \\ pop_1([a_1, \dots, a_{\ell-1}, a_\ell]) &:= [a_1, \dots, a_{\ell-1}], \\ top([a_1, \dots, a_{\ell-1}, a_\ell]) &:= a_\ell. \end{aligned}$$

If $[s_1, \dots, s_\ell]$ is a store of level $n > 1$, the following operations are possible:

$$\begin{aligned} push_n([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, s_\ell, s_\ell], \\ push_k([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, push_k(s_\ell)] \text{ if } 2 \leq k < n, \\ push_1^a([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, push_1^a(s_\ell)] \text{ for all } a \in \Gamma, \\ pop_n([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, s_{\ell-1}], \\ pop_k([s_1, \dots, s_{\ell-1}, s_\ell]) &:= [s_1, \dots, s_{\ell-1}, pop_k(s_\ell)] \text{ if } 1 \leq k < n, \\ top([s_1, \dots, s_{\ell-1}, s_\ell]) &:= top(s_\ell). \end{aligned}$$

The operation pop_k is undefined on a store, whose top store of level k is empty. Similarly top is undefined on a store, whose top 1-store is empty. Given Γ and n , the set Op_n of operations (on a store) of level n consists of:

$push_k$ for all $2 \leq k \leq n$, $push_1^a$ for all $a \in \Gamma$, and pop_k for all $1 \leq k \leq n$.

A *higher order pushdown system* of level n (or *n -HPDS*) is a tuple $H = (P, \Gamma, \Delta)$ where P is the finite set of control locations, Γ the finite store alphabet, and $\Delta \subseteq P \times \Gamma \times P \times Op_n$ the finite set of (unlabeled) transition rules. A *configuration* of an n -HPDS H is a pair (p, s) where $p \in P$ and s is an n -store. The set of n -stores is denoted $\mathcal{S}_n$. We do not consider HPDS as accepting devices, hence there is no input alphabet. A HPDS $H = (P, \Gamma, \Delta)$ defines a *transition graph* (V, E) , where $V = \{(p, s) : p \in P, s \in \mathcal{S}_n\}$ is the set of all configurations, and

$$(p, s)E(p', s') \iff \exists (p, a, p', \sigma) \in \Delta : top(s) = a \text{ and } s' = \sigma(s).$$

Note that if the top 1-store is empty, no transition is possible. If necessary one can add a “bottom store symbol” $\perp \in \Gamma$ and define explicitly the corresponding transitions, such that it cannot be erased. To define a parity game on the graph of a HPDS, we assign a priority and a player to each control state, and we consider an initial configuration: a *game structure on a HPDS* H is a tuple $\mathcal{G} = (H, P_0, P_1, \Omega, s_0)$, where $P = P_0 \uplus P_1$ is a partition of the control states of H , $\Omega : P \rightarrow [max]$ is a priority function, and $s_0 \in V$. This extends naturally to a partition of the set of configurations and to a priority function defined on this set: with the notations of Section 2.1, $V_0 = P_0 \times \mathcal{S}_n$, $V_1 = P_1 \times \mathcal{S}_n$, $\Omega((p, s)) = \Omega(p)$, and E is defined above.

2.3 Caucal Hierarchy

We recall the definitions from [5]. Let L be a countable set of symbols for labeling arcs. A graph is here simple, oriented and arc labeled in a finite subset of L . Formally, a *graph* G is a subset of $V \times L \times V$, where V is an arbitrary set and such that its *label* set

$$\begin{aligned} L_G &:= \{a \in L \mid \exists s, t : (s, a, t) \in G\} && \text{is finite, but its vertex set} \\ V_G &:= \{s \mid \exists a, t : (s, a, t) \in G \vee (t, a, s) \in G\} && \text{is finite or countable.} \end{aligned}$$

We write also $t \xrightarrow[G]{a} s$ (or $t \xrightarrow{a} s$) for $(t, a, s) \in G$.

A *finite graph* is a graph whose vertex set is finite. A *tree* is a graph where each vertex has at most one predecessor, the unique root has no predecessor, and each vertex is accessible from the root. A vertex labeled tree is a tree, with a labeling function associating to each node a letter from a finite alphabet. The *unfolding* of a graph G is the following forest (set of trees):

$$Unf(G) := \{ws \xrightarrow{a} wsat : w \in (V_G \cdot L_G)^* \wedge s \xrightarrow[G]{a} t\} .$$

The unfolding $Unf(G, s)$ of a graph G from a vertex s is the restriction of $Unf(G)$ to the vertices accessible from s . Given a set of graphs $\mathcal{H}$, $Unf(\mathcal{H})$ is the set of graphs obtained by unfolding from the graphs of $\mathcal{H}$. Inverse arcs are introduced to move up and down in trees: we have a set $\bar{L} := \{\bar{a} \mid a \in L\}$ of fresh symbols in bijection with L . By definition, we have an arc $(s, \bar{a}, t)$ iff (t, a, s) is an arc of G . Note that in a tree there is at most one inverse arc from a given node.

In the usual way $s \xrightarrow[G]{w}^* t$ means that there is a *path* from s to t labeled by the word w . A *substitution* is a relation $h \subseteq L \times (L \cup \bar{L})^*$. It has finite domain if $Dom(h) := \{a \mid h(a) \neq \emptyset\}$ is finite. In this case, the inverse mapping of any graph G by h is

$$h^{-1}(G) = \{s \xrightarrow{a} t \mid \exists w \in h(a) : s \xrightarrow[G]{w}^* t\} .$$

The mapping h is rational if $h(a)$ is rational for every $a \in L$. Given a set of graphs $\mathcal{H}$, $Rat^{-1}(\mathcal{H})$ is the set of graphs obtained by inverse rational mapping from the graphs of $\mathcal{H}$. Let $Tree_0$ be the set of finite trees. The *Caucal Hierarchy* is defined in the following way:

$$\begin{aligned} Graph_n &:= Rat^{-1}(Tree_n) , \\ Tree_{n+1} &:= Unf(Graph_n) . \end{aligned}$$

Here $Graph_0$ is the set of finite graphs, $Tree_1$ is the set of regular trees of finite degree and $Graph_1$ is the set of prefix-recognizable graphs [4]. The other levels are mostly unknown.

Theorem 1 ([5]) $\bigcup_{n \geq 0} Graph_n$ is a family of graphs having a decidable monadic theory.

As a corollary, μ -calculus model checking on these graphs is decidable, and one can determine the winner of a parity game. But this result of decidability in [5] rely on the results from [4,7,19] whereas for the restricted framework of games we give here a direct algorithmic construction for determining the winner and a winning strategy.

2.4 Graph Automaton

An alternating parity graph automaton, or *graph automaton* for short, as defined in [15] is a tuple $\mathcal{A} = (Q, W, \delta, q_0, \Omega)$ where Q is a finite set of states, W is a finite set of edge labels, δ is the transition function to be defined below, $q_0 \in Q$ is the initial state, $\Omega : Q \rightarrow [max]$ is a priority function defining the acceptance condition: the minimal priority appearing infinitely often should be even. Let $next(W) = \{\varepsilon\} \cup \bigcup_{a \in W} \{[a], \langle a \rangle\}$, and $\mathcal{B}^+(next(W) \times Q)$ be the set of positive Boolean formulas built from the atoms in $next(W) \times Q$. The transition function is of the form $\delta : Q \rightarrow \mathcal{B}^+(next(W) \times Q)$. In the case of graphs, we will consider $W \subseteq L$, whereas in the case of trees, we will allow $W \subseteq L \cup \bar{L}$.

A run of a graph automaton $\mathcal{A} = (Q, W, \delta, q_0, \Omega)$ over a graph $G \subseteq V \times L \times V$ from a vertex $s_0 \in V$ is a labeled tree $\langle T_r, r \rangle$ in which every node is labeled by an element of $V \times Q$. This tree is like the unfolding of the product of the automaton by the graph. A node in T_r , labeled by (s, q) , describes a “copy” of the automaton that is in state q and is situated at the vertex s of G . Note that many nodes of T_r can correspond to the same vertex of G , because the automaton can come back to a previously visited vertex and because of alternation. The label of a node and its successors have to satisfy the transition function. Formally, a run $\langle T_r, r \rangle$ is a Σ_r -vertex labeled tree, where $\Sigma_r := V \times Q$ and $\langle T_r, r \rangle$ satisfies the following conditions:

- $r(t_0) = (s_0, q_0)$ where t_0 is the root of T_r .
- Consider $y \in T_r$ with $r(y) = (s, q)$ and $\delta(q) = \theta$. Then there is a (possibly empty) set $Y \subseteq next(W) \times Q$, such that Y satisfies θ , and for all $\langle d, q' \rangle \in Y$, the following hold:
 - If $d = \varepsilon$ then there exists a successor y' of y in T_r such that $r(y') = (s, q')$.
 - If $d = \langle a \rangle$ then there exists a successor y' of y in T_r , and a vertex s' such that $s \xrightarrow{a}_G s'$ and $r(y') = (s', q')$.
 - If $d = [a]$ then for each vertex s' such that $s \xrightarrow{a}_G s'$, there exists a successor y' of y in T_r such that $r(y') = (s', q')$.

The priority of a node $y \in T_r$, with $r(y) = (s, q)$, is $\Omega(q)$. A run is accepting if it satisfies the parity condition: along each infinite branch of T_r , the minimal priority appearing infinitely often is even.

When G is a tree, $\mathcal{A}$ is like an alternating two-way parity automaton of [15], because it can go up and down, but here the degree of the tree can be infinite. It is more general than the model of [12] which cannot distinguish between son and parent node. For the proofs we will also consider a tree automaton (defined as a graph automaton) that “reads” the labels of the vertices.

One can also consider the *model-checking game*: in a given graph Player 0 wants to prove that a formula is true and Player 1 has to challenge this. Similarly Player 0 wants to find an accepting run and Player 1 wants to refute it: a graph $G \subseteq V \times L \times V$ and a graph automaton $\mathcal{A}$ over G where $W = L_G$ define a parity game denoted by $(G, \mathcal{A})$. The configurations of the game are pairs $(s, q) \in V \times Q$, the initial configuration is (s_0, q_0) . In general one needs also other configurations corresponding to subformulas of $\delta(q)$ to allow existential / universal choices: Player 0 makes the existential choices, Player 1 makes the universal ones. It is well known that a run is a strategy for Player 0, and an accepting run is a winning strategy for Player 0, see e.g. [11, ch. 4].

3 Game-Simulation between HPDS and Caucal Graphs

In this section we show an equivalence between a model based on graph transformations, where the vertex set is “abstract” —the Caucal graphs— and a model based on rewriting of “concrete” nodes —the HPDS. This game simulation should be compared to the notion of weak (bi)simulation. Moreover it seems that one can deduce from the following construction that each transition graph of a HPDS is a graph of the Caucal hierarchy of the same level.

3.1 From HPDS to Caucal Graphs

Theorem 2 *Given a game structure $\mathcal{G}$ on a HPDS H of level n , one can construct a graph automaton $\mathcal{A}$ and a tree $T \in Tree_n$ such that $\mathcal{G}$ is game-simulated by $(T, \mathcal{A})$.*

The tree $T \in Tree_n$ depends only on n and Γ .

Proof: (sketch) We describe the construction for $n = 1, 2$ and 3 before we give the generalization. Let $\mathcal{G} = (H, P_0, P_1, \Omega, s_0)$, $H = (P, \Gamma, \Delta)$ of level n .

Case $n = 1$. The idea here is similar to that of [15] and [4].

Let T_1 be the complete Γ -tree. It is the unfolding of a finite graph with a unique vertex and so $T_1 \in Tree_1$. See an example in Figure 1 where $\Gamma = \{a, b\}$. This tree is isomorphic to Γ^* in the sense that each node is associated to the label of the path from the root to it (we write the store from bottom to top, so we consider suffix rewriting in the application of the rules). It is easy to simulate a 1-store with this tree: each node corresponds to a word, which is a store content. Intuitively the effect of a transition $(p, a, p', push_1^b)$ on the store is simulated over T_1 by a path $\bar{a}ab$. Formally the state space of $\mathcal{A}$ is $Q = P \times (L \cup \bar{L})^{\leq 3}$, where a state (p, ε) on a node $v \in T_1$ represents a configuration $(p, [v])$ of the HPDS (by abuse v is associated to a word of Γ^*), whereas the states (p, x) , $x \neq \varepsilon$ are intermediate states that simulate the behavior of the store. From these intermediate states, the transition is somehow “deterministic”: $\forall a \in L \cup \bar{L}, x \in (L \cup \bar{L})^{\leq 2}$:

$$\begin{aligned} &\text{if } p \in P_0 \text{ then } \delta((p, ax)) = \langle \langle a \rangle, (p, x) \rangle, \\ &\text{if } p \in P_1 \text{ then } \delta((p, ax)) = \langle [a], (p, x) \rangle. \end{aligned}$$

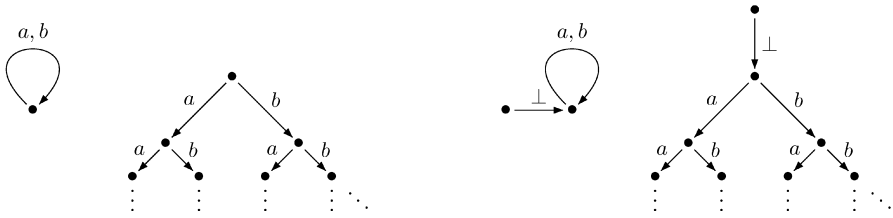


Fig. 1. The complete $\{a, b\}$ -tree T_1 obtained by unfolding of a finite graph, and the same with a “bottom stack symbol”

Note that here (on T_1), if $a \in L$ the “actions” $\langle a \rangle$ and $[a]$ are equivalent. From the states (p, ε) the corresponding player has to choose the move:

$$\begin{aligned} \text{if } p \in P_0 \text{ then } \delta((p, \varepsilon)) &= \bigvee_{(p, a, p', pop_1) \in \Delta} \langle \varepsilon, (p', \bar{a}) \rangle \vee \bigvee_{(p, a, p', push_1^b) \in \Delta} \langle \varepsilon, (p', \bar{a}ab) \rangle, \\ \text{if } p \in P_1 \text{ then } \delta((p, \varepsilon)) &= \bigwedge_{(p, a, p', pop_1) \in \Delta} \langle \varepsilon, (p', \bar{a}) \rangle \wedge \bigwedge_{(p, a, p', push_1^b) \in \Delta} \langle \varepsilon, (p', \bar{a}ab) \rangle. \end{aligned}$$

We see again that the convention is satisfied: when the play is in a deadlock, the player who should play loses immediately.

Case $n = 2$. For each letter $a \in \Gamma$, we assume that we have a fresh symbol $\dot{a}$ in L . We define the graph $G_1 \in Graph_1$ from the tree T_1 :

$$G_1 = h_1^{-1}(T_1),$$

where the (finite) substitution h_1 is the following:

$$\begin{aligned} h_1(a) &= a \text{ for all } a \in \Gamma, & h_1(2) &= \varepsilon, \\ h_1(\dot{a}) &= \bar{a} \text{ for all } a \in \Gamma. \end{aligned}$$

Hence we suppose that $2 \in L$ is a fresh symbol. A part of the graph G_1 is pictured in Figure 2. The loops labeled by 2 will be used to simulate the “copy” of the store content, *i.e.*, an operation $push_2$.

Then the tree $T_2 \in Tree_2$ is the unfolding of G_1 from the vertex that was the root of T_1 . In Figure 3 extra node-labels are added. They represent the corresponding 2-store. Note that several nodes can represent the same store content. The operations on 2-stores are simulated by paths in T_2 . More precisely, the effect of a transition is simulated in the following way if $\Gamma = \{a, b\}$:

$$\begin{aligned} (p, a, p', push_1^b) &\text{ corresponds to } \dot{a}ab, \\ (p, a, p', pop_1) &\text{ corresponds to } \dot{a}, \\ (p, a, p', push_2) &\text{ corresponds to } \dot{a}a2, \\ (p, b, p', pop_2) &\text{ corresponds to } \dot{b}(\bar{a} + \bar{b} + \bar{a} + \bar{b})^* \bar{2}. \end{aligned}$$

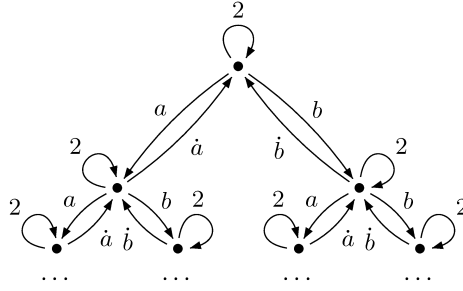


Fig. 2. Graph G_1 for $\Gamma = \{a, b\}$

Of course the expression $\dot{b}(\bar{a} + \bar{b} + \bar{a} + \bar{b})^* \bar{2}$ is regular, and one can move along such a path using three states of $\mathcal{A}$. Because we are on a tree, there is no infinite upward path. Following a 2-arc allows to copy the top 1-store because we stay exactly in the same position in G_1 . For popping the top 1-store, one has to find the last 2-arc that was used, and follow it in the reverse direction. Note that just after a $push_2$ (a 2-arc), we cannot move along an inverse arc $\bar{a}$ (to simulate a pop_1), that's why the arcs $\dot{a}$ are necessary.

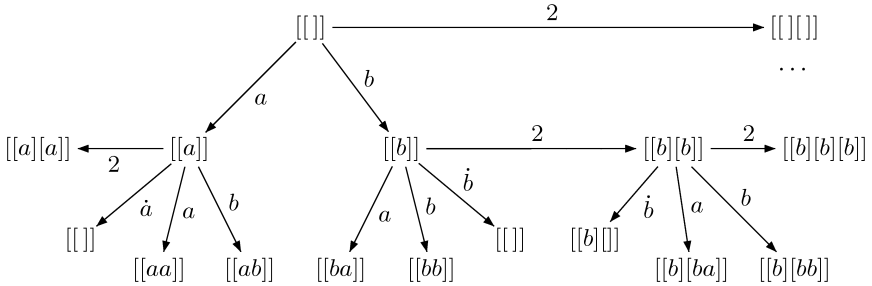


Fig. 3. An initial part of the tree T_2

Case $n = 3$. We go on with $G_2 \in Graph_2$, defined from T_2 :

$$G_2 = h_2^{-1}(T_2)$$

where the substitution h_2 is the following:

$$\begin{aligned} h_2(a) &= a \text{ for all } a \in \Gamma, & h_2(2) &= 2, \\ h_2(\dot{a}) &= \dot{a} \text{ for all } a \in \Gamma, & h_2(\dot{2}) &= \{\bar{a}, \bar{a} \mid a \in \Gamma\}^* \bar{2}, \\ h_2(3) &= \varepsilon. \end{aligned}$$

Then $T_3 \in Tree_3$ is the unfolding of G_2 from the “root” (of T_2). On T_3 one can simulate a 3-store, almost the same way as a 2-store is simulated on T_2 (here $\Gamma = \{a, b\}$):

$$\begin{aligned}
(p, a, p', push_1^b) &\text{ corresponds to } \dot{a}ab, \\
(p, a, p', pop_1) &\text{ corresponds to } \dot{a}, \\
(p, a, p', push_2) &\text{ corresponds to } \dot{a}a2, \\
(p, a, p', pop_2) &\text{ corresponds to } \dot{a}\dot{2}, \\
(p, a, p', push_3) &\text{ corresponds to } \dot{a}a3, \\
(p, a, p', pop_3) &\text{ corresponds to } \dot{a}(\bar{2} + \bar{\dot{2}} + \bar{a} + \bar{b} + \bar{\dot{a}} + \bar{\dot{b}})^* \bar{3}.
\end{aligned}$$

General case. It is easy to follow the construction: for $n \geq 3$, G_n is obtained from T_n using substitution h_n :

$$\begin{aligned}
h_n(a) &= a \text{ for all } a \in \Gamma, & h_n(k) &= k \text{ for all } 2 \leq k \leq n, \\
h_n(\dot{a}) &= \dot{a} \text{ for all } a \in \Gamma, & h_n(\dot{k}) &= \dot{k} \text{ for all } 2 \leq k < n, \\
h_n(\dot{n}) &= \left\{ \bar{a}, \bar{\dot{a}}, \bar{k}, \bar{\dot{k}} \mid a \in \Gamma, 2 \leq k < n \right\}^* \bar{n}, \\
h_n(n+1) &= \varepsilon,
\end{aligned}$$

and T_{n+1} is the unfolding of G_n from the “root”. The automaton $\mathcal{A}$ has the same states as H plus auxiliary states for the regular expressions. It is clear that the winner of $\mathcal{G}$ is the winner of $(T, \mathcal{A})$, and a winning strategy in $(T, \mathcal{A})$ can be translated to a winning strategy in $\mathcal{G}$ (the other direction holds also here). ■

3.2 From Caucal Graphs to HPDS

Lemma 3 *Given a graph $G \in Graph_n$ and a graph automaton $\mathcal{A}$, one can construct a game structure $\mathcal{G}$ on a HPDS H of level n such that $(G, \mathcal{A})$ is game-simulated by $\mathcal{G}$.*

Proof: (sketch) The result is clear for $n = 0$, because G and $\mathcal{A}$ have a finite number of vertices and states.

Given $T_1 \in Tree_1$, $T_1 = Unf(G_0, s)$ for some $G_0 \in Graph_0$, we let $\Gamma = V_{G_0} \times L_{G_0}$. Letters from Γ will be pushed on a 1-store to remember the position in the unfolding, which is a path from s . Additionally the labels from L_{G_0} will allow to determine which inverse arc is possible from the current position. To simplify the notation we write $\langle a, q' \rangle \in \delta(q)$ if $\langle a, q' \rangle$ is an atom present in the formula $\delta(q)$. It is clear that the existential/universal choices in the formula can be expressed in the control states of a HPDS, so we skip this part and concentrate on the actual “moves”:

$\langle a, q' \rangle \in \delta(q)$ corresponds to $(q, (v, -), q', push_1^{(u,a)})$ if $v \xrightarrow{G_0} u$,
 $\langle \bar{a}, q' \rangle \in \delta(q)$ corresponds to $(q, (-, a), q', pop_1)$.

A graph in $Graph_1$ can be simulated the same way using intermediate states for the rational substitutions.

Let $T_2 \in Tree_2$, $T_2 = Unf(G_1, s)$, T_2 can be simulated by a 2-store: each transition of G_1 is simulated on the top 1-store just like above, but the top 1-store has to be “copied” by a $push_2$ operation to keep track of the unfolding. It is also necessary to remember at each move the label of the arc of G_1 that was used. A solution is to use the following stack alphabet:

$$\Gamma = (V_{G_0} \times L_{G_0}) \uplus (\{2\} \times L_{G_1}) .$$

An action $\langle a, q' \rangle \in \delta(q)$ is simulated by the following sequence of operations:

$push_2$
 $< \text{simulation of an } a\text{-arc of } G_1 \text{ on the top 1-store } >$
 $push_1^{(2,a)} .$

And an action $\langle \bar{a}, q' \rangle \in \delta(q)$ in the following way:

$< \text{check that the top symbol is } (2, a) >$
 $pop_2 .$

And so on for $n \geq 3$. This construction is more natural if we use the model of higher order pushdown automata from [9], but both models are clearly equivalent [13]. ■

4 Reducing the Hierarchy Level

In this section we present our main result: an algorithmic solution of parity games on the graphs of the Caucal hierarchy, and hence on HPDS. The proof is by induction on the definition of the hierarchy, using the next two lemmas to obtain graphs of lower levels.

Lemma 4 *Given $G \in Graph_n$ and a graph automaton $\mathcal{A}$, the game $(G, \mathcal{A})$ can be effectively simulated by a game $(T, \mathcal{B})$, where $T \in Tree_n$, such that $G = h^{-1}(T)$, and $\mathcal{B}$ is a graph automaton.*

The proof uses similar techniques as in [2] or [15] for the case of prefix-recognizable graphs.

Proof: By definition $G = h^{-1}(T)$. The aim is to “simulate” an a -transition of $\mathcal{A}$ along an arc of G by a path in T : a sequence of transitions of $\mathcal{B}$ labeled by

a word of $h(a)$. The automaton $\mathcal{B} = (Q_{\mathcal{B}}, W_{\mathcal{B}}, \delta_{\mathcal{B}}, q_0, \Omega_{\mathcal{B}})$ will have the same states as $\mathcal{A}$ plus auxiliary states for this simulation. For each $a \in L$, $h(a)$ is regular. If $h(a) \neq \emptyset$, let

$$\mathcal{C}_a = (Q_a, W_a, \Delta_a, q_{0a}, F_a)$$

be a (non-deterministic) finite automaton on finite words recognizing $h(a)$. Here F_a is the set of final states. We consider $\mathcal{C}_a$ as a finite graph, and note the transitions $q_a \xrightarrow{\mathcal{C}_a} q'_a$ for $q_a, q'_a \in Q_a$. The new auxiliary states of $\mathcal{B}$ are of the form $(q_a, [q])$ and $(q_a, \langle q \rangle)$ for $q \in Q_{\mathcal{A}}$, $q_a \in Q_a$. To obtain the transitions of $\mathcal{B}$ from the transitions of $\mathcal{A}$,

$$\begin{aligned} \text{each atom } \langle [a], q \rangle &\text{ is replaced by } \langle \varepsilon, (q_{0a}, [q]) \rangle, \\ \text{and each } \langle \langle a \rangle, q \rangle &\text{ is replaced by } \langle \varepsilon, (q_{0a}, \langle q \rangle) \rangle \end{aligned}$$

in the body of a transition $\delta_{\mathcal{A}}(q')$. Of course the atoms $\langle \varepsilon, q \rangle$ remain unchanged. Then the new transitions of $\mathcal{B}$ are

$$\begin{aligned} \delta_{\mathcal{B}}((q_a, [q])) &= \bigwedge_{q_a \xrightarrow{\mathcal{C}_a} q'_a} \langle [b], (q'_a, [q]) \rangle \wedge \bigwedge_{q_a \in F_a} \langle \varepsilon, q \rangle, \\ \delta_{\mathcal{B}}((q_a, \langle q \rangle)) &= \bigvee_{q_a \xrightarrow{\mathcal{C}_a} q'_a} \langle \langle b \rangle, (q'_a, \langle q \rangle) \rangle \vee \bigvee_{q_a \in F_a} \langle \varepsilon, q \rangle, \end{aligned}$$

for each $a \in L$ such that $h(a) \neq \emptyset$.

To avoid the game to stay forever in the intermediate nodes of $\mathcal{B}$, we assign to these nodes a priority that is losing for the corresponding player. Suppose that the priority function $\Omega_{\mathcal{A}}$ of $\mathcal{A}$ ranges from 0 to $2c$, $c \geq 0$, then we fix

$$\Omega_{\mathcal{B}}((q_a, [q])) = 2c, \quad \Omega_{\mathcal{B}}((q_a, \langle q \rangle)) = 2c + 1.$$

And dually if the maximal priority of $\mathcal{A}$ is $2c + 1$, then the new priorities are $2c + 1$ and $2c + 2$. They do not interfere with the “real” game $(G, \mathcal{A})$. So one has one new priority and in the worst case the number of states of $\mathcal{B}$ is $|Q_{\mathcal{B}}| = |Q_{\mathcal{A}}| \left(1 + \sum_{h(a) \neq \emptyset} |Q_a|\right)$. ■

Lemma 5 *Given $T \in \text{Tree}_{n+1}$ and a graph automaton $\mathcal{A}$, the game $(T, \mathcal{A})$ can be effectively simulated by a game $(G, \mathcal{B})$, where $G \in \text{Graph}_n$, such that $T = \text{Unf}(G, s)$, and $\mathcal{B}$ is a graph automaton.*

This result is related to the k – covering of [7], where k is the number of states of $\mathcal{A}$. The proof is based here on the construction of a *one-way* tree automaton that is equivalent to $\mathcal{A}$. This construction was presented in [17] only in the case of (deterministic) trees of finite degree. The idea is that if Player 0 has a winning strategy in $(T, \mathcal{A})$, then he has also a positional winning strategy [8]: choosing always the same transition from the same vertex. This strategy can be encoded

as a labeling of T using a (big) finite alphabet. Then several conditions have to be checked to verify that this strategy is winning, but it can be done by a one-way automaton. Finely this strategy can be non-deterministically guessed by the automaton. And a one-way automaton cannot distinguish T and G .

We give here a flavor of this proof, details are in Appendix. Formally a *strategy* for $\mathcal{A}$ and a given tree T is a mapping

$$\tau : V_T \longrightarrow \mathcal{P}(Q \times \text{next}(W) \times Q) .$$

An element $(q, d, q') \in \tau(x)$ means that when arriving at node $x \in V_T$ in state q , the automaton should send a copy in state q' to the node in direction d (and maybe other copies in other directions). A strategy must satisfy the transition of $\mathcal{A}$, and a strategy has to be followed:

- $\forall x \in V_T, \forall (q, d, q') \in \tau(x) :$
- $\{(d_2, q_2) \mid (q, d_2, q_2) \in \tau(x)\}$ satisfies $\delta(q)$ and:
 - if $d = \varepsilon$ then $\exists d_1, q_1, (q', d_1, q_1) \in \tau(x)$ or $\emptyset$ satisfies $\delta(q')$,
 - if $d = [a]$ then $\forall y : x \xrightarrow{a}_T y \Rightarrow \exists d_1, q_1, (q', d_1, q_1) \in \tau(y)$ or $\emptyset$ satisfies $\delta(q')$,
 - if $d = \langle a \rangle$ then $\exists y : x \xrightarrow{a}_T y \wedge \exists d_1, q_1, (q', d_1, q_1) \in \tau(y)$ or $\emptyset$ satisfies $\delta(q')$.

For the root $t_0 \in V_T$ we have:

$$\exists d_1, q_1, (q_0, d_1, q_1) \in \tau(t_0) \text{ or } \emptyset \text{ satisfies } \delta(q_0) . \quad (1)$$

Considering $St := \mathcal{P}(Q \times \text{next}(W) \times Q)$ as an alphabet, a St -labeled tree (T, τ) defines a positional strategy on the tree T . One can construct a *one-way* automaton that checks that this strategy is correct according to the previous requirements.

The second step of the reduction from two-way to one-way is concerned with the priorities seen along (a branch of) the run, when one follows a strategy τ . To check the acceptance condition, it is necessary to follow each path of $\mathcal{A}$ in T up and down, and remember the priorities appearing. Such a path can be decomposed into a *downwards* path and several finite *detours* from the path, that come back to their origin (in a loop). Because each node has a unique parent and $\mathcal{A}$ starts at the root, we consider only downwards detour (each move $\bar{a}$ is in a detour). That is to say, if a node is visited more than once by a run, we know that the first time it was visited, the run came from above. This idea of detour is close to the idea of subgame in [18]. To keep track of these finite detours, we use the following annotation. An *annotation* for $\mathcal{A}$, a given tree T and a strategy τ is a mapping

$$\eta : V_T \longrightarrow \mathcal{P}(Q \times [\text{max}] \times Q) .$$

Intuitively $(q, f, q') \in \eta(x)$ means that from node x and state q there is a detour that comes back to x with state q' and the smallest priority seen along this detour is f . Again η can be considered as a labeling of T , and a *one-way* automaton can

check that the annotation is consistent with respect to the strategy in reading both labelings. A typical requirement is:

$$(q, [a], q_1) \in \tau(x) \Rightarrow \forall y \in V_T : x \xrightarrow{a} y \Rightarrow [(q_1, \bar{a}, q') \in \tau(y) \Rightarrow (q, \min(\Omega(q_1), \Omega(q')), q') \in \eta(x)] .$$

The last step is to check every possible branch of the run by using the detours: it is easy to define a one-way automaton $\mathcal{E}$ that “simulates” (follow) a branch of the run of $\mathcal{A}$. One can change the acceptance condition of $\mathcal{E}$ such that it accepts a tree labeled by τ and η iff *there exists* a branch in the corresponding run of $\mathcal{A}$ that *violates* the acceptance condition of $\mathcal{A}$. Then using techniques from [16] one can determinize and complement $\mathcal{E}$. Finally the product of the previous automata has to be build, to check all conditions together, and a “projection” is necessary to nondeterministically guess the labels, *i.e.*, the strategy and the annotation.

Theorem 6 *Parity games on higher order pushdown systems are solvable: one can determine the winner and compute a winning strategy.*

As a corollary we get a new proof that the μ -calculus model checking of these graphs is decidable (it was known as a consequence of the MSO-decidability).

Proof: Given a game structure $\mathcal{G}$ on a HPDS H of level n , one obtains from Theorem 2 a graph automaton $\mathcal{A}$ and a tree $T \in Tree_n$ such that $(T, \mathcal{A})$ is a game simulation of $\mathcal{G}$. By successive reductions using Lemmas 4 and 5, one can obtain a game on a finite graph which is equivalent to the initial game. Using classical techniques (see [11, ch. 7]), one can solve this game, and compute a positional strategy for the winner. Then one can step by step reconstruct the strategy for the graphs of higher levels. ■

5 Complexity, Strategy

The one-step reduction of Lemma 5 is in exponential time in the description of $T \in Tree_{n+1}$ and $\mathcal{A}$, and the size of the output is also exponential. For this reason the complexity of the complete solution of a parity game on a Caucal graph or on a HPDS is a tower of exponentials where the height is the level of the graph. The classical translation from parity game to μ -calculus to MSO and the corresponding decision procedure is already non-elementary (in the number of priorities) for level 1 graphs. And following [19] the (one-step) transformation of an MSO-formula from the unfolding to the original graph is also non-elementary.

Using the reductions presented here, one can compute a winning strategy for a 1-HPDS game which is a finite automaton that reads the current configuration and outputs the “next move”, like in [15,3]. But it is more natural to consider a *pushdown strategy* as introduced in [18]. It is a pushdown transducer that reads the moves of Player 1 and outputs the moves of Player 0. It needs additional memory (the stack), but the computation of the “next move” can be done in constant time. When we recompose the game, a strategy for an n -HPDS game is an n -HPDS with input and output which possibly needs to execute several transitions to compute the “next move” from a given configuration.

Acknowledgment. Great thanks to Didier Caucal, Christof Löding, Wolfgang Thomas, Stefan Wöhrle and to the referees for useful remarks.

References

1. T. CACHAT, *Symbolic strategy synthesis for games on pushdown graphs*, ICALP'02, LNCS 2380, pp. 704–715, 2002.
2. T. CACHAT, *Uniform solution of parity games on prefix-recognizable graphs*, INFINITY 2002, ENTCS 68(6), 2002.
3. T. CACHAT, *Two-way tree automata solving pushdown games*, ch. 17 in [11].
4. D. CAUCAL, *On infinite transition graphs having a decidable monadic theory*, ICALP'96, LNCS 1099, pp. 194–205, 1996.
5. D. CAUCAL, *On infinite terms having a decidable monadic theory*, MFCS'02, LNCS 2420, pp. 165–176, 2002.
6. D. CAUCAL, O. BURKART, F. MOLLER AND B. STEFFEN, *Verification on infinite structures*, Handbook of process algebra, Ch. 9, pp. 545–623, Elsevier, 2001.
7. B. COURCELLE AND I. WALUKIEWICZ, *Monadic second-order logic, graph convergences and unfoldings of transition systems*, Annals of Pure and Applied Logic 92–1, pp. 35–62, 1998.
8. E. A. EMERSON AND C. S. JUTLA, *Tree automata, mu-calculus and determinacy*, FoCS'91, IEEE Computer Society Press, pp. 368–377, 1991.
9. J. ENGELFRIET, *Iterated push-down automata*, 15th STOC, pp. 365–373, 1983.
10. J. ESPARZA, D. HANSEL, P. ROSSMANITH AND S. SCHWOON, *Efficient algorithm for model checking pushdown systems*, Technische Universität München, 2000.
11. E. GRÄDEL, W. THOMAS AND T. WILKE eds., *Automata, Logics, and Infinite Games*, A Guide to Current Research, LNCS 2500, 2002.
12. E. GRÄDEL AND I. WALUKIEWICZ, *Guarded fixed point logic*, LICS '99, IEEE Computer Society Press, pp. 45–54, 1999.
13. T. KNAPIK, D. NIWINSKI AND P. URZYCZYN, *Higher-order pushdown trees are easy*, FoSSaCS'02, LNCS 2303, pp. 205–222, 2002.
14. O. KUPFERMAN, M. Y. VARDI AND N. PITERMAN, *Model checking linear properties of prefix-recognizable systems*, CAV'02, LNCS 2404, pp. 371–385.
15. O. KUPFERMAN AND M. Y. VARDI, *An automata-theoretic approach to reasoning about infinite-state systems*, CAV'00, LNCS 1855, 2000.
16. W. THOMAS, *Languages, Automata, and Logic*, Handbook of formal language theory, vol. III, pp. 389–455, Springer-Verlag, 1997.
17. M. Y. VARDI, *Reasoning about the past with two-way automata.*, ICALP'98, LNCS 1443, pp. 628–641, 1998.
18. I. WALUKIEWICZ, *Pushdown processes: games and model checking*, CAV'96, LNCS 1102, pp. 62–74, 1996. Full version in Information and Computation 164, pp. 234–263, 2001.
19. I. WALUKIEWICZ, *Monadic second order logic on tree-like structures*, STACS'96, LNCS 1046, pp. 401–414. Full version in TCS 275 (2002), no. 1–2, pp. 311–346.

Undecidability of Weak Bisimulation Equivalence for 1-Counter Processes

Richard Mayr

Department of Computer Science, Albert-Ludwigs-University Freiburg
Georges-Koehler-Allee 51, D-79110 Freiburg, Germany.
`mayrri@informatik.uni-freiburg.de`
Fax: +49 761 203 8182

Abstract. We show that checking weak bisimulation equivalence of 1-counter nets (Petri nets with only one unbounded place) is undecidable. This implies the undecidability of weak bisimulation equivalence for 1-counter machines. The undecidability result carries over to normed 1-counter nets/machines.

Keywords: 1-counter nets, 1-counter machines, bisimulation

1 Introduction

Bisimulation equivalence plays a central role in the theory of process algebras [5]. The decidability and complexity of bisimulation problems for infinite-state systems has been studied intensively (see [10,1] for surveys). Here we consider process models with a finite control and one unbounded counter (i.e., a register holding a natural number). There are 1-counter machines (Minsky counter machines [6] with only one counter) and 1-counter nets (Petri nets [7] with only one unbounded place). 1-counter nets are equivalent to the subclass of 1-counter machines where the counter cannot be tested for zero. 1-counter machines are equivalent to a subclass of pushdown automata (with only one stack symbol plus a bottom symbol that can never be removed).

The state of the art: Strong bisimilarity was shown to be decidable for 1-counter machines (and thus also for 1-counter nets) by Jančar [3] and later for general pushdown automata by Sénizergues [9]. Weak (and even strong) bisimilarity are undecidable for general Petri nets [2], but the proof in [2] uses several unbounded places. Weak bisimilarity was shown to be undecidable for pushdown automata by Srba [12]. The decidability of weak bisimilarity for the less expressive models of 1-counter machines and 1-counter nets was still open.

Our contribution. We show that weak bisimilarity is undecidable for 1-counter nets (even if they are normed). The undecidability of weak bisimilarity for 1-counter machines follows directly. This more general undecidability result subsumes the previously known undecidability of weak bisimilarity for PDA [12] and Petri nets [2].

2 Definitions

1-counter nets are Petri nets with only one unbounded place. We describe them in a simplified notation.

Definition 1. A 1-counter net is given by $(S, X, \text{Act}, \Delta)$, where S is a finite set of control-states, X is a special symbol for the one unbounded place, Act is a finite set of atomic actions and Δ is a finite set of rewrite rules. The markings of this net are described in the form sX^n with $s \in S$ and $n \in \mathbb{N}_0$ (i.e., there are n tokens on the unbounded place and the configuration of the bounded rest of the net is s). The transitions of the net are described by the finite set Δ of rewrite rules of the form $s_1X^{m_1} \xrightarrow{a} s_2X^{m_2}$ with $s_1, s_2 \in S$, $m_1, m_2 \in \mathbb{N}_0$ and $a \in \text{Act}$. The labeled transition relation $\xrightarrow{a}$ on configurations (markings) of the 1-counter net is defined as follows: We have

$$s_1X^n \xrightarrow{a} s_2X^{n-m_1+m_2}$$

iff there exists some rule $(s_1X^{m_1} \xrightarrow{a} s_2X^{m_2}) \in \Delta$ s.t. $n \geq m_1$. 1-counter nets are equivalent to 1-counter machines where the counter cannot be tested for 0.

We consider the semantic equivalence *weak bisimulation equivalence* (also called *weak bisimilarity*) [5] over labeled transition systems (e.g., those generated by 1-counter nets).

Definition 2. The action τ is a special ‘silent’ internal action. The extended transition relation ‘ $\xRightarrow{a}$ ’ is defined by $E \xRightarrow{a} F$ iff either $E = F$ and $a = \tau$, or $E \xrightarrow{i} E' \xRightarrow{a} E'' \xrightarrow{j} F$ for some $i, j \in \mathbb{N}_0$. A binary relation R over states in a labeled transition graph is a weak bisimulation iff whenever $(E, F) \in R$ then for every $a \in \text{Act}$: if $E \xRightarrow{a} E'$ then there is $F \xRightarrow{a} F'$ s.t. $(E', F') \in R$ and if $F \xRightarrow{a} F'$ then there is $E \xRightarrow{a} E'$ s.t. $(E', F') \in R$. It is easy to show that weak bisimulations are closed under union and there exists a largest weak bisimulation which is an equivalence, denoted by $\approx$. States E, F are weakly bisimilar, written $E \approx F$, iff there is a weak bisimulation relating them. (Sometimes weak bisimulation is defined with $\Rightarrow$ instead of $\rightarrow$ everywhere. However, the two definitions are equivalent.)

Weak bisimulation equivalence can also be described by *weak bisimulation games* [13] between two players. One player, the ‘attacker’, tries to prove that two given processes P_1, P_2 are not weakly bisimilar, while the other player, the ‘defender’, tries to prevent this. A configuration of a weak bisimulation game is given by a pair of states (A, B) . Initially, this pair is (P_1, P_2) . The weak bisimulation game is played in rounds. In every round of the game the attacker chooses one process (i.e., either A or B) and performs an action (e.g., $A \xrightarrow{a} A'$). The defender must imitate this move and perform the same action in the other process. However, the defender is allowed to do an arbitrary number of τ -actions before and afterwards, since the defender uses the extended transition relation ‘ $\xRightarrow{a}$ ’, e.g., $B \xRightarrow{a} B'$. After the round the new configuration of the weak

bisimulation game is (A', B') . If one player cannot move then the other player wins. The defender wins every infinite game. Two processes are weakly bisimilar iff the defender has a winning strategy and non-weakly-bisimilar iff the attacker has a winning strategy.

We show the undecidability of weak bisimulation equivalence for 1-counter nets by a reduction from the acceptance problem of Minsky 2-counter machines.

Definition 3. A n -counter machine [6] M is described by a finite set of states Q , an initial state $q_0 \in Q$, a final state $\text{accept} \in Q$, n counters $c_1, \dots, c_n$ and a finite set of instructions of the form $(q : c_i := c_i + 1; \text{goto } q')$ or $(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{goto } q'')$ where $i \in \{1, \dots, n\}$ and $q, q', q'' \in Q$. A configuration of M is described by a tuple $(q, m_1, \dots, m_n)$ where $q \in Q$ and $m_i \in \mathbb{N}_0$ is the content of the counter c_i ($1 \leq i \leq n$). The possible computation steps are defined as follows:

1. $(q, m_1, \dots, m_n) \rightarrow (q', m_1, \dots, m_i + 1, \dots, m_n)$
if there is an instruction $(q : c_i := c_i + 1; \text{goto } q')$.
2. $(q, m_1, \dots, m_n) \rightarrow (q', m_1, \dots, m_n)$
if there is an instruction $(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{goto } q'')$
and $m_i = 0$.
3. $(q, m_1, \dots, m_n) \rightarrow (q'', m_1, \dots, m_i - 1, \dots, m_n)$
if there is an instruction $(q : \text{If } c_i = 0 \text{ then goto } q' \text{ else } c_i := c_i - 1; \text{goto } q'')$
and $m_i > 0$.

A counter machine is deterministic iff for every control-state $q \in Q$ there is at most one instruction $(q : \dots)$ at this control-state. A deterministic 2-counter machine accepts an input $n_1, n_2 \in \mathbb{N}_0$ iff the run starting at configuration (q_0, n_1, n_2) is finite and ends in the control-state 'accept'.

The following problem was shown to be undecidable by Minsky [6].

CM

Instance: A deterministic 2-counter machine M with initial state q_0 and $n_1, n_2 \in \mathbb{N}_0$.

Question: Does M accept (q_0, n_1, n_2) ?

We consider the problem of weak bisimulation equivalence of 1-counter nets.

1-CN $\approx$ 1-CN

Instance: A 1-counter net $(S, X, \text{Act}, \Delta)$ and $s_1, s_2 \in S$ and $n \in \mathbb{N}_0$.

Question: $s_1 X^n \approx s_2 X^n$?

We show the undecidability of $1\text{-CN} \approx 1\text{-CN}$ by a reduction from CM.

3 The Idea

The idea for the reduction is to encode the execution of the 2-counter machine into the weak bisimulation game between the two 1-counter nets. Every computation step of the 2-counter machine is emulated in a finite number of rounds

in the weak bisimulation game. The attacker has a universal winning strategy in the weak bisimulation game if and only if the 2-counter machine accepts. Thus, the two 1-counter nets are non-weakly-bisimilar iff the 2-counter machine accepts (i.e., the answer to $1\text{-CN} \approx 1\text{-CN}$ is no iff the answer to CM is yes).

The crucial part of the construction is that each of the two 1-counter nets stores the whole configuration of the 2-counter machine. Thus, the two numbers $n_1, n_2 \in \mathbb{N}_0$ in the two counters of the 2-counter machine must be stored in one number n of the 1-counter net. This is done by Gödel-coding of the form $n = 2^{n_1} 3^{n_2}$. The problem with this coding is that increment/decrement operations on n_1, n_2 now correspond to multiplication/division operations with constants on n . E.g., $n_2 := n_2 + 1$ is encoded by $n := 3 * n$. Testing n_1 or n_2 for zero is equivalent to testing n for non-divisibility by 2 or 3.

The central part of the proof is to show that it is possible to encode the operations of multiplication and division with constants and tests for divisibility with constants into weak bisimulation games on 1-counter nets. (Note that the same cannot be done for strong bisimulation, and indeed strong bisimilarity is decidable for 1-counter machines and 1-counter nets [3].)

4 Auxiliary Constructions

First, we describe some transition rules used for testing if a given number is exactly twice (or three times) as big as another.

$$\begin{array}{lll} t(1)X \xrightarrow{c} t(1) & t(2)X \xrightarrow{c} t(2)' & t(2)' \xrightarrow{c} t(2) \\ t(3)X \xrightarrow{c} t(3)' & t(3)' \xrightarrow{c} t(3)'' & t(3)'' \xrightarrow{c} t(3) \end{array}$$

Lemma 4. $t(1)X^n \approx t(1)X^m$ iff $n = m$, $t(1)X^n \approx t(2)X^m$ iff $n = 2m$, and $t(1)X^n \approx t(3)X^m$ iff $n = 3m$.

Proof. By induction on n, m . □

These transition rules are used for testing if a given number is divisible by 3.

$$\begin{array}{ll} t(3, 0)X \xrightarrow{c} t(3, 2) & t(3, 0)'X \xrightarrow{c} t(3, 2)' \\ t(3, 2)X \xrightarrow{c} t(3, 1) & t(3, 2)'X \xrightarrow{c} t(3, 1)' \\ t(3, 1)X \xrightarrow{c} t(3, 0) & t(3, 1)'X \xrightarrow{c} t(3, 0)' \\ t(3, 2) \xrightarrow{d} t(3, 2) & t(3, 2)'X \xrightarrow{d} t(3, 2)'X \\ t(3, 1) \xrightarrow{d} t(3, 1) & t(3, 1)'X \xrightarrow{d} t(3, 1)'X \end{array}$$

Lemma 5. $\forall i \in \{0, 1, 2\}. t(3, i)X^n \approx t(3, i)'X^n \iff n \bmod 3 = i$

Proof. By induction on n . For the base case $n = 0$ we have $t(3, 0) \not\rightarrow$ and $t(3, 0)' \not\rightarrow$ and thus $t(3, 0) \approx t(3, 0)'$. Furthermore, $t(3, 2) \xrightarrow{d} t(3, 2)$, but $t(3, 2)' \not\xrightarrow{d}$ and thus $t(3, 2) \not\approx t(3, 2)'$. Finally, $t(3, 1) \xrightarrow{d} t(3, 1)$, but $t(3, 1)' \not\xrightarrow{d}$ and thus $t(3, 1) \not\approx t(3, 1)'$.

Now let $n \geq 1$. (Let, by convention, $-1 \bmod 3 = 2$.) If $t(3, i)X^n \approx t(3, i)'X^n$ then $t(3, i)X^n \xrightarrow{c} t(3, (i-1) \bmod 3)X^{n-1}$, $t(3, i)'X^n \xrightarrow{c} t(3, (i-1) \bmod 3)'X^{n-1}$ and $t(3, (i-1) \bmod 3)X^{n-1} \approx t(3, (i-1) \bmod 3)'X^{n-1}$. By induction hypothesis $(n-1) \bmod 3 = (i-1) \bmod 3$ and thus $n \bmod 3 = i \bmod 3$. Since $i \in \{0, 1, 2\}$ we get $n \bmod 3 = i$.

Now let $n \bmod 3 = i$. For $i \in \{1, 2\}$ we have $t(3, i)X^n \xrightarrow{d} t(3, i)X^n$ and $t(3, i)'X^n \xrightarrow{d} t(3, i)'X^n$ since $n \geq 1$. (If $i = 0$ then the action d cannot be performed in either process.) Furthermore, we have $t(3, i)X^n \xrightarrow{c} t(3, (i-1) \bmod 3)X^{n-1}$ and $t(3, i)'X^n \xrightarrow{c} t(3, (i-1) \bmod 3)'X^{n-1}$. As $n \bmod 3 = i$ we have $(n-1) \bmod 3 = (i-1) \bmod 3$ and thus by induction hypothesis $t(3, (i-1) \bmod 3)X^{n-1} \approx t(3, (i-1) \bmod 3)'X^{n-1}$. It follows that $t(3, i)X^n \approx t(3, i)'X^n$. $\square$

These transition rules are used for testing if a given number is divisible by 2.

$$\begin{array}{ll} t(2, 0)X \xrightarrow{c} t(2, 1) & t(2, 0)'X \xrightarrow{c} t(2, 1)' \\ t(2, 1)X \xrightarrow{c} t(2, 0) & t(2, 1)'X \xrightarrow{c} t(2, 0)' \\ t(2, 1) \xrightarrow{d} t(2, 1) & t(2, 1)'X \xrightarrow{d} t(2, 1)'X \end{array}$$

Lemma 6. $\forall i \in \{0, 1\}. t(2, i)X^n \approx t(2, i)'X^n \iff n \bmod 2 = i$

Proof. Analogously to the proof of Lemma 5. $\square$

5 The Main Result

Let M be a Minsky 2-counter machine with a set of control states Q , initial control-state $q_0 \in Q$ and input values $n_1, n_2 \in \mathbb{N}_0$. We construct a 1-counter net (S, X, Act, Δ) such that M accepts (q_0, n_1, n_2) iff $q_0X^n \not\approx q_0'X^n$, where $n = 2^{n_1}3^{n_2}$ and $q_0, q_0' \in S$.

The configuration of the 2-counter machine is encoded in the 1-counter net as follows. The control-states of the 2-counter machine are encoded directly into the finite control of the 1-counter net. The natural numbers $n_1, n_2 \in \mathbb{N}_0$ stored in the counters c_1, c_2 of the 2-counter machine are encoded as X^n in the 1-counter net, where $n = 2^{n_1}3^{n_2}$.

Remark 7. Let $n_1, n_2 \in \mathbb{N}_0$ and $n = 2^{n_1}3^{n_2}$. Then we have: $n_1 = 0$ iff $n \bmod 2 \neq 0$, $n_2 = 0$ iff $n \bmod 3 \neq 0$. $n_1 := n_1 + 1$ corresponds to $n := 2n$, $n_1 := n_1 - 1$ to $n := n/2$, $n_2 := n_2 + 1$ to $n := 3n$ and $n_2 := n_2 - 1$ to $n := n/3$.

For every instruction $(q : c_2 := c_2 + 1; \text{goto } p)$ of the Minsky 2-counter machine we define the following rules of the 1-counter net. For some control-states q we define a new control-state $G(q)$ ('G' for 'generate') which behaves as follows: First an arbitrary number of symbols X are added or removed using only τ -actions. Then the action ' a ' is performed and the control-state becomes q .

$$\begin{array}{ll} G(q_2) \xrightarrow{\tau} G(q_2)X & G(q'_1) \xrightarrow{\tau} G(q'_1)X \\ G(q_2)X \xrightarrow{\tau} G(q_2) & G(q'_1)X \xrightarrow{\tau} G(q'_1) \\ G(q_2) \xrightarrow{a} q_2 & G(q'_1) \xrightarrow{a} q'_1 \end{array}$$

Now come the rules for encoding the operation.

$$\begin{array}{ll} q \xrightarrow{a} q_1 & \\ q \xrightarrow{\tau} G(q'_1) & q' \xrightarrow{\tau} G(q'_1) \\ q_1 \xrightarrow{t} t(3) & q'_1 \xrightarrow{t} t(1) \\ & q'_1 \xrightarrow{a} q'_2 \\ q_1 \xrightarrow{\tau} G(q_2) & q'_1 \xrightarrow{\tau} G(q_2) \\ q_2 \xrightarrow{t} t(1) & q'_2 \xrightarrow{t} t(1) \\ q_2 \xrightarrow{a} p & q'_2 \xrightarrow{a} p' \end{array}$$

The following lemma shows that these rules encode the 2-counter machine operation into the weak bisimulation game on the 1-counter net, provided that both players (attacker and defender) play optimally (i.e., avoid to lose if they can).

Lemma 8. *Let $(q : c_2 := c_2 + 1; \text{goto } p)$ be the instruction of the Minsky 2-counter machine at control-state q . Let $n_1, n_2 \in \mathbb{N}_0$ and $n = 2^{n_1}3^{n_2}$. The weak bisimulation game starting at the configuration $(qX^n, q'X^n)$ has the following properties:*

- The attacker has a strategy by which he can either (depending on the moves of the defender player) win, or at least force the weak bisimulation game into the configuration $(pX^m, p'X^m)$, where $m = 2^{n_1}3^{n_2+1}$.
- The defender has a strategy by which he can either (depending on the moves of the attacker) win, or at least enforce that the weak bisimulation game goes on and eventually reaches the configuration $(pX^m, p'X^m)$, where $m = 2^{n_1}3^{n_2+1}$.

Proof. We start the weak bisimulation game at $(qX^n, q'X^n)$. It proceeds as follows:

1. The attacker must play $qX^n \xrightarrow{a} q_1X^n$, otherwise the defender could make the two processes syntactically equal in the same round and win.
2. Now the defender has several choices:
 - a) If the defender plays $q'X^n \xrightarrow{a} G(q_2)X^k$ for some $k \in \mathbb{N}_0$ (by the rules for $G(q'_1)$ and $G(q_2)$) then the attacker can win. This is because $q_1X^n \not\approx G(q_2)X^k$, since $q_1X^n \xrightarrow{t}$ and $G(q_2)X^k \not\xrightarrow{t}$.

- b) Therefore the defender will play $q'X^n \xrightarrow{a} q'_1X^k$ for some $k \in \mathbb{N}_0$ (by definition of the rules for $G(q'_1)$). The defender will choose $k = 3n$, because otherwise the attacker could win. If $k \neq 3n$ then the attacker can play $q_1X^n \xrightarrow{t} t(3)X^n$ to which the defender must reply by $q'_1X^k \xrightarrow{t} t(1)X^k$. By Lemma 4 we have $t(3)X^n \not\approx t(1)X^k$ for $k \neq 3n$ and so the attacker can win. Therefore the defender will choose $k = 3n$. The configuration of the weak bisimulation game is now (q_1X^n, q'_1X^k) for $k = 3n$.
3. Now the attacker has several choices:
- a) If the attacker plays $q_1X^n \xrightarrow{t} t(3)X^n$ the defender replies $q'_1X^k \xrightarrow{t} t(1)X^k$. Since $k = 3n$ we have $t(3)X^n \approx t(1)X^k$ by Lemma 4 and thus the defender can win. (Analogously if the attacker plays $q'_1X^k \xrightarrow{t} t(1)X^k$.)
 - b) If the attacker plays $q_1X^n \xrightarrow{\tau} G(q_2)X^n$ the defender replies $q'_1X^k \xrightarrow{\tau} G(q_2)X^n$ (by definition of the rules for $G(q_2)$) and the defender wins. Similarly, if the attacker plays $q'_1X^k \xrightarrow{\tau} G(q_2)X^k$ then the defender replies $q_1X^n \xrightarrow{\tau} G(q_2)X^k$.
 - c) Therefore the attacker will choose the only remaining move $q'_1X^k \xrightarrow{a} q'_2X^k$.
4. To this move the defender will by reply by $q_1X^n \xrightarrow{a} q_2X^l$ (by definition of the rules for $G(q_2)$) for some $l \in \mathbb{N}_0$. The defender will choose $l = k$, because otherwise the attacker could win. If $l \neq k$ then the attacker can play $q_2X^l \xrightarrow{t} t(1)X^l$ to which the defender must reply by $q'_2X^k \xrightarrow{t} t(1)X^k$. By Lemma 4 we have $t(1)X^l \not\approx t(1)X^k$ for $l \neq k$ and so the attacker can win. Therefore the defender will choose $l = k$. The configuration of the weak bisimulation game is now (q_2X^k, q'_2X^k) with $k = 3n$.
5. Now the attacker has two choices:
- a) If the attacker plays $q_2X^k \xrightarrow{t} t(1)X^k$ the defender replies $q'_2X^k \xrightarrow{t} t(1)X^k$ and the defender wins. (Analogously if the attacker plays $q'_2X^k \xrightarrow{t} t(1)X^k$.)
 - b) Therefore the attacker will play $q_2X^k \xrightarrow{a} pX^k$ and the defender replies $q'_2X^k \xrightarrow{a} p'X^k$ (or vice-versa).
6. So, finally, unless one of the players has played sub-optimally and lost, we have reached the configuration $(pX^k, p'X^k)$ with $k = 3n$. As $n = 2^{n_1}3^{n_2}$ we have $k = 2^{n_1}3^{n_2+1} = m$. $\square$

Remark 9. For the instructions of the form $(q : c_1 := c_1 + 1; \text{goto } p)$ the construction is analogous. The only difference is that the constant $t(3)$ is replaced by $t(2)$ (since the number n in the 1-counter net is not tripled, but doubled). A lemma analogous to Lemma 8 is easy to show.

For every instruction of the form $(q : \text{If } c_2 = 0 \text{ then goto } p \text{ else } c_2 := c_2 - 1; \text{goto } r)$ of the Minsky 2-counter machine we define the following rules of the 1-counter net.

First, some auxiliary rules can generate (or remove) an arbitrary number of symbols X .

$$\begin{array}{ll} G(D(3, r)_1') \xrightarrow{\tau} G(D(3, r)_1')X & G(D(3, r)_2) \xrightarrow{\tau} G(D(3, r)_2)X \\ G(D(3, r)_1')X \xrightarrow{\tau} G(D(3, r)_1') & G(D(3, r)_2)X \xrightarrow{\tau} G(D(3, r)_2) \\ G(D(3, r)_1') \xrightarrow{a} D(3, r)_1' & G(D(3, r)_2) \xrightarrow{a} D(3, r)_2 \end{array}$$

The following auxiliary rules are used to encode the operation of dividing the number n by 3 (provided that it is divisible by 3) and then going to control-state r or r' .

$$\begin{array}{ll} D(3, r) \xrightarrow{a} D(3, r)_1 & \\ D(3, r) \xrightarrow{\tau} G(D(3, r)_1') & D(3, r)' \xrightarrow{\tau} G(D(3, r)_1') \\ D(3, r)_1 \xrightarrow{t} t(1) & D(3, r)' \xrightarrow{t} t(3) \\ & D(3, r)_1' \xrightarrow{a} D(3, r)_2' \\ D(3, r)_1 \xrightarrow{\tau} G(D(3, r)_2) & D(3, r)_1' \xrightarrow{\tau} G(D(3, r)_2) \\ D(3, r)_2 \xrightarrow{t} t(1) & D(3, r)_2' \xrightarrow{t} t(1) \\ D(3, r)_2 \xrightarrow{a} r & D(3, r)_2' \xrightarrow{a} r' \end{array}$$

Now come the rules for encoding the test-and-decrement operation. The intuition is that by action ‘ a ’ the attacker claims that $c_2 = 0$, (i.e., $n \bmod 3 \neq 0$) and by action ‘ b ’ the attacker claims $c_2 > 0$ (i.e., $n \bmod 3 = 0$). The defender can check these claims and win if they are wrong. The states $q_1(i)$ or $q_2(i)$ encode the counter-claims of the defender that $n \bmod 3 = i$.

$$\begin{array}{llll} q \xrightarrow{a} q_1 & q \xrightarrow{b} q_2 & & \\ q \xrightarrow{a} q_1' & q \xrightarrow{b} q_2' & q' \xrightarrow{a} q_1' & q' \xrightarrow{b} q_2' \\ q \xrightarrow{a} q_1(0) & q \xrightarrow{b} q_2(1) & q' \xrightarrow{a} q_1(0) & q' \xrightarrow{b} q_2(1) \\ & q \xrightarrow{b} q_2(2) & & q' \xrightarrow{b} q_2(2) \\ q_1 \xrightarrow{a} p & & q_1' \xrightarrow{a} p' & q_1(0) \xrightarrow{a} p \\ q_1 \xrightarrow{t} t(3, 0) & & q_1' \xrightarrow{t} t(3, 0) & q_1(0) \xrightarrow{t} t(3, 0)' \\ q_2 \xrightarrow{a} D(3, r) & & q_2' \xrightarrow{a} D(3, r)' & q_2(1) \xrightarrow{a} D(3, r) \\ & & & q_2(2) \xrightarrow{a} D(3, r) \\ q_2 \xrightarrow{t_1} t(3, 1) & q_2 \xrightarrow{t_2} t(3, 2) & q_2' \xrightarrow{t_1} t(3, 1) & q_2' \xrightarrow{t_2} t(3, 2) \\ & & q_2(1) \xrightarrow{t_1} t(3, 1)' & q_2(1) \xrightarrow{t_2} t(3, 2) \\ & & q_2(2) \xrightarrow{t_1} t(3, 1) & q_2(2) \xrightarrow{t_2} t(3, 2)' \end{array}$$

Lemma 10. *If $k = n \bmod 3$ and $k \in \{1, 2\}$ then $q_2X^n \approx q_2(k)X^n$.*

Proof. Let $j := 3 - k$. The defender has the following winning strategy:

- If the attacker plays $q_2X^n \xrightarrow{a} D(3, r)X^n$ then the defender can play $q_2(k)X^n \xrightarrow{a} D(3, r)X^n$ (and vice-versa) and thus the defender wins.
- If the attacker plays $q_2X^n \xrightarrow{t_j} t(3, j)X^n$ then the defender can play $q_2(k)X^n \xrightarrow{t_j} t(3, j)X^n$ (and vice-versa) and thus the defender wins.

- If the attacker plays $q_2X^n \xrightarrow{t_k} t(3, k)X^n$ then the defender can play $q_2(k)X^n \xrightarrow{t_k} t(3, k)'X^n$ (and vice-versa). By Lemma 5 $t(3, k)X^n \approx t(3, k)'X^n$ (since $k = n \bmod 3$) and thus the defender wins. $\square$

Lemma 11. *If $n \bmod 3 = 0$ then $q_1X^n \approx q_1(0)X^n$.*

Proof. The defender has the following winning strategy.

- If the attacker plays $q_1X^n \xrightarrow{a} pX^n$ then the defender replies $q_1(0)X^n \xrightarrow{a} pX^n$ (and vice-versa) and thus the defender wins.
- If the attacker plays $q_1X^n \xrightarrow{t} t(3, 0)X^n$ then the defender replies $q_1(0)X^n \xrightarrow{t} t(3, 0)'X^n$ (and vice-versa). By Lemma 5 we have $t(3, 0)X^n \approx t(3, 0)'X^n$ (since $n \bmod 3 = 0$) and thus the defender wins. $\square$

The following two lemmas show that these rules encode the 2-counter machine operation test-and-decrement into the weak bisimulation game on the 1-counter net.

Lemma 12. *Let $(q : \text{if } c_2 = 0 \text{ then goto } p \text{ else } c_2 := c_2 - 1; \text{goto } r)$ be a Minsky 2-counter machine instruction, $n_1, n_2 \in \mathbb{N}_0$ and $n = 2^{n_1}3^{n_2}$. If $n_2 = 0$ then the weak bisimulation game starting at the configuration $(qX^n, q'X^n)$ has the following properties:*

- *The attacker has a strategy by which he can either (depending on the moves of the defender player) win, or at least force the weak bisimulation game into the configuration $(pX^n, p'X^n)$.*
- *The defender has a strategy by which he can either (depending on the moves of the attacker) win, or at least enforce that the weak bisimulation game goes on and eventually reaches the configuration $(pX^n, p'X^n)$.*

Proof. Note that $n \bmod 3 \in \{1, 2\}$, because $n_2 = 0$. We start the weak bisimulation game at $(qX^n, q'X^n)$. It proceeds as follows:

1. The attacker has several choices:
 - a) If the attacker makes any move that is not $qX^n \xrightarrow{a} q_1X^n$ or $qX^n \xrightarrow{b} q_2X^n$ then the defender can make the two processes syntactically equal in the same round and thus the defender wins.
 - b) If the attacker plays $qX^n \xrightarrow{b} q_2X^n$ then the defender has the following strategy to win. Let $k := n \bmod 3$. $k \in \{1, 2\}$, since $n_2 = 0$. The defender plays $q'X^n \xrightarrow{a} q_2(k)X^n$. By Lemma 10 $q_2X^n \approx q_2(k)X^n$ and the defender wins.
 - c) Therefore the attacker will do $qX^n \xrightarrow{a} q_1X^n$.
2. Now the defender has two choices:

- a) If the defender plays $q'X^n \xrightarrow{a} q_1(0)X^n$ then the attacker has the following winning strategy. In the next round the attacker plays $q_1X^n \xrightarrow{t} t(3,0)X^n$ to which the defender can only reply $q_1(0)X^n \xrightarrow{t} t(3,0)'X^n$. By Lemma 5 we have $t(3,0)X^n \not\approx t(3,0)'X^n$, because $n \bmod 3 \neq 0$, and thus the attacker can win.
- b) Therefore the defender plays $q'X^n \xrightarrow{a} q'_1X^n$. The configuration is now (q_1X^n, q'_1X^n) .
3. Now the attacker has two choices: If the attacker plays $q_1X^n \xrightarrow{t} t(3,0)X^n$ then the defender replies $q'_1X^n \xrightarrow{t} t(3,0)X^n$ (and vice versa) and so the defender wins. Therefore the attacker will play $q_1X^n \xrightarrow{a} pX^n$ (or $q'_1X^n \xrightarrow{a} p'X^n$, this case is symmetric).
4. The defender can only reply $q'_1X^n \xrightarrow{a} p'X^n$. The configuration is now $(pX^n, p'X^n)$. $\square$

Lemma 13. *Let $(q : \text{If } c_2 = 0 \text{ then goto } p \text{ else } c_2 := c_2 - 1; \text{goto } r)$ be the instruction of the Minsky 2-counter machine at control-state q . Let $n_1, n_2 \in \mathbb{N}_0$ and $n = 2^{n_1}3^{n_2}$. If $n_2 > 0$ then the weak bisimulation game starting at the configuration $(qX^n, q'X^n)$ has the following properties:*

- *The attacker has a strategy by which he can either (depending on the moves of the defender player) win, or at least force the weak bisimulation game into the configuration $(rX^m, r'X^m)$, where $m = 2^{n_1}3^{n_2-1}$.*
- *The defender has a strategy by which he can either (depending on the moves of the attacker) win, or at least enforce that the weak bisimulation game goes on and eventually reaches the configuration $(rX^m, r'X^m)$, where $m = 2^{n_1}3^{n_2-1}$.*

Proof. Note that $n \bmod 3 = 0$, because $n_2 > 0$. We start the weak bisimulation game at $(qX^n, q'X^n)$. It proceeds as follows:

1. The attacker has several choices:
 - a) If the attacker makes any move that is not $qX^n \xrightarrow{a} q_1X^n$ or $qX^n \xrightarrow{b} q_2X^n$ then the defender can make the two processes syntactically equal in the same round and thus the defender wins.
 - b) If the attacker plays $qX^n \xrightarrow{a} q_1X^n$ then the defender can reply $q'X^n \xrightarrow{a} q_1(0)X^n$ and wins, because $q_1X^n \approx q_1(0)X^n$ for $n \bmod 3 = 0$, by Lemma 11.
 - c) Therefore the attacker will play $qX^n \xrightarrow{b} q_2X^n$.
2. Now the defender has three choices:
 - a) If the defender does $q'X^n \xrightarrow{b} q_2(1)X^n$ then the attacker has the following winning strategy. The attacker plays $q_2X^n \xrightarrow{t} t(3,1)X^n$ to which the defender can only reply $q_2(1)X^n \xrightarrow{t} t(3,1)'X^n$. By Lemma 5 $t(3,1)X^n \not\approx t(3,1)'X^n$ (since $n \bmod 3 \neq 1$) and thus the attacker can win.
 - b) Similarly, if the defender does $q'X^n \xrightarrow{b} q_2(2)X^n$ then the attacker can also win by Lemma 5, since $n \bmod 3 \neq 2$.

- c) Therefore the defender plays $q'X^n \xrightarrow{b} q'_2X^n$. The configuration is now (q_2X^n, q'_2X^n) .
3. The attacker now plays $q_2X^n \xrightarrow{a} D(3, r)X^n$ to which the defender replies $q'_2X^n \xrightarrow{a} D(3, r)'X^n$ (or vice-versa). (If the attacker played any other move by action t_1 or t_2 then the processes would become syntactically equal in the same round and the defender would win.) The configuration is now $(D(3, r)X^n, D(3, r)'X^n)$.
4. The attacker must now play $D(3, r)X^n \xrightarrow{a} D(3, r)_1X^n$, otherwise the defender could make the two processes syntactically equal in the same round and win.
5. Now the defender has several choices:
- a) If the defender plays $D(3, r)'X^n \xrightarrow{a} G(D(3, r)_2)X^k$ for some $k \in \mathbb{N}_0$ (by the rules for $G(D(3, r)'_1)$ and $G(D(3, r)_2)$) then the attacker wins. This is because

$$D(3, r)_1X^n \not\approx G(D(3, r)_2)X^k$$

since $D(3, r)_1X^n \xrightarrow{t} t(1)X^n$ and $G(D(3, r)_2)X^k \not\xrightarrow{t} t(1)X^n$.

- b) Therefore the defender plays $D(3, r)'X^n \xrightarrow{a} D(3, r)'_1X^k$ for some $k \in \mathbb{N}_0$ (by def. of the rules for $G(D(3, r)'_1)$). The defender will choose $k = n/3$ (this is possible, since $n \bmod 3 = 0$) for the following reason. If $k \neq n/3$ then the attacker can play $D(3, r)_1X^n \xrightarrow{t} t(1)X^n$ to which the defender must reply by $D(3, r)'_1X^k \xrightarrow{t} t(3)X^k$. By Lemma 4 we have $t(1)X^n \not\approx t(3)X^k$ for $k \neq n/3$ and so the attacker can win. Therefore the defender will choose $k = n/3$. So the configuration is now $(D(3, r)_1X^n, D(3, r)'_1X^k)$ with $k = n/3$.
6. Now the attacker has several choices:
- a) If the attacker plays $D(3, r)_1X^n \xrightarrow{t} t(1)X^n$ the defender replies $D(3, r)'_1X^k \xrightarrow{t} t(3)X^k$. Since $k = n/3$ we have $t(1)X^n \approx t(3)X^k$ by Lemma 4 and thus the defender can win. (Analogously if the attacker plays $D(3, r)'_1X^k \xrightarrow{t} t(3)X^k$.)
- b) If the attacker plays $D(3, r)_1X^n \xrightarrow{\tau} G(D(3, r)_2)X^n$ then the defender can reply $D(3, r)'_1X^k \xrightarrow{\tau} G(D(3, r)_2)X^n$ (by def. of the rules for $G(D(3, r)_2)$) and the defender wins. Similarly, if the attacker plays $D(3, r)'_1X^k \xrightarrow{\tau} G(D(3, r)_2)X^k$ then the defender replies $D(3, r)_1X^n \xrightarrow{\tau} G(D(3, r)_2)X^k$.
- c) Therefore the attacker chooses the only remaining move $D(3, r)'_1X^k \xrightarrow{a} D(3, r)'_2X^k$.
7. To this move the defender will reply by $D(3, r)_1X^n \xrightarrow{a} D(3, r)_2X^l$ (by definition of the rules for $G(D(3, r)_2)$) for some $l \in \mathbb{N}_0$. The defender will choose $l = k$ for the following reason. If $l \neq k$ then the attacker can play $D(3, r)_2X^l \xrightarrow{t} t(1)X^l$ to which the defender must reply by $D(3, r)'_2X^k \xrightarrow{t} t(1)X^k$. By Lemma 4 we have $t(1)X^l \not\approx t(1)X^k$ for $l \neq k$ and so the attacker can win. Therefore the defender will choose $l = k$. So the configuration of the weak bisimulation game is now $(D(3, r)_2X^k, D(3, r)'_2X^k)$ with $k = n/3$.
8. The attacker has two choices:

- a) If the attacker plays $D(3, r)_2 X^k \xrightarrow{t} t(1)X^k$ the defender replies $D(3, r)_2' X^k \xrightarrow{t} t(1)X^k$ (and vice-versa) and the defender wins.
 - b) Therefore the attacker will play $D(3, r)_2 X^k \xrightarrow{a} rX^k$ and the defender must reply $D(3, r)_2' X^k \xrightarrow{a} r'X^k$ (or vice-versa).
9. So, finally, unless one of the players has played sub-optimally and lost, we have reached the configuration $(rX^k, r'X^k)$ with $k = n/3$. As $n = 2^{n_1}3^{n_2}$ and $n_2 > 0$ we have $k = 2^{n_1}3^{n_2-1} = m$. $\square$

Remark 14. For the instructions of the form $(q : \text{If } c_1 = 0 \text{ then goto } p \text{ else } c_1 := c_1 - 1; \text{goto } r)$ the construction is similar, but slightly simpler. In this case we test the number n for divisibility by 2 and divide by 2 if possible (instead of by 3 for instructions on counter c_2 as shown above). Lemmas analogous to Lemma 12 and Lemma 13 are easy to show.

Finally, we add one last rule to the 1-counter net, which is used to distinguish accepting- and non-accepting states of the Minsky 2-counter machine. By Definition 3 the 2-counter machine has only one accepting state ‘accept’. We add the rule

$$\text{accept} \xrightarrow{e} \text{accept}$$

Thus we get $\text{accept}X^n \not\approx \text{accept}'X^m$ for all n, m , since action e is not possible at control-state accept' .

Lemma 15. *Let M be a Minsky 2-counter machine with a set of control states Q , initial control-state $q_0 \in Q$ and input values $n_1, n_2 \in \mathbb{N}_0$. One can effectively construct a 1-counter net $(S, X, \text{Act}, \Delta)$ (depending only on M) such that M accepts (q_0, n_1, n_2) iff $q_0X^n \not\approx q_0'X^n$, where $n = 2^{n_1}3^{n_2}$ and $q_0, q_0' \in S$.*

Proof. The 1-counter net is constructed as shown above and depends only on M . Now we show the correctness.

- If M accepts (q_0, n_1, n_2) then the attacker has a winning strategy in the weak bisimulation game starting at $(q_0X^n, q_0'X^n)$. By Lemma 8, Remark 9, Lemma 12, Lemma 13 and Remark 14 the attacker has a strategy by which (depending on the moves of the defender) he either wins directly, or simulates the 2-counter machine operation correctly. Since M accepts (q_0, n_1, n_2) , the weak bisimulation game will eventually reach some configuration $(\text{accept}X^i, \text{accept}'X^i)$ for some i (unless the attacker has already won earlier). Now the attacker wins, because $\text{accept}X^i \not\approx \text{accept}'X^i$. Therefore $q_0X^n \not\approx q_0'X^n$.
- If M does not accept (q_0, n_1, n_2) then the defender has a winning strategy in the weak bisimulation game starting at the configuration $(q_0X^n, q_0'X^n)$. By Lemma 8, Remark 9, Lemma 12, Lemma 13 and Remark 14 the defender has a strategy by which (depending on the moves of the attacker) he either wins directly, or simulates the 2-counter machine operation correctly. Since M does not accept (q_0, n_1, n_2) , the weak bisimulation game will never reach

a configuration with the state *accept*. So the weak bisimulation game will either continue forever, or the defender will win directly by one of the cases shown above. In any case the defender wins and thus $q_0X^n \approx q'_0X^n$. $\square$

Now we can show the main theorem.

Theorem 16. *There exists a fixed 1-counter net, such that, for input $n \in \mathbb{N}_0$, the question $q_0X^n \approx q'_0X^n$ is undecidable.*

Proof. There exists a fixed universal Minsky 2-counter machine M_u (analogously to the universal Turing-machine) [6] for which the problem, for input $m \in \mathbb{N}_0$, if it accepts $(q_0, m, 0)$ is undecidable. Based on M_u , we construct a fixed 1-counter net as shown above. Let $n = 2^m$. By Lemma 15 we have $q_0X^n \not\approx q'_0X^n$ iff M_u accepts $(q_0, m, 0)$. $\square$

Corollary 17. *Weak bisimulation equivalence is undecidable for 1-counter nets and 1-counter machines.*

Remark 18. The question $q_0X^0 \approx q'_0X^0$ is also undecidable for general (non-fixed) 1-counter nets, since acceptance of $(q_0, 0, 0)$ is undecidable for 2-counter machines.

Furthermore, all our undecidability results carry over to the subclass of *normed* 1-counter nets. (Normedness means here that from every reachable configuration it is possible to empty the unbounded place.) It suffices to add the following rules to our construction above: $q \xrightarrow{y} z$, $q' \xrightarrow{y} z$ and $zX \xrightarrow{\tau} z$ for all $q \in Q$. (z is a new state and y a new action.) This modified system is normed, but still preserves all properties shown above.

6 Conclusion

We have shown the undecidability of weak bisimulation equivalence for (normed) 1-counter nets and 1-counter machines. (This contrasts with strong bisimulation equivalence which is decidable for these models [3]). Our undecidability result is more general than Srba's result on undecidability of weak bisimilarity for pushdown automata [12]. No stack is needed, since one simple counter suffices. Moreover, it is not even necessary that the counter can be tested for zero. Our construction uses only one unbounded Petri net place.

However, a crucial requirement for our construction is the existence of a global finite control of our (possibly infinite-state) processes. Therefore our undecidability proof does not carry over to those classes of infinite-state processes that are not closed under product with finite automata, like context-free processes (BPA), basic parallel processes (BPP) or PA-processes (see, e.g., [4,10]). Undecidability of weak bisimilarity for PA-processes has been shown with a very different technique [11]. Decidability of weak bisimilarity for BPA and BPP is

still open, but conjectured to be decidable (especially for BPP, due to some recent results by Jančar [10]).

Still, (normed) 1-counter nets are a very weak model that is subsumed by most classes of infinite-state systems. Therefore one can say as a rule of thumb: “Weak bisimilarity is undecidable for most classes of infinite-state systems that are closed under product with finite-automata.”

Remark 19. The undecidability result for weak bisimilarity of 1-counter nets carries over to the even weaker model of lossy 1-counter nets (where the unbounded place can spontaneously lose tokens). Note that, up-to weak bisimilarity, lossy 1-counter nets are a proper subclass of 1-counter nets. The proof is similar to the one given here, but more technically complex in some details. The idea is to use an additional technique from [8] by which one can ensure that whenever one player loses tokens then the other player wins, thus effectively ruling out lossy behavior. The proof can be found in the forthcoming journal version of this paper.

References

- [1] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.
- [2] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *TCS*, 148:281–301, 1995.
- [3] P. Jančar. Decidability of bisimilarity for one-counter processes. *Information and Computation*, 158:1–17, 2000.
- [4] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.
- [5] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- [6] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.
- [7] J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.
- [8] Ph. Schnoebelen. Bisimulation and other undecidable equivalences for lossy channel systems. In *Proc. of TACS 2001*, volume 2215 of *LNCS*, pages 385–399. Springer Verlag, 2001.
- [9] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proc. of FOCS’98*. IEEE, 1998.
- [10] J. Srba. Roadmap of infinite results. *Bulletin of the European Association for Theoretical Computer Science*, 78:163–175, October 2002. Columns: Concurrency. Regularly updated online version at <http://www.brics.dk/~srba/roadmap>.
- [11] J. Srba. Undecidability of weak bisimilarity for PA-processes. In *Proc. Developments in Language Theory 2002*, *LNCS*. Springer-Verlag, 2002. To appear.
- [12] J. Srba. Undecidability of weak bisimilarity for pushdown processes. In *Proc. of CONCUR 2002*, volume 2421 of *LNCS*, pages 579–593. Springer Verlag, 2002.
- [13] C. Stirling. The joys of bisimulation. In *Proc. of MFCS’98*, volume 1450 of *LNCS*, pages 142–151. Springer Verlag, 1998.

Bisimulation Proof Methods for Mobile Ambients

Massimo Merro¹ and Francesco Zappa Nardelli²

¹ Università di Verona, Italy

² LIENS, Paris, France

Abstract. We study the behavioural theory of Cardelli and Gordon's Mobile Ambients. We give an *LTS* based operational semantics, and a labelled *bisimulation* based equivalence that coincides with reduction barbed congruence. We also provide *up-to proof techniques* and prove a set of *algebraic laws*, including the perfect firewall equation.

Introduction

The calculus of *Mobile Ambients* [4], abbreviated MA, has been introduced as a process calculus for describing *mobile agents*. In MA, the term $n[P]$ represents an agent, or *ambient*, named n , executing the code P . The ambient n is a bounded, protected, and (potentially) mobile space where the computation P takes place. In turn P may contain other ambients, may perform (local) *communications*, or may exercise *capabilities*, which allow entry to or exit from named ambients. *Ambient names*, such as n , are used to control access to the ambient's computation space and may be dynamically created as in the π -calculus, [17], using the construct $(\nu n)P$. A *system* in MA consists of a collection of ambients running in parallel where the knowledge of certain ambient names may be restricted.

Behavioural equality is a central idea in process calculi. In this paper we focus on a generalisation of *reduction barbed congruence* of [12]. Reduction barbed congruence is the largest equivalence relation that (i) is a congruence, (ii) preserves, in some sense, the reduction semantics of the language; (iii) preserves *barbs*, some simple observational property of terms.

However, context-based behavioural equalities, such as reduction barbed congruence, suffer from the universal quantification on contexts. Simpler proof techniques are based on *labelled bisimilarities*, [19], whose definitions do not use context quantification. These bisimilarities should imply, or (better) coincide with, reduction barbed congruence [20,1,8]. The behaviour of processes is characterised using co-inductive relations defined over a *labelled transition system*, or *LTS*, a collection of relations of the form

$$P \xrightarrow{\alpha} Q.$$

Intuitively, the action α in the judgment $P \xrightarrow{\alpha} Q$ represents some small context P can interact with; if the labelled bisimilarity coincides with reduction barbed

congruence then this collection of small contexts, codified as actions, is sufficient to capture all possible interactions that processes can have with arbitrary contexts.

Although the idea of bisimulation is very general and does not rely on the specific syntax of the calculus, the definition of an appropriate notion of bisimilarity for Mobile Ambients revealed harder than expected. The reasons of that can be resumed as follows:

- It is difficult for an ambient n to control interferences that may originate either from other ambients in its environment or from the computation running at n itself, [13].
- Ambient mobility is asynchronous — no permission is required to migrate into an ambient. As noticed in [23], this may cause a *stuttering* phenomenon originated by ambients that may repeatedly enter and exit another ambient. As stuttering cannot be observed, any successful characterisation of reduction barbed congruence should not observe stuttering [23].
- One of the main algebraic laws of MA is the *perfect firewall equation*, [4]:

$$(\nu n)n[P] = \mathbf{0} \quad \text{for } n \text{ not in } P.$$

If you suppose $P = \mathbf{in}_k.\mathbf{0}$, it is evident that a bisimilarity that wants to capture this law must not observe the movements of *secret ambients*, that is those ambients, like n , whose names are not known by the rest of the system.

In [14], it is introduced a labelled bisimilarity for an “easier” variant of MA, called SAP, equipped with (i) *synchronous mobility*, as in Levi and Sangiorgi’s *Safe Ambients* [13], and (ii) *passwords* to exercise control over, and differentiate between, different ambients that may wish to exercise a capability. The main result in [14] is the characterisation of reduction barbed congruence in terms of the labelled bisimilarity. The result holds only in SAP and crucially relies on the two features (i) and (ii) mentioned above.

The current paper is the natural continuation of [14] where, now, we tackle the original problem: *to provide bisimulation proof methods for Mobile Ambients*.

Contribution. First of all, as in the Distributed π -calculus [11], we rewrite the syntax of MA in two levels: *processes* and *systems*. This is because we are interested in studying systems rather than processes. So, our behavioural equalities are defined over systems. This little expedient allows us to (i) focus on higher-order actions, where movement of code is involved, and (ii) to model stuttering in terms of standard τ -actions.

We introduce a new labelled transition system for MA that is used to define a labelled bisimilarity over systems. The resulting bisimilarity, denoted by $\approx$, is in *late* style. The definition of $\approx$ reminds us that one of the asynchronous bisimilarity found in [1]. Indeed, as for inputs in asynchronous π , our bisimilarity does not observe the movements of secret ambients.

We prove that the relation $\approx$ completely characterises reduction barbed congruence over systems. Then, we enhance our proof methods by defining two *up-to*

Table 1. The Mobile Ambients in Two Levels: Syntax and Reduction Rules

<i>Names:</i> $n, h, \dots \in \mathbf{N}$					
<i>Systems:</i>		<i>Processes:</i>		<i>Capabilities:</i>	
$M, N ::= \mathbf{0}$	nil system	$P, Q, R ::= \mathbf{0}$	nil process	$C ::= \text{in}_n$	may enter into n
$M_1 \mid M_2$	parallel	$P_1 \mid P_2$	parallel	out_n	may exit out of n
$(\nu n)M$	restriction	$(\nu n)P$	restriction	open_n	may open n
$n[M]$	ambient	$n[P]$	ambient		
		$C.P$	prefixing		
		$!C.P$	replication		
<i>Reduction Rules:</i>					
$n[\text{in}_m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$			$m[n[\text{out}_m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$		
$\text{open}_n.n.P \mid n[Q] \rightarrow P \mid Q$			$P \equiv Q \quad Q \rightarrow R \quad R \equiv S \text{ implies } P \rightarrow S$		

proof techniques, along the lines of [18,22,25]. More precisely, we develop both *up-to expansion* and *up-to context* proof techniques and prove their soundness. We are not aware of other forms of up-to proof techniques for higher-order calculi. Finally, we apply our bisimulation proof methods to prove a collection of *algebraic laws* (including the perfect firewall equation); we also prove the correctness of the protocol, introduced in [4], for controlling access through a firewall.

The treatment of *communication* is only outlined; however, in [15], the above mentioned results are smoothly extended to the calculus with communication.

The paper ends with a comparison with related work. For lack of space proofs are sketched or omitted; full proofs can be found in [15].

1 Mobile Ambients in Two Levels

In Table 1 we report the syntax of MA, where $\mathbf{N}$ denotes an infinite set of names. Unlike other definitions of MA in the literature, our syntax is defined in a two-level structure, a lower one for *processes*, and an upper one for *systems*.

The syntax for processes is standard, [4], except for replication that is replaced by replicated prefixing, $!C.P$. As in the π -calculus, replicated prefixing allows us to derive a simpler LTS; theory and results in this paper could be easily adapted to a calculus with full replication. A system is a collection of ambients running in parallel, where the knowledge of certain ambient names may be restricted among two or more ambients. We use a number of notational conventions. Parallel composition has the lowest precedence among the operators. The process $C.C'.P$ is read as $C.(C'.P)$. We omit trailing dead processes, writing C for $C.\mathbf{0}$, and $n[]$ for $n[\mathbf{0}]$. Restriction $(\nu n)P$ acts as binder for name n , and the set of *free names* of P , $\text{fn}(P)$, is defined accordingly.

A *static context* is a process context where the hole does not appear under a prefix or a replication. The dynamics of the calculus is specified by a *reduction relation*, $\rightarrow$, which is the least relation over processes closed under static contexts and satisfying the rules in Table 1. As systems are processes with a special structure, the rules of Table 1 also describe the evolution of systems. The *reduction semantics* relies on an auxiliary relation called *structural congruence* that

brings the participants of a potential interaction into contiguous positions. We refer to [4] for the definition of structural congruence, $\equiv$. It is easy to check that systems always reduce to systems.

We introduce a basic equivalence by considering natural, desirable, properties. We choose a generalisation of the *reduction barbed congruence*, [12], a *contextual*, *reduction closed*, and *barb preserving* equivalence relation. We now explain what these properties mean.

A *system context* is a context generated by the following grammar:

$$C[-] ::= - \mid C[-] \mid M \mid (\nu n)C[-] \mid n[C[-] \mid P]$$

where M is an arbitrary system, and P is an arbitrary process. A relation $\mathcal{R}$ over systems is *contextual* if $M \mathcal{R} N$ implies $C[M] \mathcal{R} C[N]$ for all system contexts $C[-]$.

A relation $\mathcal{R}$ over systems is *reduction closed* if whenever $M \mathcal{R} N$ and $M \rightarrow M'$ there is some N' such that $N \rightarrow^* N'$ and $M' \mathcal{R} N'$, where $\rightarrow^*$ denotes the reflexive and transitive closure of $\rightarrow$.

In Mobile Ambients the observation predicate $M \downarrow_n$ denotes the possibility of the system M interacting with the environment via the ambient n . We write $M \downarrow_n$ if $M \equiv (\nu \tilde{m})(n[P] \mid M')$ with $n \notin \{\tilde{m}\}$. We write $M \Downarrow_n$ if there exists M' such that $M \rightarrow^* M'$ and $M' \downarrow_n$. A relation $\mathcal{R}$ over systems is *barb preserving* if $M \mathcal{R} N$ and $M \downarrow_n$ implies $N \downarrow_n$.

Definition 1 (Reduction barbed congruence). *Reduction barbed congruence, written $\cong$, is the largest symmetric relation over systems which is reduction closed, contextual, and barb preserving.*

2 A Labelled Transition Semantics

In our language, the prefixes C give rise to transitions of the form $P \xrightarrow{C} Q$; for example we have

$$\text{in}_n.P_1 \mid P_2 \xrightarrow{\text{in}_n} P_1 \mid P_2.$$

However, similarly to [14], capabilities induce different and more complicated actions. The LTS is defined over processes, although in the labelled bisimilarity we only consider actions going from systems to systems. We make a distinction between *pre-actions* and *env-actions*: the former denote the possibility to exercise certain capabilities whereas the latter model the interaction of a system with its environment. As usual, we also have τ -actions to model internal computations. Only env-actions and τ -actions model the evolution of a system at run-time.

The pre-actions, defined in Table 3, are of the form $P \xrightarrow{\pi} O$ where the ranges of π and of O , the *outcomes*, are reported in Table 2. An outcome may be a simple process Q , if for example π is a prefix of the language, or a *concretion*, of the form $(\nu \tilde{m})(P)Q$, when an ambient boundary is somehow involved. In this case, intuitively, P represents the part of the system affected by the action while

Table 2. Pre-actions, Env-actions, Actions, Concretions, and Outcomes

<i>Pre-actions:</i> $\pi ::= \text{in}_n \mid \text{out}_n$		<i>Env-actions:</i> $\sigma ::= k.\text{enter}_n \mid k.\text{exit}_n$	
$\mid \text{open}_n$	$\mid \text{enter}_n$	$\mid *. \text{enter}_n$	$\mid *. \text{exit}_n$
$\mid \text{amb}_n$	$\mid \text{exit}_n$	$\mid n.\overline{\text{enter}}_k$	$\mid k.\text{open}_n$
<i>Outcomes:</i> $O ::= P \mid K$		<i>Concretions:</i> $K ::= (\nu \tilde{m})\langle P \rangle Q$	<i>Actions:</i> $\alpha ::= \sigma \cup \tau$

Table 3. Labelled Transition System - Pre-actions

$(\pi \text{ Pfx}) \frac{-}{\pi.P \xrightarrow{\pi} P}$	$(\pi \text{ Repl Pfx}) \frac{-}{!\pi.P \xrightarrow{\pi} P \mid !\pi.P}$	$(\pi \text{ Exit}) \frac{P \xrightarrow{\text{out}_n} P_1}{m[P] \xrightarrow{\text{exit}_n} \langle m[P_1] \rangle \mathbf{0}}$
$(\pi \text{ Amb}) \frac{-}{n[P] \xrightarrow{\text{amb}_n} \langle P \rangle \mathbf{0}}$	$(\pi \text{ Enter}) \frac{P \xrightarrow{\text{in}_n} P_1}{m[P] \xrightarrow{\text{enter}_n} \langle m[P_1] \rangle \mathbf{0}}$	$(\pi \text{ Res}) \frac{P \xrightarrow{\pi} O \quad n \notin \text{fn}(\pi)}{(\nu n)P \xrightarrow{\pi} (\nu n)O}$
	$(\pi \text{ Par}) \frac{P \xrightarrow{\pi} O \quad Q \mid P \xrightarrow{\pi} Q \mid O}{P \mid Q \xrightarrow{\pi} O \mid Q}$	

Q is not, and $\tilde{m}$ is the set of private names shared by P and Q . We adopt the convention that if K is the concretion $(\nu \tilde{m})\langle P \rangle Q$, then $(\nu r)K$ is a shorthand for $(\nu \tilde{m})\langle P \rangle (\nu r)Q$, if $r \notin \text{fn}(P)$, and the concretion $(\nu r \tilde{m})\langle P \rangle Q$ otherwise. We have a similar convention for the rule $(\pi \text{ Par})$: $K \mid R$ is defined to be the concretion $(\nu \tilde{m})\langle P \rangle (Q \mid R)$, where $\tilde{m}$ are chosen, using α -conversion if necessary, so that $\text{fn}(R) \cap \{\tilde{m}\} = \emptyset$. Moreover, $(\nu \tilde{m})\langle P \rangle (\mathbf{0} \mid R)$ is abbreviated by $(\nu \tilde{m})\langle P \rangle R$.

The τ -actions, defined in Table 4, model the internal evolution of processes. The env-actions, defined in Table 5, are of the form $M \xrightarrow{\sigma} M'$, where the range of σ is given in Table 2. In practice, env-actions turn concretions into running systems by explicitly introducing the environment's ambient interacting with the process being considered. The content of this ambient is instantiated later, in the bisimilarity, with a process. For convenience, we extend the syntax of processes with the special process $\circ$ to pinpoint those ambients whose content will be instantiated later. The process $\circ$ does not reduce: it is simply a placeholder. Note that, unlike pre-actions and τ -actions, env-actions do not have structural rules; this is because env-actions are supposed to be performed by systems that can directly interact with the environment.

We call *actions* the set of env-actions extended with τ . As our bisimilarity will be defined over systems, we will only consider actions (and not pre-actions) in its definition.

Proposition 1. *If T is a system (resp. a process), and $T \xrightarrow{\alpha} T'$, then T' is a system (resp. a process), possibly containing the special process $\circ$.*

Table 4. Labelled Transition System - τ -actions

$(\tau \text{ Enter}) \frac{P \xrightarrow{\text{enter}.n} (\nu \tilde{p}) \langle P_1 \rangle P_2 \quad Q \xrightarrow{\text{amb}.n} (\nu \tilde{q}) \langle Q_1 \rangle Q_2^{(*)}}{P \mid Q \xrightarrow{\tau} (\nu \tilde{p}) (\nu \tilde{q}) (n[P_1 \mid Q_1] \mid P_2 \mid Q_2)}$ $Q \mid P \xrightarrow{\tau} (\nu \tilde{q}) (\nu \tilde{p}) (n[Q_1 \mid P_1] \mid Q_2 \mid P_2)$	$(\tau \text{ Par}) \frac{P \xrightarrow{\tau} P' \quad Q \xrightarrow{\tau} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$
$(\tau \text{ Exit}) \frac{P \xrightarrow{\text{exit}.n} (\nu \tilde{m}) \langle k[P_1] \rangle P_2}{n[P] \xrightarrow{\tau} (\nu \tilde{m}) (k[P_1] \mid n[P_2])}$	$(\tau \text{ Amb}) \frac{P \xrightarrow{\tau} Q}{n[P] \xrightarrow{\tau} n[Q]}$
$(\tau \text{ Open}) \frac{P \xrightarrow{\text{open}.n} P_1 \quad Q \xrightarrow{\text{amb}.n} (\nu \tilde{m}) \langle Q_1 \rangle Q_2}{P \mid Q \xrightarrow{\tau} P_1 \mid (\nu \tilde{m}) (Q_1 \mid Q_2)}$ $Q \mid P \xrightarrow{\tau} (\nu \tilde{m}) (Q_1 \mid Q_2) \mid P_1$	$(\tau \text{ Res}) \frac{P \xrightarrow{\tau} P'}{(\nu n)P \xrightarrow{\tau} (\nu n)P'}$

(*) We require $((\text{fn}(P_1) \cup \text{fn}(P_2)) \cap \{\tilde{q}\}) = ((\text{fn}(Q_1) \cup \text{fn}(Q_2)) \cap \{\tilde{p}\}) = \emptyset$

Table 5. Labelled Transition System - Env-actions

$(\text{Enter}) \frac{P \xrightarrow{\text{enter}.n} (\nu \tilde{m}) \langle k[P_1] \rangle P_2^{(\dagger)}}{P \xrightarrow{k.\text{enter}.n} (\nu \tilde{m}) (n[\circ \mid k[P_1]] \mid P_2)}$	$(\text{Exit}) \frac{P \xrightarrow{\text{exit}.n} (\nu \tilde{m}) \langle k[P_1] \rangle P_2^{(\dagger)}}{P \xrightarrow{k.\text{exit}.n} (\nu \tilde{m}) (k[P_1] \mid n[\circ \mid P_2])}$
$(\text{Co-Enter}) \frac{P \xrightarrow{\text{amb}.n} (\nu \tilde{m}) \langle P_1 \rangle P_2^{(\dagger)}}{P \xrightarrow{n.\text{enter}.k} (\nu \tilde{m}) (n[P_1 \mid k[\circ]] \mid P_2)}$	$(\text{Open}) \frac{P \xrightarrow{\text{amb}.n} (\nu \tilde{m}) \langle P_1 \rangle P_2}{P \xrightarrow{k.\text{open}.n} k[\circ \mid (\nu \tilde{m}) (P_1 \mid P_2)]}$
$(\text{Enter Shh}) \frac{P \xrightarrow{\text{enter}.n} (\nu \tilde{m}) \langle k[P_1] \rangle P_2^{(\ddagger)}}{P \xrightarrow{*.\text{enter}.n} (\nu \tilde{m}) (n[\circ \mid k[P_1]] \mid P_2)}$	$(\text{Exit Shh}) \frac{P \xrightarrow{\text{exit}.n} (\nu \tilde{m}) \langle k[P_1] \rangle P_2^{(\ddagger)}}{P \xrightarrow{*.\text{exit}.n} (\nu \tilde{m}) (k[P_1] \mid n[\circ \mid P_2])}$

($\dagger$) We require $k \notin \tilde{m}$. ($\ddagger$) We require $k \neq n$ and $k \in \tilde{m}$

We explain the rules induced by the the prefix **in**, the *immigration* of ambients. A typical example of an ambient m migrating into an ambient n follows:

$$(\nu m)(m[\text{in}.n.P_1 \mid P_2] \mid M) \mid n[Q] \rightarrow (\nu m)(M \mid n[m[P_1 \mid P_2] \mid Q])$$

The driving force behind the migration is the activation of the prefix **in**. n , within the ambient m . It induces a capability in the ambient m to migrate into n , that we formalise as a new action **enter**. n . Thus, an application of $(\pi \text{ Enter})$ gives

$$m[\text{in}.n.P_1 \mid P_2] \xrightarrow{\text{enter}.n} \langle m[P_1 \mid P_2] \rangle \mathbf{0}$$

and more generally, using the structural rules $(\pi \text{ Res})$ and $(\pi \text{ Par})$,

$$(\nu m)(m[\text{in}.n.P_1 \mid P_2] \mid M) \xrightarrow{\text{enter}.n} (\nu m) \langle m[P_1 \mid P_2] \rangle M.$$

This means that the ambient $m[\text{in}.n.P_1 \mid P_2]$ has the capability to enter an ambient n ; if the capability is exercised, the ambient $m[P_1 \mid P_2]$ will enter n , while M will be the residual at the original location. Of course, the action can only be executed if there is an ambient n in parallel. The rule $(\pi \text{ Amb})$ allows to check for the presence of ambients. So we have

$$n[Q] \xrightarrow{\text{amb}.n} \langle Q \rangle \mathbf{0}.$$

Here, the concretion $\langle Q \rangle \mathbf{0}$ says that Q is in n , while $\mathbf{0}$ is outside. Finally, the communication $(\tau \text{ Enter})$ allows these two complementary actions to occur simultaneously, executing the migration of the ambient $m[P_1 \mid P_2]$ from its current computational space into the ambient n , giving rise to the original move above:

$$(\nu m)(m[\text{in}_n.P_1 \mid P_2] \mid M) \mid n[Q] \xrightarrow{\tau} (\nu m)(M \mid n[m[P_1 \mid P_2] \mid Q]).$$

Env-actions model the interaction of mobile agents with their environment. For instance, using the rule (Enter Shh), we derive from

$$(\nu m)(m[\text{in}_n.P_1 \mid P_2] \mid M) \xrightarrow{\text{enter}_n} (\nu m)\langle m[P_1 \mid P_2] \rangle M$$

the transition

$$(\nu m)(m[\text{in}_n.P_1 \mid P_2] \mid M) \xrightarrow{*.\text{enter}_n} (\nu m)(n[\circ \mid m[P_1 \mid P_2]] \mid M).$$

This transition denotes a private (*secret*) ambient entering an ambient n provided by the environment. The computation running at n can be specified later by instantiating the placeholder $\circ$. Had the ambient name m not been restricted, we would have used the rule (Enter) to derive

$$m[\text{in}_n.P_1 \mid P_2] \mid M \xrightarrow{m.\text{enter}_n} n[\circ \mid m[P_1 \mid P_2]] \mid M$$

to model a public ambient m that enters an ambient n provided by the environment.

The rules for *emigration* and *opening* follow the same lines. Finally, whenever a system offers a public ambient n at top-level, a context can interact with the system by providing an ambient entering n . The rule Co-Enter captures this interaction between system and environment.

The LTS based semantics coincides with the reduction semantics of Section 1.

Theorem 1. *If $P \xrightarrow{\tau} P'$ then $P \rightarrow P'$. If $P \rightarrow P'$ then $P \xrightarrow{\tau} \equiv P'$.*

Remark 1. From the result above, it is easy to establish that if $M \cong N$ then (i) $M \Downarrow n$ iff $N \Downarrow n$ and (ii) $M \Rightarrow M'$ implies there is N' such that $N \Rightarrow N'$ and $M' \cong N'$. In the sequel we will use these properties without comment.

3 Characterising Reduction Barbed Congruence

In this section we define a labelled bisimilarity for MA that completely characterises reduction barbed congruence.

Since we are interested in *weak bisimilarities*, that abstract over τ -actions, we introduce the notion of weak action. The definition is standard: $\Rightarrow$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}$; $\xRightarrow{\alpha}$ denotes $\Rightarrow \xrightarrow{\alpha} \Rightarrow$; $\xRightarrow{\hat{\alpha}}$ denotes $\Rightarrow$ if $\alpha = \tau$ and $\xRightarrow{\alpha}$ otherwise.

In the previous section we said that actions (and more precisely env-actions) introduce a special process $\circ$ to pinpoint those ambients whose content will be specified in the bisimilarity. The $\bullet$ operator instantiates the placeholder with a process, as defined below.

Definition 2. Let T and T_i be either systems or processes. Then, for a process P , we define:

$$\begin{array}{lll}
 0 \bullet P & \stackrel{\text{def}}{=} & 0 \\
 n[R] \bullet P & \stackrel{\text{def}}{=} & n[R \bullet P] \\
 !C.R \bullet P & \stackrel{\text{def}}{=} & !C.(R \bullet P)
 \end{array}
 \qquad
 \begin{array}{lll}
 (T_1 \mid T_2) \bullet P & \stackrel{\text{def}}{=} & (T_1 \bullet P) \mid (T_2 \bullet P) \\
 (\nu n)T \bullet P & \stackrel{\text{def}}{=} & (\nu n)(T \bullet P) \text{ if } n \notin \text{fn}(P) \\
 C.R \bullet P & \stackrel{\text{def}}{=} & C.(R \bullet P)
 \end{array}
 \qquad
 \circ \bullet P \stackrel{\text{def}}{=} P$$

Everything is in place to define our bisimilarity over systems.

Definition 3 (Bisimilarity). A symmetric relation $\mathcal{R}$ is a bisimulation if $M \mathcal{R} N$ implies:

- if $M \xrightarrow{\alpha} M'$, $\alpha \notin \{*\text{.enter}_n, *\text{.exit}_n\}$, then there is a system N' such that $N \xrightarrow{\hat{\alpha}} N'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$;
- if $M \xrightarrow{*\text{.enter}_n} M'$ then there is a system N' such that $N \mid n[\circ] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$;
- if $M \xrightarrow{*\text{.exit}_n} M'$ then there is a system N' such that $n[\circ \mid N] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$.

Systems M and N are bisimilar, written $M \approx N$, if $M \mathcal{R} N$ for some bisimulation $\mathcal{R}$.

The bisimilarity above has a universal quantification over the process P provided by the environment. This process instantiates the special process $\circ$ generated via env-actions. The bisimilarity is defined in a *late* style as the existential quantification precedes the universal one. Another possibility would be to define the bisimilarity in *early* style where the universal quantification over the environment's contribution P precedes that over the derivative N' . In [15] we prove that, as in $\text{HO}\pi$ [21], late and early bisimilarity coincide.

Finally, notice that actions $*\text{.enter}_n$ and $*\text{.exit}_n$ are treated apart asking for weaker matching requirements. This is because both actions are not observable. Somehow, this is very similar to what happens with input actions in the asynchronous π -calculus [1].

We prove that $\approx$ is a proof techniques for reduction barbed congruence.

Theorem 2. *Bisimilarity is contextual and is an equivalence relation*

Notice that when proving transitivity we use the fact that $\approx$ is preserved by parallel composition and ambient nesting. These two properties do not rely on the transitivity of $\approx$, and are necessary to deal with the env-actions $*\text{.enter}_n$ and $*\text{.exit}_n$.

We can finally state our soundness result.

Theorem 3 (Soundness). *Bisimilarity is contained in reduction barbed congruence.*

Proof By Theorem 2 and the fact that bisimilarity is reduction closed and barb-preserving. $\square$

Table 6. Contexts for visible actions

$\alpha = k.\text{enter}_n$	$C_\alpha[-] = n[\circ \mid \text{done}[\text{in}_k.\text{out}_k.\text{out}_n]] \mid -$
$\alpha = k.\text{exit}_n$	$C_\alpha[-] = (\nu a)a[\text{in}_k.\text{out}_k.\text{done}[\text{out}_a]] \mid n[\circ \mid -]$
$\alpha = n.\overline{\text{enter}}_k$	$C_\alpha[-] = (\nu a)a[\text{in}_n.k[\text{out}_a.(\circ \mid (\nu b)b[\text{out}_k.\text{out}_n.\text{done}[\text{out}_b]]])] \mid -$
$\alpha = k.\text{open}_n$	$C_\alpha[-] = k[\circ \mid (\nu a, b)(\text{open}.b.\text{open}.a.\text{done}[\text{out}_k] \mid a[- \mid \text{open}.n.b[\text{out}_a]])]$ where a and b are fresh.

We now prove that $\approx$ is more than a proof techniques. It actually characterises the reduction barbed congruence. The main challenge here is to design the contexts capable to observe our visible actions. The definition of these contexts, $C_\alpha[-]$, for every visible action α , is given in Table 6. The special ambient name **done** is used as *fresh* barb to signal the consumption of actions.

To prove our characterisation result it suffices to show that reduction barbed congruence is contained in bisimilarity. At this end we must prove a correspondence between visible actions α and their corresponding contexts $C_\alpha[-]$.

The following lemma says that the defining contexts are sound, that is, they can successfully mimic the execution of visible actions.

Lemma 1. *Let M be a system, and let $\alpha \in \{k.\text{enter}_n, k.\text{exit}_n, n.\overline{\text{enter}}_k, k.\text{open}_n\}$. For all processes P , if $M \xrightarrow{\alpha} M'$, then $C_\alpha[M] \bullet P \Rightarrow \cong M' \bullet P \mid \text{done}[]$.*

To complete the correspondence proof between actions α and their contexts $C_\alpha[-]$, we have to prove the converse of Lemma 1, formalised in Lemma 2. Such result requires a few technical definitions given in Table 7. The symbol $\oplus$ denotes a form of internal choice, whereas the context $\text{SPY}_\alpha\langle i, j, - \rangle$ is a technical tool to guarantee that the process P provided by the environment does not perform any action. This is essential to the proof of the completeness result because it guarantees that the contribution of P is the same on both sides. The ability of $\text{SPY}_\alpha\langle i, j, P \rangle$ to “spy” on P stems from the fact that one of the two fresh barbs i and j is lost when P performs any action.

Lemma 2. *Let M be a system, let $\alpha \in \{k.\text{enter}_n, k.\text{exit}_n, n.\overline{\text{enter}}_k, k.\text{open}_n\}$, and let i, j be fresh names for M . For all processes P with $\{i, j\} \cap \text{fn}(P) = \emptyset$, if $C_\alpha[M] \bullet \text{SPY}_\alpha\langle i, j, P \rangle \Rightarrow \cong O \mid \text{done}[]$ and $O \Downarrow_{i,j}$, then there exists a system M' such that $O \cong M' \bullet \text{SPY}_\alpha\langle i, j, P \rangle$ and $M \xRightarrow{\alpha} M'$.*

Theorem 4 (Completeness). *Reduction barbed congruence is contained in bisimilarity.*

Proof [Sketch] We prove that the relation $\mathcal{R} = \{(M, N) \mid M \cong N\}$ is a late bisimulation. Suppose $M \mathcal{R} N$ and $M \xrightarrow{\alpha} M'$. Suppose also $\alpha \in \{k.\text{enter}_n, k.\text{exit}_n, n.\overline{\text{enter}}_k, k.\text{open}_n\}$. We must find a system N' such that $N \xRightarrow{\alpha} N'$ and for all P , $M' \bullet P \cong N' \bullet P$.

Table 7. Auxiliary contexts and processes

$-_1 \oplus -_2$	$= (\nu o)(o[] \mid \text{open}_{o.o}._1 \mid \text{open}_{o.o}._2)$
$\text{SPY}_\alpha\langle i, j, - \rangle = (i[\text{out}_{.n}] \mid -) \oplus (j[\text{out}_{.n}] \mid -)$ if $\alpha \in \{k.\text{enter}_{.n}, k.\text{exit}_{.n}, k.\text{open}_{.n}, *. \text{enter}_{.n}, *. \text{exit}_{.n}\}$	
$\text{SPY}_\alpha\langle i, j, - \rangle = (i[\text{out}_{.k}.\text{out}_{.n}] \mid -) \oplus (j[\text{out}_{.k}.\text{out}_{.n}] \mid -)$ if $\alpha \in \{n.\overline{\text{enter}}_{.k}\}$	

The idea of the proof is to use a particular context which mimics the effect of the action α , and also allows us to subsequently compare the residuals of the two systems. This context has the form

$$D_\alpha\langle P \rangle[-] = (C_\alpha[-] \mid \text{Flip}) \bullet \text{SPY}_\alpha\langle i, j, P \rangle$$

where $C_\alpha[-]$ are the contexts in Table 6 and Flip is the system:

$$(\nu k)k[\text{in}_{\text{done}}.\text{out}_{\text{done}}.(\text{succ}[\text{out}_{.k}] \oplus \text{fail}[\text{out}_{.k}])]$$

with succ and fail fresh names. Intuitively, the existence of the fresh barb fail indicates that the action α has not yet happened, whereas the presence of succ together with the absence of fail ensures that the action α has been performed, and has been reported via done .

As $\cong$ is contextual, $M \cong N$ implies that, for all processes P , it holds $D_\alpha\langle P \rangle[M] \cong D_\alpha\langle P \rangle[N]$. By Lemma 1, and by inspecting the reduction of the Flip process, we observe that:

$$D_\alpha\langle P \rangle[M] \Rightarrow \cong M' \bullet \text{SPY}_\alpha\langle i, j, P \rangle \mid \text{done}[] \mid \text{Flip} \Rightarrow \cong M' \bullet \text{SPY}_\alpha\langle i, j, P \rangle \mid \text{done}[] \mid \text{succ}[]$$

where $M' \bullet \text{SPY}_\alpha\langle i, j, P \rangle \mid \text{done}[] \mid \text{succ}[] \Downarrow_{i,j,\text{succ}} \not\Downarrow_{\text{fail}}$. Call this outcome O_1 . This reduction must be matched by a corresponding reduction $D_\alpha\langle P \rangle[N] \Rightarrow O_2$, where $O_1 \cong O_2$. However, the possible matching reductions are constrained by the barbs of O_1 , because it must hold $O_2 \Downarrow_{i,j,\text{succ}} \not\Downarrow_{\text{fail}}$.

As $O_2 \Downarrow_{\text{succ}} \not\Downarrow_{\text{fail}}$, it must be $O_2 \cong \hat{N} \mid \text{done}[] \mid \text{succ}[]$, for some systems $\hat{N}$. As $O_2 \Downarrow_{i,j}$, the previous observation can be combined with Lemma 2 to derive the existence of a system (over the extended process syntax) N' such that $\hat{N} \cong N' \bullet \text{SPY}_\alpha\langle i, j, P \rangle$ and a weak action $N \xRightarrow{\alpha} N'$.

To conclude we must establish that for all P , it holds $M' \bullet P \cong N' \bullet P$. As barbed congruence is preserved by restriction, we have $(\nu \text{done}, \text{succ})O_1 \cong (\nu \text{done}, \text{succ})O_2$. As $(\nu \text{done})\text{done}[] \cong (\nu \text{succ})\text{succ}[] \cong \mathbf{0}$, it follows that $M' \bullet \text{SPY}_\alpha\langle i, j, P \rangle \cong N' \bullet \text{SPY}_\alpha\langle i, j, P \rangle$. Again, $\cong$ is preserved by restriction and $(\nu i, j)\text{SPY}_\alpha\langle i, j, P \rangle \cong P$. So, we can finally derive $M' \bullet P \mathcal{R} N' \bullet P$, for all processes P .

To complete the proof we need to consider the actions $*. \text{enter}_{.n}$ and $*. \text{exit}_{.n}$: as they are not observable, these cases are much easier. $\square$

By Theorems 3 and 4 we conclude that *bisimilarity and reduction barbed congruence coincide*.

Synchronous communication of capabilities can be added to MA: the *output process*, $\langle E \rangle.P$, outputs the message E and then continues as P , and the input process $(x).Q$ receives a message and binds it to x in Q , which then executes. As discussed in [28,23], synchrony is not unrealistic because communication in MA is always local. Our LTS needs to be extended with rules analogues to the rules that deal with communication in [14]. The proof of Theorem 1 can be easily completed. More interestingly, in our framework, communication capabilities cannot be observed at top-level: this in turn implies that our bisimulations can be applied to the extended calculus, and all the results of Section 3 and Section 4 hold without modifications.

4 Up-to Proof Techniques

In this section we adapt well-known *up-to* proof techniques [18,22] to our setting. These techniques allow us to reduce the size of the relation to exhibit to prove that two processes are bisimilar. We focus on the *up-to expansion* [24] and the *up-to context* techniques [22]. As in π -calculus, these can be merged: for lack of space we only report the resulting technique.

Roughly, the expansion, written $\lesssim$, is an asymmetric variant of the bisimilarity that allows us to count the number of silent moves performed by a process.

Definition 4 (Expansion). *A relation $\mathcal{R}$ over systems is an expansion if $M \mathcal{R} N$ implies:*

- if $M \xrightarrow{\alpha} M'$, $\alpha \notin \{*.enter\_n, *.exit\_n\}$, then there exists a system N' such that $N \xrightarrow{\hat{\alpha}} N'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$;
- if $M \xrightarrow{*.enter\_n} M'$ then there is a system N' such that $N \mid n[\circ] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$;
- if $M \xrightarrow{*.exit\_n} M'$ then there is a system N' such that $n[\circ \mid N] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$;
- if $N \xrightarrow{\alpha} N'$, $\alpha \notin \{*.enter\_n, *.exit\_n\}$, then there exists a system M' such that $M \xrightarrow{\hat{\alpha}} M'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$;
- if $N \xrightarrow{*.enter\_n} N'$ then $(M \mid n[P]) \mathcal{R} N' \bullet P$, for all processes P ;
- if $N \xrightarrow{*.exit\_n} N'$ then $n[M \mid P] \mathcal{R} N' \bullet P$, for all processes P .

We write $M \lesssim N$, if $M \mathcal{R} N$ for some expansion $\mathcal{R}$.

Definition 5 (Bisimulation up to context and up to $\gtrsim$). *A symmetric relation $\mathcal{R}$ is a bisimulation up to context and up to $\gtrsim$ if $P \mathcal{R} Q$ implies:*

- if $M \xrightarrow{\alpha} M''$, $\alpha \notin \{*.enter\_n, *.exit\_n\}$, then there exists a system N'' such that $N \xrightarrow{\hat{\alpha}} N''$, and for all processes P there is a system context $C[-]$ and systems M' and N' such that $M'' \bullet P \gtrsim C[M']$, $N'' \bullet P \gtrsim C[N']$, and $M' \mathcal{R} N'$;

- if $M \xrightarrow{*.\text{enter}_n} M''$ then there exists a system N'' such that $N \mid n[\circ] \Rightarrow N''$, and for all processes P there is a system context $C[-]$ and systems M' and N' such that $M'' \bullet P \gtrsim C[M']$, $N'' \bullet P \gtrsim C[N']$, and $M' \mathcal{R} N'$;
- if $M \xrightarrow{*.\text{exit}_n} M''$ then there exist a system N'' such that $n[\circ \mid N] \Rightarrow N''$, and for all processes P there is a system context $C[-]$ and systems M' and N' such that $M'' \bullet P \gtrsim C[M']$, $N'' \bullet P \gtrsim C[N']$, and $M' \mathcal{R} N'$.

Theorem 5. If $\mathcal{R}$ is a bisimulation up to context and up to $\gtrsim$, then $\mathcal{R} \subseteq \approx$.

5 Algebraic Theory

Here we prove a collection of algebraic laws using our bisimulation proof methods, and the correctness of a protocol for controlling access through a *firewall*, first proposed in [4]. We recall that M, N range over systems and P, Q, R over processes. The first two laws are examples of local communication within private ambients without interference. The third law is the well-known perfect firewall law. The following four laws represent non-interference properties about movements of private ambients. Finally, the last two laws say when opening cannot be interfered.

Theorem 6.

1. $(\nu n)n[\langle W \rangle.P \mid (x).Q \mid M] \cong (\nu n)n[P \mid Q\{^W/x\} \mid M]$ if $n \notin \text{fn}(M)$
2. $(\nu n)n[\langle W \rangle.P \mid (x).Q \mid \prod_{j \in J} \text{open}_{k_j}.R_j] \cong (\nu n)n[P \mid Q\{^W/x\} \mid \prod_{j \in J} \text{open}_{k_j}.R_j]$
3. $(\nu n)n[P] \cong \mathbf{0}$ if $n \notin \text{fn}(P)$
4. $(\nu n)((\nu m)m[\text{in}_n.P] \mid n[M]) \cong (\nu n)n[(\nu m)m[P] \mid M]$ if $n \notin \text{fn}(M)$
5. $(\nu m, n)(m[\text{in}_n.P] \mid n[\prod_{j \in J} \text{open}_{k_j}.R_j]) \cong (\nu m, n)n[m[P] \mid \prod_{j \in J} \text{open}_{k_j}.R_j]$
6. $(\nu n)n[(\nu m)m[\text{out}_n.P] \mid M] \cong (\nu n)((\nu m)m[P] \mid n[M])$ if $n \notin \text{fn}(M)$
7. $(\nu n)n[m[\text{out}_n.P] \mid \prod_{j \in J} \text{open}_{k_j}.R_j] \cong (\nu n)(m[P] \mid n[\prod_{j \in J} \text{open}_{k_j}.R_j])$ if $m \neq k_j$, for $j \in J$
8. $n[(\nu m)(\text{open}_m.P \mid m[N]) \mid Q] \cong n[(\nu m)(P \mid N) \mid Q]$ if $Q \equiv M \mid \prod_{j \in J} \langle W_j \rangle.R_j$ and $m \notin \text{fn}(N)$
9. $(\nu n)n[(\nu m)(\text{open}_m.P \mid m[Q]) \mid R] \cong (\nu n)n[(\nu m)(P \mid Q) \mid R]$ if $R \equiv \prod_{i \in I} \langle W_i \rangle.S_i \mid \prod_{j \in J} \text{open}_{k_j}.R_j$ and $m, n \notin \text{fn}(Q)$.

Proof [Sketch] The laws above are proved by exhibiting the appropriate bisimulation, possibly up to context. We illustrate the proof of the law (3). Let $\mathcal{S} = \{((\nu n)n[Q], \mathbf{0}) \mid \forall Q \text{ s.t. } n \notin \text{fn}(Q)\}$. We show that $\mathcal{S}$ is a bisimulation up to context and up to structural congruence. The most delicate cases are those regarding the silent moves $*.\text{enter}_k$ and $*.\text{exit}_k$. For instance, if

$$(\nu n)n[P] \xrightarrow{*.\text{enter}_k} (\nu n)k[\circ \mid n[P']] \equiv k[\circ \mid (\nu n)n[P']] \quad \text{then} \quad \mathbf{0} \mid k[\circ] \Rightarrow \equiv k[\circ \mid \mathbf{0}]$$

and up to context and structural congruence we are still in $\mathcal{S}$. □

Crossing a firewall. Consider the following protocol in MA:

$$AG \stackrel{\text{def}}{=} m[\text{open}_k.(x).x.Q] \qquad FW \stackrel{\text{def}}{=} (\nu w)w[\text{open}_m.P \mid k[\text{out}_w.\text{in}_m.\langle \text{in}_w \rangle]]$$

The ambient w represents the firewall; the ambient m is a trusted agent containing a process Q , supposed to cross the firewall. The firewall ambient sends into the agent a pilot ambient k with the capability in_w for entering the firewall. The agent acquires the capability by opening k . The process Q carried by the agent is finally liberated inside the firewall by the opening of ambient m . Names m and k act like passwords that guarantee the access only to authorised agents. The correctness (of a slight variant) of the protocol above is shown in [4] for may-testing [6] proving that $(\nu m, k)(AG \mid FG) \cong (\nu w)w[Q \mid P]$ under the conditions that $w \notin \text{fn}(Q)$, $x \notin \text{fv}(Q)$, $\{m, k\} \cap (\text{fn}(P) \cup \text{fn}(Q)) = \emptyset$. The proof relies on non-trivial contextual reasonings.

The system on the right can be obtained from that one on the left by executing six τ -actions. We prove that $\cong$ is insensitive to all these τ -actions.

Lemma 3. *Let P , Q , and R be processes. Then*

1. $(\nu k, m, w)(k[\text{in}_m.P] \mid m[\text{open}_k.Q] \mid w[\text{open}_m.R]) \cong (\nu k, m, w)(m[k[P] \mid \text{open}_k.Q] \mid w[\text{open}_m.R])$
2. $(\nu m, w)(m[\langle \text{in}_w \rangle \mid (x).P] \mid w[\text{open}_m.Q]) \cong (\nu m, w)(m[P\{\text{in}_w/x\}] \mid w[\text{open}_m.Q])$

Theorem 7. *If $w \notin \text{fn}(Q)$, $x \notin \text{fv}(Q)$, and $\{m, k\} \cap (\text{fn}(P) \cup \text{fn}(Q)) = \emptyset$, then $(\nu m, k)(AG \mid FG) \cong (\nu w)w[Q \mid P]$.*

Proof The result follows from transitivity of $\approx$, because of Law (7) of Theorem 6, Law (1) of Lemma 3, Law (9) of Theorem 6, Law (2) of Lemma 3, and Laws (5) and (9) of Theorem 6. $\square$

6 Related Work

Higher-order LTSs for Mobile Ambients can be found in [3,10,27,7]. But we are not aware of any form of bisimilarity defined using these LTSs. A simple first-order LTS for MA without restriction is proposed by Sangiorgi in [23]. Using this LTS the author defines an *intensional* bisimilarity for MA that separates terms on the basis of their internal structure.

Our work is the natural prosecution of [14] where it was proposed an LTS and a labelled characterisation of reduction barbed congruence for a variant of Levi and Sangiorgi's Safe Ambients, called SAP. The main differences with respect to [14] are the following:

- SAP differs from MA for having co-capabilities and passwords, both features are essential to prove the characterisation result in SAP.

- Our env-actions, unlike those in [14], are truly late, as they do not mention the process provided by the environment. This process can be added *later*, when playing the bisimulation game.
- Our actions for ambient’s movement, unlike those in SAP, report the name of the migrating ambient. For instance, in $k.\text{enter}_n$ we say that ambient k enters n . The knowledge of k is necessary to make the action observable for the environment. This is not needed in SAP, because movements can be observed by means of co-capabilities.
- Co-capabilities also allow the observation of the movement of an ambient whose name is private. As a consequence, the perfect firewall equation does not hold neither in SAP, nor in Safe Ambients. In MA, the movements of an ambient whose name is private cannot be observed. This is why the perfect firewall equation holds.

Apart from [14], other forms of bisimilarity for higher-order calculi, such as Distributed π -calculus [11], Seal [28], Nomadic Pict [26], and a Calculus for Mobile Resources [9], can be found in [16,5,26,9,2], but only [16,9,2] prove labelled characterisations of a contextually defined notion of equivalence. The perfect firewall equation as already been proved for Morris-style contextual equivalence in [10] using a context lemma.

Finally, we believe that interesting labelled characterisations of typed reduction barbed congruence for MA can be derived along the lines of [16], enhancing the algebraic laws of Section 5.

Acknowledgements. Vladimiro Sassone spotted a problem in an early draft of the paper. The anonymous referees contributed useful comments. Francesco Zappa Nardelli is grateful to the Foundations of Computing Group of University of Sussex for the kind hospitality and support. He is partly funded by ‘MyThS: Models and Types for Security in Mobile Distributed Systems’, EU FET-GC IST-2001-32617.

References

1. R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π -calculus. *Theoretical Computer Science*, 195:291–324, 1998.
2. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication interference in mobile boxed ambients. Forthcoming Technical Report. An extended abstract appeared in Proc. FSTTCS’02, LNCS, Springer-Verlag.
3. L. Cardelli and A. Gordon. A commitment relation for the ambient calculus. Unpublished notes, 1996.
4. L. Cardelli and A. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000. An extended abstract appeared in *Proc. of FoSSaCS ’98*.
5. G. Castagna and F. Zappa Nardelli. The seal calculus revisited: Contextual equivalence and bisimilarity. In *Proc. 22nd FSTTCS ’02*, volume 2556 of LNCS. Springer-Verlag, 2002.
6. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

7. G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *Proc. ICTCS*, volume 2202 of *LNCS*, 2001.
8. C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. In *Proc. 25th ICALP*, pages 844–855, 1998.
9. J.C. Godskesen, T. Hildebrandt, and V. Sassone. A calculus of mobile resources. In *Proc. 10th CONCUR '02*, volume 2421 of *LNCS*, 2002.
10. A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. *Journal of Mathematical Structures in Computer Science*, 12:1–38, 2002.
11. M. Hennessy and J. Riely. A typed language for distributed mobile processes. In *Proc. 25th POPL*. ACM Press, 1998.
12. K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.
13. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proc. 27th POPL*. ACM Press, 2000.
14. M. Merro and M. Hennessy. Bisimulation congruences in safe ambients. In *Proc. 29th POPL '02*. ACM Press, 2002.
15. M. Merro and F. Zappa Nardelli. Bisimulation proof methods for mobile ambients. Computer Science Report 2003:01, http://cogslib.cogs.susx.ac.uk/csr_abs.php?cs, University of Sussex, 2003.
16. M. Hennessy M. Merro and J. Rathke. Towards a behavioural theory of access and mobility control in distributed system. To appear in *Proc. 5th FoSSaCS '03*, *LNCS*, 2003, Springer-Verlag.
17. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.
18. R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. 19th ICALP*, volume 623 of *LNCS*, pages 685–695. Springer Verlag, 1992.
19. D.M. Park. Concurrency on automata and infinite sequences. In P. Deussen, editor, *Conf. on Theoretical Computer Science*, volume 104 of *LNCS*. Springer Verlag, 1981.
20. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.
21. D. Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131(2):141–178, 1996.
22. D. Sangiorgi. On the bisimulation proof method. *Journal of Mathematical Structures in Computer Science*, 8:447–479, 1998.
23. D. Sangiorgi. Extensionality and intensionality of the ambient logic. In *Proc. 28th POPL*. ACM Press, 2001.
24. D. Sangiorgi and R. Milner. The problem of “Weak Bisimulation up to”. In *Proc. CONCUR '92*, volume 630 of *LNCS*, pages 32–46. Springer Verlag, 1992.
25. D. Sangiorgi and D. Walker. *The π -calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
26. A. Unyapoth and P. Sewell. Nomadic Pict: Correct communication infrastructures for mobile computation. In *Proc. 28th POPL*. ACM, January 2001.
27. M. G. Vigliotti. Transition systems for the ambient calculus. Master thesis, Imperial College of Science, Technology and Medicine (University of London), September 1999.
28. J. Vitek and G. Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, 1999.

On Equivalent Representations of Infinite Structures

Arnaud Carayol and Thomas Colcombet

IRISA, Campus universitaire de Beaulieu,
35042 Rennes Cedex, France {Arnaud.Carayol, Thomas.Colcombet}@irisa.fr

Abstract. According to Barthelman and Blumensath, the following families of infinite graphs are isomorphic: (1) prefix-recognisable graphs, (2) graph solutions of VR equational systems and (3) MS interpretations of regular trees. In this paper, we consider the extension of prefix-recognisable graphs to prefix-recognisable structures and of graphs solutions of VR equational systems to structures solutions of positive quantifier free definable (PQFD) equational systems. We extend Barthelman and Blumensath's result to structures parameterised by infinite graphs by proving that the following families of structures are equivalent: (1) prefix-recognisable structures restricted by a language accepted by an infinite deterministic automaton, (2) solutions of infinite PQFD equational systems and (3) MS interpretations of the unfoldings of infinite deterministic graphs. Furthermore, we show that the addition of a *fuse* operator, that merges several vertices together, to PQFD equational systems does not increase their expressive power.

1 Introduction

The automatic verification of properties on infinite structures is an important technique for proving behavioural properties on programs. A natural encoding of a program behaviour is an infinite directed graph where vertices are states of the machine, and edges mimic the transition steps of the program. Properties on the program can then be expressed as logical formulas referring to this graph (or its unfolding when considering *e.g.* temporal logics). The problem of model-checking is then to decide the satisfaction of a formula over the graph. This problem is usually undecidable. However, on certain families of infinite graphs and for some given logics the model-checking problem is decidable.

In this work, we are dealing with monadic second-order (MS) logic: an extension of first-order logic which allows quantification over sets of vertices. The first decidability result for this logic over an infinite graph was provided by Büchi for the infinite semi-line. Rabin extended this result to the infinite binary tree.

With the work of Muller and Schupp on pushdown graphs [MS85], the focus of study shifted from infinite graphs to families of infinite graphs.

Since then, many families of graphs have been presented with various decidability and structural properties. Those families can be classified according to their representation into three categories.

The equational representation describes an infinite structure as the solution of an equational system. The family of structures (or graphs) obtained in this way depends on the choice of the operators. The most famous examples are hyperedge replacement equational structures (HR) [Cou89] and the vertex replacement equational graphs (VR) [Cou90]. The VR operators also have been extended into vertex replacement with product operators [Col02].

The transformational representation consists in applying some finite sequence of transformations over an already-known structure. Transformations can be the unfolding of graphs [CW98], Shelah-Muchnik-Walukiewicz tree-like construction [Wal96], or logically defined transformations (FO interpretations, inverse finite or rational mappings [Cau96,Urv02], MS interpretations or general MS-definable transductions [Cou94]).

The internal representation amounts to give an exact description of both the universe and the relations of the structure. The most used universe is the set of words over a given finite alphabet. Relations over words can then be defined by means of many techniques:

Rewriting: Prefix (or suffix) rewriting of words describes the family of push-down graphs [MS85,Cau92]. When the set of rules is recognisable, it leads to prefix-recognisable graphs [Cau96].

Transductions: Relations recognised by synchronised transductions describe the class of automatic graphs [S n92] and structures [Blu99]. When the transduction is rational, it defines the rational graphs [Mor00].

Structures defined over the universe of closed terms have also been presented [DT90,Blu99,L d02,Col02].

The above mentioned techniques are not independent from each other. Many connections have been stated in the literature. In our case we are specially involved with the following: the graphs solution of VR equational systems are isomorphic to prefix-recognisable graphs [Bar97] and to MS interpretations of infinite regular trees [Blu01].

To some extent, these classes of graphs are defined upon *finite* objects. In particular, a VR-equational graph is the solution of a *finite* system of equations and prefix-recognisable graph is a rewriting system restricted to the language accepted by a *finite* automaton. These two kinds of graphs are equivalent and can be obtained by MS-definable transduction of the unfolding of a *finite* graph.

We generalise this triple equivalence to structures defined by *infinite* objects. We show that interpretation of infinite systems of PQFD equations (which is a natural extension of VR operators introduced in [Bar97]), *PR*-structures restricted by infinite deterministic automaton and MS-definable transductions of the unfoldings of deterministic infinite graphs are equivalent. Furthermore, this equivalence is effective in the sense that MS-definable transductions link the system of equations, the automaton and the graph.

In [CM02], the authors prove that for describing sets of finite structures the addition of a *fuse* operator — which merges vertices together — to PQFD-like operators does not increase the expressivity of the considered systems. The authors also emphasizes on how this extension unifies the description of HR-equational et VR-equational graphs. We naturally investigated the infinite counterpart of this result and proved under reasonable technical restrictions that the

addition of a *fuse* operator to PQFD operators does not increase their expressive power. The two results are however technically significantly different.

The rest of the paper is divided as follows. Section 2 introduces the basic definitions. Section 3 presents structures defined by equational systems and Section 4 defines unfolding and states the first inclusion. Section 5 introduces PR-systems and states the last two inclusions.

2 Definitions

Relational Structures

We define the *global signature* Ξ to be equal to $\bigcup_{n \geq 0} \Xi_n$ where Ξ_n is an infinite set of symbols of *arity* n . For any R in Ξ , $|R|$ designates the arity of R .

A *relational structure* $\mathcal{S}$ is a pair $(\mathcal{U}, Val)$ where $\mathcal{U}$ is an ‘at most countable’ set called the universe and Val associates to a symbol of arity n a subset of $\mathcal{U}^n$. We will write $R^{\mathcal{S}}$ instead of $Val(R)$. Moreover, we suppose that Val has a finite support (i.e. the set of R such that $Val(R) \neq \emptyset$ is finite). A *signature* Σ of $\mathcal{S}$ is a *finite* set which contains the support of Val .

The *restriction* of a structure $\mathcal{S} = (\mathcal{U}, Val)$ to a universe $\mathcal{U}' \subseteq \mathcal{U}$ is denoted $\mathcal{S}|_{\mathcal{U}'}$ and designates the structure $(\mathcal{U}', Val')$ where $Val'(R) = Val(R) \cap (\mathcal{U}')^{|R|}$.

Two structures $\mathcal{S}$ and $\mathcal{S}'$ of respective universe $\mathcal{U}$ and $\mathcal{U}'$ are *isomorphic* (written $\mathcal{S} \approx \mathcal{S}'$) if there exists a one to one mapping ρ from $\mathcal{U}$ onto $\mathcal{U}'$ such that for any symbol $R \in \Xi$, $R^{\mathcal{S}}(x_1, \dots, x_n) \Leftrightarrow R^{\mathcal{S}'}(\rho(x_1), \dots, \rho(x_n))$.

Graphs

A *directed graph* G (or simply a *graph*) labelled by a finite set E is a relational structure admitting a signature with binary symbols only (identified with E). The universe is denoted by V and its elements are called vertices. A directed graph is *rooted* if its signatures contain an unary relation *root* which is interpreted as a singleton. By slight abuse, we will use the constant *root* in our formulas. The graph is said to be *deterministic* if for any $x, y, z \in V$ and for any relation $e \in E$, if $e(x, y)$ and $e(x, z)$ then $y = z$.

A *path* π in a graph G labelled by E is a finite sequence $v_1 e_1 \dots e_{n-1} v_n$ in $(VE)^* V$ such that for all $i \in [1, n-1]$, $e_i(v_i, v_{i+1})$. For any $w \in E^*$, we write $x \xRightarrow{w} y$ if there exists a path $v_1 e_1 \dots e_{n-1} v_n$ between x and y such that $w = e_1 \dots e_{n-1}$. For $W \subseteq E^*$ a language, $x \xRightarrow{W} y$ holds iff for some $w \in W$, $x \xRightarrow{w} y$.

Given a graph G labelled by E of universe V and a finite set of *fresh* binary symbols $\mathcal{K} = \{k_1, \dots, k_n\}$ (i.e $\mathcal{K} \cap E = \emptyset$), the $\mathcal{K}$ -copying of G is the graph G' of universe $V \times [0, n]$ and such that for any relation $R \in E$, $R^{G'} = \{((x_1, 0), \dots, (x_{|R|}, 0)) \mid (x_1, \dots, x_{|R|}) \in R^G\}$ and for $k_i \in \mathcal{K}$, $k_i^{G'} = \{((x, 0), (x, i)) \mid x \in V\}$.

Example 1. Throughout this paper we illustrate all the techniques for describing structures with one example: the step-ladder graph depicted in Figure 1.

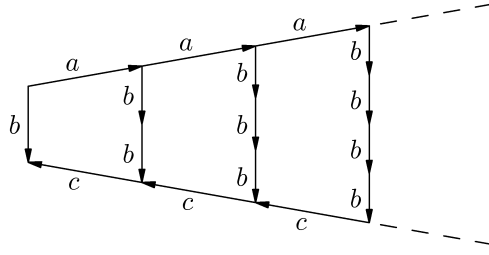


Fig. 1. The step-ladder graph.

Monadic Second Order Logic

In the following, we assume that first order variables are ranged over by $x, y, z \dots$ whereas monadic second order variables are ranged over by $X, Y, Z \dots$. First order variables are interpreted as elements of the universe whereas monadic second order variables are interpreted as subsets of the universe. The atomic predicates of monadic formulas are $x \in X$, $x = y$ and $R(x_1, \dots, x_{|R|})$. Monadic formulas are then inductively defined as $\exists X. \Phi$, $\exists x. \Phi$, $\neg \Phi$ and $\Phi \vee \Psi$ for Φ and Ψ formulas. MS formulas have the usual semantic [Tho97]. If $\Phi(x_1, \dots, x_n)$ is an MS formula and if $(u_1, \dots, u_n)$ is a tuple of elements of $\mathcal{U}$, $\mathcal{S} \models \Phi(u_1, \dots, u_n)$ means that $\mathcal{S}$ models Φ when x_i is interpreted by u_i for all $i \in [1, n]$.

A *MS interpretation* $\mathcal{I}$ is given by a MS formula $\delta(x)$ together with a finite set of formulas $(\Phi_R)_{R \in \Sigma}$ where Φ_R has free variables in $\{x_1, \dots, x_{|R|}\}$. $\mathcal{I}$ associates to each structure $\mathcal{S}$ of universe $\mathcal{U}$ the structure $\mathcal{I}(\mathcal{S})$ of universe $\mathcal{U}_{\mathcal{I}(\mathcal{S})} = \{x \in \mathcal{U} \mid \mathcal{S} \models \delta(x)\}$ and such that if $R \in \Sigma$, $R^{\mathcal{I}(\mathcal{S})} = \{\bar{x} \in (\mathcal{U}')^{|R|} \mid \mathcal{S} \models \Phi_R(\bar{x})\}$ (if $R \notin \Sigma$, $R^{\mathcal{I}(\mathcal{S})} = \emptyset$).

An *MS-definable transduction* $\mathcal{T}$ [Cou94] is the composition of a $\mathcal{K}$ -copying operation and an MS interpretation. This transformation preserves the decidability of the MS theory.

3 Equational Systems

In this section we present how to describe infinite structures as solutions of equational systems over a given set of operators. Classical examples of this approach are hyperedge replacement systems [Cou89] or vertex replacement (VR) [Cou90] graphs.

For the rest of this section, we fix a signature Σ . For $\mathcal{V}$ a given set of variable names, we write $B^+(\mathcal{V})$ the set of positive boolean formulas over variables $\mathcal{V}$. Those formulas are built upon predicates of the signature applied to variables in $\mathcal{V}$, of the boolean connectives $\wedge$ and $\vee$, and the constants **t** (true) and **f** (false). Quantifiers as well as negation are not allowed.

We use the set of symbols $PQFD = PQFD_0 \cup PQFD_1 \cup PQFD_2$ with:

$$\begin{aligned} PQFD_0 &= \{\mathbf{one}\} & PQFD_2 &= \{\oplus\} \\ PQFD_1 &= \{\mathbf{pqfd}[(\phi_R)_{R \in \Sigma}] \mid \forall R \in \Sigma, \phi_R \in B^+(x_1, \dots, x_{|R|})\} \end{aligned}$$

Symbols in $PQFD_i$ have arity i . The mapping $_$ gives their semantic:

Singleton structure one: $\mathcal{U}_{\text{one}} = \{0\}$ and $R^{\text{one}} = \emptyset$ for any symbol R ,

Positive quantifier-free definable interpretation $\text{pqfd}[(\phi_R)]$:

given a relational structure $\mathcal{S}$, $\mathcal{U}_{\text{pqfd}[(\phi_R)](\mathcal{S})} = \mathcal{U}_{\mathcal{S}}$,

and $R^{\text{pqfd}[(\phi_R)](\mathcal{S})}(u_1, \dots, u_{|R|})$ iff $\mathcal{S} \models \phi_R(u_1, \dots, u_{|R|})$,

Disjoint union $\oplus$: given two structures $\mathcal{S}_1$ and $\mathcal{S}_2$,

$\mathcal{U}_{\mathcal{S}_1 \oplus \mathcal{S}_2} = \{1\} \times \mathcal{U}_{\mathcal{S}_1} \cup \{2\} \times \mathcal{U}_{\mathcal{S}_2}$ and,

for any symbol R , $R^{\mathcal{S}_1 \oplus \mathcal{S}_2} = \{((1, x_1), \dots, (1, x_{|R|})) \mid R^{\mathcal{S}_1}(x_1, \dots, x_{|R|})\} \cup \{((2, x_1), \dots, (2, x_{|R|})) \mid R^{\mathcal{S}_2}(x_1, \dots, x_{|R|})\}$.

A similar set of operators has been introduced by Barthelman [Bar97].

Let us emphasize that this set of operators provides a strict and natural extension to relational structures of vertex replacement (VR) graph operators. Let us illustrate how to obtain VR systems with PQFD systems on graphs. The usual definition of VR operators works over coloured directed graphs: directed graphs labelled by a finite set E and extended with a mapping which associates to each vertex a color belonging to some given finite set C of colors. In our case, we can encode such a graph into a structure over the signature $\Sigma = C \cup E$ where symbols in C and E have respective arity 1 and 2 and encode respectively the fact that a vertex has a given color, and the presence of an edge between two nodes. We can now introduce the four VR operators, and their equivalent PQFD expression.

Single vertex constant of color c — simply written $\mathbf{c}$ — represents the graph with one vertex of colour c and no edge.

It can be expressed as $\text{pqfd}[\phi_c](\text{one})$ with $\phi_{c'} = \mathbf{f}$ for any $c' \neq c$, $\phi_c = \mathbf{t}$ and $\phi_a = \mathbf{f}$ for any $a \in E$.

Disjoint union — written $\oplus$ as for structures — performs the disjoint union of two coloured graphs.

It can naturally be encoded by the disjoint union of structures $\oplus$.

Renaming color c_1 into color c_2 of a coloured graph G — written $\text{ren}_{c_1, c_2}(G)$ — changes the color mapping in such a way that every vertex of color other than c_1 keeps its original color, and vertices of original color c_1 have new color c_2 .

Let us suppose for simplicity that $c_1 \neq c_2$. The renaming operator can be encoded by $\text{pqfd}[(\phi_R)_{R \in \Sigma}](G)$ with $\phi_{c_1} = \mathbf{f}$, $\phi_{c_2} = c_1(x_1) \vee c_2(x_1)$, $\phi_c = c(x_1)$ for $c \in C - \{c_1, c_2\}$, and $\phi_a = a(x_1, x_2)$ for $a \in E$.

Adding edges labelled by a between color c_1 and color c_2 to a graph G — written $\text{add}_{c_1, c_2, a}(G)$ — adds to the coloured graph G all possible edges labelled by a with as origin a vertex of color c_1 and as destination a vertex of color c_2 .

The edge-adding operator can be encoded by $\text{pqfd}[(\phi_R)_{R \in \Sigma}](G)$ with $\phi_a = a(x_1, x_2) \vee (c_1(x_1) \wedge c_2(x_2))$, $\phi_b = b(x_1, x_2)$ for $b \in E - \{a\}$ and $\phi_c = c(x_1)$ for any $c \in C$.

PQFD operators can be used in equational systems: One can equip structures with the partial order of inclusion defined by $\mathcal{S} \subseteq \mathcal{S}'$ iff $\mathcal{U}_{\mathcal{S}} \subseteq \mathcal{U}_{\mathcal{S}'}$ and $R^{\mathcal{S}} \subseteq R^{\mathcal{S}'}$

for any symbol R . This ordering is a complete partial order (cpo) admitting the only structure of empty universe $\perp$ as smallest element. The semantic of operators is continuous with respect to this cpo. It means that a (even infinite) system of equations using PQFD operators admits a unique smallest solution.

Example 2. Let us illustrate infinite VR systems of equations for producing the graph of Figure 1. Let us first introduce the intermediate coloured graphs X_n presented in Figure 2.

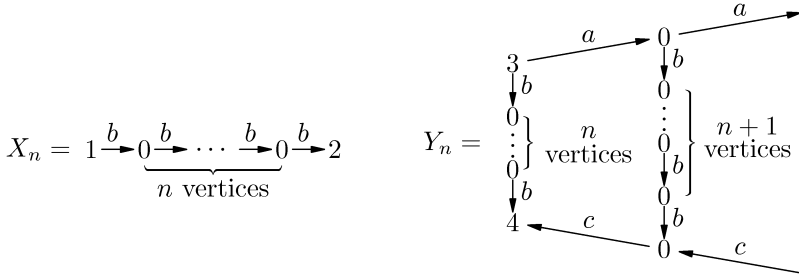


Fig. 2. The graphs X_n and Y_n .

The X_n graphs can be defined by the following recursive equations:

$$X_0 = \mathbf{add}_{1,2,b}(\mathbf{1} \oplus \mathbf{2}) \quad \text{and} \quad X_{n+1} = \mathbf{ren}_{3,2}(\mathbf{ren}_{2,0}(\mathbf{add}_{2,3,b}(X_n \oplus \mathbf{3}))) \quad (1)$$

We can now define the Y_n coloured graphs (notice Y_0 is isomorphic to the graph of Figure 1). They satisfy the following equation:

$$Y_n = \mathbf{ren}_{2,4}(\mathbf{ren}_{1,3}(\mathbf{ren}_{4,0}(\mathbf{ren}_{3,0}(\mathbf{add}_{1,3,a}(\mathbf{add}_{4,2,c}(X_n \oplus Y_{n+1})))))) \quad (2)$$

In fact the coloured graphs X_n and Y_n are the smallest possible graphs satisfying the equations (1) and (2): the step-ladder graph is the smallest solution of this equational system. Let us notice that, though infinite, this equational system can be represented by an infinite graph as depicted in Figure 3.

This process of encoding the equational system into a rooted graph is general. Formally, a rooted graph $\mathcal{E}$ is a PQFD-equational system if its edges:

- are labelled by $\{\oplus_1, \oplus_2\} \cup PQFD_0 \cup PQFD_1$ and
- for all element x of $\mathcal{U}_{\mathcal{S}}$,
 - if there is an edge labelled by **one** of target x then no edges originate from x .
 - else, either two edges originate from x , and are labelled by respectively $\oplus_1$ and $\oplus_2$,
or only one edge has origin x , and this edge is labelled by **one** or **pqfd** $[(\phi_R)]$ for some ϕ_R .

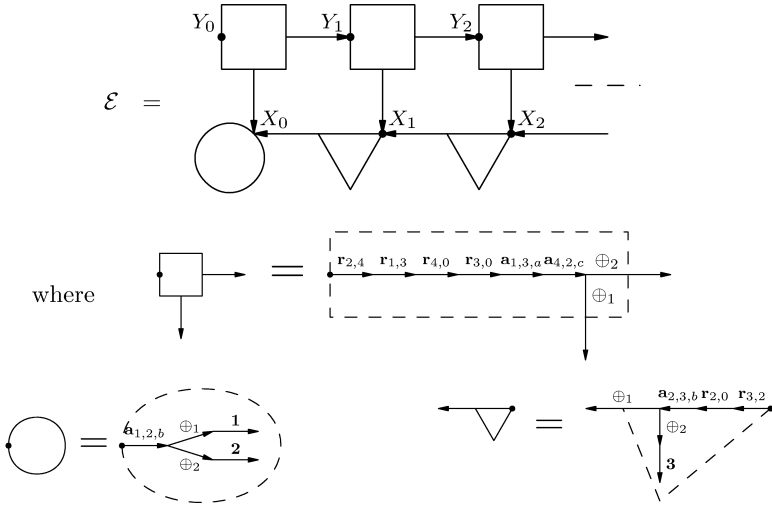


Fig. 3. An infinite VR equational system $\mathcal{E}$ describing the graph of Figure 1

The solution of such a system is defined as follows: let $\sigma^\mathcal{E}$ be the smallest function from vertices of $\mathcal{E}$ to structures satisfying:

- If $\text{one}^\mathcal{E}(x, y)$ then $\sigma^\mathcal{E}(x) = \underline{\text{one}}$,
- if $\text{pqfd}[(\phi_R)]^\mathcal{E}(x, y)$ then $\sigma^\mathcal{E}(x) = \text{pqfd}[(\phi_R)](\sigma^\mathcal{E}(y))$,
- and if $\oplus_1^\mathcal{E}(x, y)$ and $\oplus_2^\mathcal{E}(x, z)$ then $\sigma^\mathcal{E}(x) = \sigma^\mathcal{E}(y) \underline{\oplus} \sigma^\mathcal{E}(z)$.

then the semantic of the equational system $\mathcal{E}$, written $\llbracket \mathcal{E} \rrbracket$ is the graph $\sigma^\mathcal{E}(\text{root})$.

We will also make use of another operator: for $p \in \Sigma_1$ a unary symbol, the operator **fuse** _{p} applied to a structure $\mathcal{S}$ keeps the structure unchanged but collapses all the elements x satisfying $p^\mathcal{S}(x)$ into a single one. Formally, we define the equivalence relation $\equiv_p$ over $\mathcal{U}_\mathcal{S}$ by $x \equiv_p y$ iff $x = y$ or $p^\mathcal{S}(x)$ and $p^\mathcal{S}(y)$. The classes of equivalence for $\equiv_p$ of an element x is written $[x]_p$. The semantic of **fuse** _{p} is then defined by $\underline{\text{fuse}}_p(\mathcal{S}) = \mathcal{S}'$ with $\mathcal{U}_{\mathcal{S}'} = \{[x]_p \mid x \in \mathcal{U}_\mathcal{S}\}$ and for any n -ary symbol R , $R^{\mathcal{S}'} = \{([v_1]_p, \dots, [v_n]_p) \mid R^\mathcal{S}(v_1, \dots, v_n)\}$. The set of operators $PQFD$ increased with **fuse** operators is written $PQFD + F$.

In fact, the cpo used has to be slightly changed for the **fuse** operators to be continuous. Furthermore, the **fuse** operators make it necessary to put some extra restrictions to the system: a $PQFD + F$ equational system is said normalised if there is no predicate $R(y_1, \dots, y_{|R|})$ such that $y_i = y_j$ for $i \neq j$ in any formula appearing in a **pqfd** operator.

4 The Transformational Approach

Successively applying a finite number of transformations to a relational structure is a second technique for obtaining new relational structures. In this work, we are basically using two such transformations: MS-definable transduction and unfolding.

MS-definable transduction has already been presented. We define here a version of unfolding suitable for deterministic rooted graphs only. Given a deterministic rooted graph G labelled by E with a vertices set V , ρ_G is the function from E^* to V such that $\rho_G(u) = x$ with $root \xRightarrow{u} x$ (since the graph is deterministic, there is at most one such x). The unfolding of G is the deterministic rooted graph $Unf(G)$ with a set of vertices $V' = \{u \mid \exists x \in V, root \xRightarrow{u} x\}$ and such that for all edge symbol a , $a^{Unf(G)}(u, v)$ iff $a^G(\rho_G(u), \rho_G(v))$. The function ρ_G is a morphism of graph and is called the reduction (following the terminology of bisimulation).

We are interested here in transforming a deterministic graph by applying successively an unfolding and a MS-definable transduction.

Example 3. Let G be the graph presented in Figure 4.a with its *root* marked by an unlabelled edge and let $\mathcal{I}$ be the MS interpretation $(\delta, \{\Phi_a, \Phi_b, \Phi_c\})$ with $\delta(x) = \mathbf{true}$, $\Phi_a(x_1, x_2) = a(x_1, x_2)$, $\Phi_b(x_1, x_2) = b(x_1, x_2)$ and $\Phi_c(x_1, x_2) = (\exists x'_1. \exists x'_2. x'_1 \xRightarrow{b^*} x_1 \wedge x'_2 \xRightarrow{b^*} x_2 \wedge a(x'_2, x'_1)) \wedge \neg(\exists z. a(x_1, z) \vee a(x_2, z))$ where $x \xRightarrow{b^*} y$ is a MS formula stating that there is a path between x and y using only edges labelled by b . $\mathcal{I}(Unf(G))$ is the step-ladder of Figure 1 (Figure 4.b presents the unfolding of G).

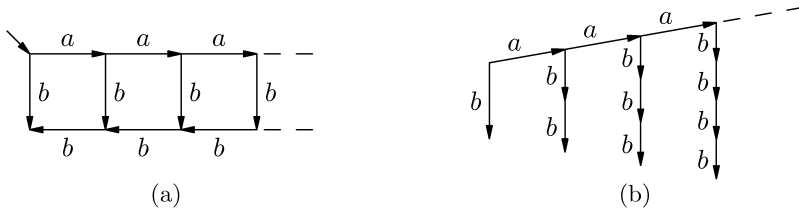


Fig. 4. The graph G (a) and its unfolding (b).

Those two transformations are sufficient for expressing $PQFD + F$ equational systems:

Lemma 1. *Given a normalised $PQFD + F$ equational system $\mathcal{E}$, there exists an MS interpretation $\mathcal{I}$ such that $\mathcal{I}(Unf(\mathcal{E}))$ is isomorphic to $\llbracket \mathcal{E} \rrbracket$.*

Proof (sketch). The first remark used in the proof is that unfolding preserves the solution of equational systems: $\llbracket \mathcal{E} \rrbracket = \llbracket Unf(\mathcal{E}) \rrbracket$.

For simplicity, let us suppose first that no **fuse** operators are used. Under this hypothesis, each element of $\mathcal{U}_{[\mathcal{E}]}$ can be uniquely identified with the **one**

operator appearing in $Unf(\mathcal{E})$ which has introduced it (if this operator is removed from the tree, then the element disappears from the structure). Let us call ρ this injective mapping from $\mathcal{U}_{[\mathcal{E}]}$ to $V_{Unf(\mathcal{E})}$. Then, there exists formulas Φ_R for all symbol R of arity n in the signature such that $Unf(\mathcal{E}) \models \Phi_R(\rho(x_1), \dots, \rho(x_n))$ iff $R^{[\mathcal{E}]}(x_1, \dots, x_n)$ holds. Let $\delta(x)$ be $(\exists y, \mathbf{one}(x, y))$, then the interpretation $\mathcal{I} = (\delta, (\Phi_R))$ is such that $\mathcal{I}(Unf(\mathcal{E}))$ is isomorphic to $[\mathcal{E}]$ (and ρ is the isomorphism).

If **fuse** operators are used, a similar ρ mapping can be provided: the difference is that it maps elements of $\mathcal{U}_{[\mathcal{E}]}$ to either **one** operators or **fuse** operators. The intention is that an element of $\mathcal{U}_{[\mathcal{E}]}$ is uniquely represented by a **one** operator iff no **fuse** operator ‘touched’ it in the equational system, in the other case, the element is uniquely represented by the closest to the root **fuse** operator in which it was involved. Apart from this distinction, the same technique is applied for providing the interpretation $\mathcal{I}$.

5 Prefix-Recognisable Structures

In this section, we focus on the internal representation of structures. Prefix-recognisable graphs have been introduced by Caucal [Cau96]. A possible description of these graphs is by systems of word rewriting. Blumensath [Blu01] extended this definition to relations of arbitrary arity. Those structures, when restricted to binary relations coincide with prefix-recognisable graphs. We give here a similar (and equivalent) definition of prefix-recognisable structures.

For simplicity, we fix a common *infinite* alphabet A . Let R, R' be two relations over A^* of respective arities k and l , we designate by $R \times R'$ the $(k+l)$ -ary relation defined by $(R \times R')(u_1, \dots, u_k, v_1, \dots, v_l)$ if and only if $R(u_1, \dots, u_k)$ and $R'(v_1, \dots, v_l)$. Let R be a k -ary relation over A^* and U a language of A^* , we designate by $U \cdot R$ the k -ary relation defined by $(U \cdot R)(uv_1, \dots, uv_k)$ iff $u \in U$ and $R(v_1, \dots, v_k)$. Let R be a k -ary relation and π a permutation of $[1, k]$, $R_\pi(x_1, \dots, x_k)$ iff $R(x_{\pi(1)}, \dots, x_{\pi(k)})$.

Definition 1. *The set of prefix-recognisable (PR) relations over A^* is the smallest set of relations satisfying:*

- for U a rational subset of A^* , the unary relation U is in PR ,
- if $R, R' \in PR$ then $R \times R' \in PR$,
- for $R, R' \in PR$ of same arity, $R \cup R' \in PR$,
- for $R \in PR$ and U a rational subset of A^* , $U \cdot R \in PR$,
- for $R \in PR$ and π a permutation of $[1, |R|]$, $R_\pi \in PR$.

Remark that the definition of each rational language only involves a finite number of letters in A . Thus each relation in PR refers to a *finite* number of letters.

A *PR-structure* is a relational structure of universe A^* with all interpretations in PR .

Prefix-recognisable graphs [Cau96] can be defined as graphs with edges defined by a finite union of relations of the form $U(V \times W)$ (with U, V and W rational languages) and vertices defined by a rational language L . This naturally

corresponds to the class of binary *PR*-systems restricted by a finite automaton. We extend this notion of restriction to infinite deterministic automaton.

In this article, we will use the term *automaton* to designate a rooted deterministic graph labelled by a finite subset of A . Moreover, we will assume that this graph comes with a set of vertices *Final*. As for finite automaton, we associate to every automaton $\mathcal{A}$ a language $\mathcal{L}_{\mathcal{A}} \subseteq A^*$ consisting of all words corresponding to the labelling of a path from *root* to an element in *Final*.

A *PR-system* $\mathcal{R}$ is a pair $(\mathcal{S}, \mathcal{A})$ where $\mathcal{S}$ is a *PR*-structure and $\mathcal{A}$ is an automaton. In the following, $\mathcal{R}$ will also designate the structure obtained by restricting $\mathcal{S}$ to $\mathcal{L}_{\mathcal{A}}$.

Example 4. Our example graph of Figure 1 can be described by a *PR*-system $\mathcal{R} = (\mathcal{S}, \mathcal{A})$. The *PR*-structure $\mathcal{S}$ has three non-empty binary relations a, b and c such that $a^{\mathcal{S}} = x^*y^* \cdot (\{\varepsilon\} \times (y + z))$, $b^{\mathcal{S}} = x^* \cdot (\{\varepsilon\} \times x)$ and $c^{\mathcal{S}} = x^* \cdot (xy^*z \times y^*z)$. The automaton $\mathcal{A}$ is presented in Figure 5.a. Its root is pointed by an unlabelled edge and all its states are in *Final*. The language recognised by $\mathcal{A}$ is the set of prefixes of $\{x^n y^n z \mid n \geq 0\}$. The graph obtained by restricting $\mathcal{S}$ to the language recognised by $\mathcal{A}$ (Figure 5.b) is isomorphic to the step-ladder (Figure 1).

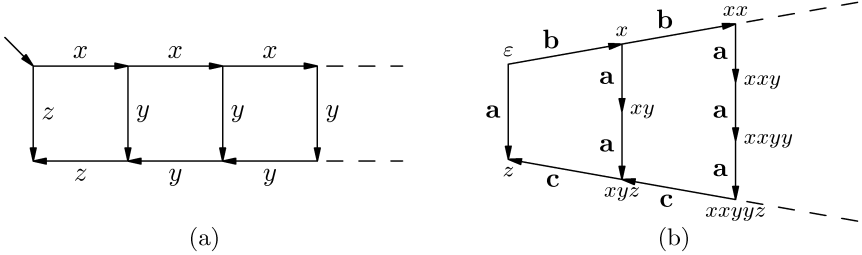


Fig. 5. The automaton $\mathcal{A}$ (a) and the *PR*-system $\mathcal{R}$ (b).

Lemma 2. *For any MS-definable transduction $\mathcal{T}$ and any deterministic graph G , there exists a *PR*-system $\mathcal{R} = (\mathcal{S}, \mathcal{A})$ such that $\mathcal{T}(\text{Unf}(G))$ is isomorphic to $\mathcal{R}$ and $\mathcal{A}$ is obtained from G by an MS-definable transduction.*

Proof (sketch). Let us consider here the simpler case where $\mathcal{T}$ is a *non-erasing* MS interpretation $(\text{true}, (\Phi_R)_{R \in \Sigma})$. For T a tree and x one of its nodes, $T_{/x}$ denotes the subtree of T rooted at x .

For every formula Φ_R of arity n , there exists an associated parity automaton $\mathcal{A}_R$. This automaton works on deterministic trees with n distinguished vertices $m_1, \dots, m_n$ called *marks*. The autaton accepts a tree T iff $T \models \Phi_R(m_1, \dots, m_n)$. We can always suppose that the states Q of $\mathcal{A}_R$ are enriched with informations about the expected marks: there is a mapping ϕ from Q to $2^{[1, n]}$ such that if a node x of T is assigned a state q in a successful run of $\mathcal{A}_R$ then the marks appearing in $T_{/x}$ are exactly the one of indices in $\phi(q)$.

We want to attach to every node of $\text{Unf}(G)$ the set of transitions of $\mathcal{A}_R$ starting a successful run on $\text{Unf}(G)_{/x}$. Let $\mathcal{M}_R$ be this application. By definition

of the runs of the automaton, the same transitions lead to the same winning runs for any two bisimilar starting nodes (x is bisimilar to y iff $Unf(G)_{/x} \approx Unf(G)_{/y}$). It follows that there is a mapping $\mathcal{B}_R$ attaching transitions to the vertices of graphs, such that $\mathcal{M}_R(Unf(G)) = Unf(\mathcal{B}_R(G))$. Furthermore, we show that this application $\mathcal{B}_R$ is an MS-interpretation (see also [Wal96] for a similar construction).

Finally, we define a n -ary PR -relation R which simulates the run of the parity automaton when $\phi(q) \neq \emptyset$, and prunes the run according to the information provided by $\mathcal{M}_R$ whenever $\phi(q) = \emptyset$.

Lemma 3. *For any PR -system $\mathcal{R} = (\mathcal{S}, \mathcal{A})$, there exists a PQFD-system $\mathcal{E}$ such that $\mathcal{E}$ is obtained by an MS definable transduction from $\mathcal{A}$ and $\mathcal{R}$ is isomorphic to $\llbracket \mathcal{E} \rrbracket$.*

Proof (sketch). The proof is syntactical. Let $(P_R)_{R \in \Sigma}$ be PR -relations and let $A_1, \dots, A_k$ be the finite automata accepting the rational languages involved in the PR -expressions describing the P 's relations and let $\mathcal{A}$ be the automaton restricting the PR -system.

We produce a new equational system working over signature Σ enriched by a new symbol for each state of an automaton A_i . The arity of the symbol is the arity of the relation in which $\mathcal{L}_{A_i}$ is used. The equational system is obtained from $\mathcal{A}$ by replacing each edge labelled by a with a **pqfd** operator which simulates simultaneously all a transitions of the A_i 's. Disjoint union operators are used to follow the branching structure of $\mathcal{A}$. **one** operators are used for each *Final* state of $\mathcal{A}$.

6 Conclusion

By combining Lemmas 1,2 and 3, we obtain the following theorem:

Theorem 1. *Let $\mathcal{F}$ be a family of deterministic graphs closed by MS-definable transductions, the following classes of structures are isomorphic:*

- the solutions of systems of equations over the PQFD operators represented by a graph in $\mathcal{F}$,
- the solutions of normalised systems of equations over the PQFD + F operators represented by a graph in $\mathcal{F}$,
- the structures obtained by applying an MS-definable transduction to the unfolding of a deterministic graph in $\mathcal{F}$,
- the prefix-recognisable structures restricted to the language accepted by a deterministic automaton in $\mathcal{F}$.

Let us notice that, according to the third representation, if $\mathcal{F}$ has a decidable MS theory, then it is also the case of the resulting structures.

Removing the normalisation of $PQFD + F$ equational systems is an open question.

Acknowledgements. We would like to thank Didier Caucal for his advices.

References

- [Bar97] K. Barthelmann. On equational simple graphs. Technical Report 9, Universität Mainz, Institut für Informatik, 1997.
- [Blu99] A. Blumensath. Automatic structures. Diploma thesis, RWTH-Aachen, 1999.
- [Blu01] A. Blumensath. Prefix-recognisable graphs and monadic second-order logic. Technical Report AIB-06-2001, RWTH Aachen, May 2001.
- [Cau92] D. Caucal. On the regular structure of prefix rewriting. *TCS*, 106:61–86, 1992.
- [Cau96] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *ICALP'96*, volume 1099 of *LNCS*, pages 194–205, 1996.
- [CM02] B. Courcelle and J.A Makowsky. Fusion in relational structures and the verification of mso logic. In *MSCS*, volume 12, pages 203–235, 2002.
- [Col02] T. Colcombet. On families of graphs having a decidable first order theory with reachability. In *ICALP'02*, 2002.
- [Cou89] B. Courcelle. The monadic second-order logic of graphs ii: infinite graphs of bounded tree width. *Math. Systems Theory*, 21:187–221, 1989.
- [Cou90] B. Courcelle. *Handbook of Theoretical Computer Science*, chapter Graph rewriting: an algebraic and logic approach. Elsevier, 1990.
- [Cou94] B. Courcelle. Monadic-second order definable graph transductions : a survey. *TCS*, vol. 126:pp. 53–75, 1994.
- [CW98] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. In *Annals of Pure and Applied Logic*, 1998.
- [DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 242–248, 1990.
- [L d02] C. L ding. Ground tree rewriting graphs of bounded tree width. In *Stacs 02*. *LNCS*, 2002.
- [Mor00] C. Morvan. On rational graphs. In J. Tiuryn, editor, *FOSSACS'00*, volume 1784 of *LNCS*, pages 252–266, 2000.
- [MS85] D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.
- [S n92] G. S nizergues. Definability in weak monadic second-order logic of some infinite graphs. In *Dagstuhl seminar on Automata theory: Infinite computations, Warden, Germany*, volume 28, page 16, 1992.
- [Tho97] W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, 3:389–455, 1997.
- [Urv02] T. Urvoy. On abstract families of graphs. In *DLT 02*, 2002.
- [Wal96] I. Walukiewicz. Monadic second order logic on tree-like structures. In *STACS'96*, volume 1046 of *LNCS*, pages 401–414, 1996.

Adaptive Raising Strategies Optimizing Relative Efficiency

Arnold Schönhage

Institut für Informatik III, Universität Bonn
Römerstrasse 164, D 53117 Bonn, Germany
schoe@cs.uni-bonn.de

Abstract. Adaptive raising by successive *trials* $t_0 < t_1 < \dots$ until some unknown goal $g > 1$ has been *found* by $t_n \geq g$, causing total cost $T(g) = t_0 + \dots + t_n$, is studied for optimizing $T(g)/g$. For corresponding games, where player G setting g and ‘finder’ F choosing $t_0, t_1, \dots$ are playing mixed strategies, we prove a “Law of optimal adapting factor e ”. Section 2 is more general about adaptive raising on several tracks, in Sect. 3 we add proofs for the optimal *competitive factors* under corresponding worst case analysis.— Methods and results are similar to those about searching for a point on a line or on many rays, see [1,3,4,5,6].

1 Adaptive Raising for Goals of Unknown Size

The subject of this study is a scheduling problem of a very basic nature, specified in the following way. Assume we have to settle some task, to reach some goal of unknown size, described by a real number $g > 1$ (e. g. measuring computing time, or any sort of cost), and assume that the only moves at our disposal are successive *trials* of increasing (else freely choosable) costs $t_0 < t_1 < \dots$ until (by some a posteriori testing after each trial) $t_n \geq g$ has been found, to be taken as indication of successful completion of that task, with total cost $T(g) = t_0 + t_1 + \dots + t_n$. Then the question is how to choose these t_j in order to achieve good *relative* efficiency, i. e. to keep the quotient $T(g)/g$ as small as possible, to achieve the smallest possible *competitive factor*.

When we restrict this question to strategies in geometric progression, like $t_j = a \cdot z^j$ with a fixed factor $z > 1$ and initial $t_0 = a$ in $1 < a \leq z$, the corresponding *worst case* analysis becomes quite simple: the first successful trial settling goals $g = a \cdot z^k + \varepsilon$ with small $\varepsilon > 0$ is $t_{k+1} = a \cdot z^{k+1}$, so such g ’s enforce total cost $T(g) = a \cdot (z^{k+2} - 1)/(z - 1)$. With $\varepsilon \rightarrow 0$, $k \rightarrow \infty$, the resulting quotients $T(g)/g$ are increasing to their limit value $z^2/(z - 1)$, which becomes minimal for $z = 2$. The simple doubling strategy $t_j = 2^{j+1}$, for example, guarantees $T(g) < 4 \cdot g - 2$. — Remarkably, this factor 4 remains the best worst case bound of such kind for any other general strategy as well.

Theorem 1.1. *For any unbounded real sequence $1 < t_0 < t_1 < t_2 < \dots$ and $g > 1$, use the minimum n with $t_n \geq g$ to define $T(g) = t_0 + \dots + t_n$. Then $\rho = \limsup_{g \rightarrow \infty} T(g)/g$ satisfies $\rho \geq 4$.*

We shall prove this (and a more general result, see Theorem 2.1) in Sect. 3 below. The main ideas for such proofs can already be found in [4] and in Sect. 2 of [1] on the similar problem of search for a point on a line and its generalization to any finite number of rays. Besides the translation to the problem of adaptive raising, our proof will also present slightly modified versions of those methods.

From a practical point of view, however, the worst case behavior will often not be of major concern. Usually we will rather be interested in strategies with good (or optimal) performance *on the average*. One way to discuss problems of that type would be to analyze such optimizations for various common or plausible probability distributions of goals, regarding g as a random variable. But how to proceed, if even the distribution of g is unknown, or arbitrary?

Here we prefer to study such issues in a *game theoretic* framework with two players F, G . For any constant $c > 1$, we define the *game of adaptive raising* $\Gamma(c)$ by the following rules. First the “goal setter” G plays some $g > 1$ and has to deposit a payment of $c \cdot g$ (with an umpire or trustee), later handed out to F . Next the “finder” F plays a sequence of t_j (as in Theorem 1.1) until $t_n \geq g$, and has then to pay $T(g) = t_0 + \dots + t_n$ to G .

In this setting optimization on the average is captured by the notion of *mixed* strategies, and so the problem of optimal relative efficiency boils down to this basic question: *For which values of c are the games $\Gamma(c)$ advantageous for F ?* — Beyond that we are, of course, also interested to know corresponding finder strategies explicitly, e.g. in the sense of clever *randomized* algorithms for F . Perhaps we may even restrict the finder’s strategies by imposing some condition of constructiveness, whereas the devilish adversary G may clearly invoke mixed strategies of any sort, relying on pure mathematical existence!

In any case, there is a very simple answer to that basic question, given in our next theorem. In view of its fundamental nature, it deserves an extra naming; we propose to call it the “Law of optimal adapting factor e ”.

Theorem 1.2. (i) *Game $\Gamma(c)$ is advantageous for F iff $c \geq e = 2.71828\dots$*
 (ii) *For any $g > 1$, finder F achieves an expected value $E(T(g)) = e \cdot g - 1$ by playing the mixed strategy choosing $t_0 = x$ at random in the interval $(1, e]$ with logarithmic distribution dx/x and then proceeding in geometric progression $t_{j+1} = e \cdot t_j$ for all j .*

Let us first prove (ii), covering the *if*-part of (i). It will be instructive to discuss such mixed strategies of geometric progressions $t_{j+1} = z \cdot t_j$ for arbitrary $z > 1$, then choosing $t_0 = x$ with distribution $(1/\ln z) \cdot dx/x$ on $(1, z]$. So let $g = b \cdot z^k$ with unique $b \in (1, z]$. For x in the lower interval $t_0 = x < b$, the final trial is $t_n = x \cdot z^{k+1}$ with $n = k + 1$, which (depending on the choice of x) causes costs $T(g|x) = x \cdot (z^{k+2} - 1)/(z - 1)$, similarly $T(g|x) = x \cdot (z^{k+1} - 1)/(z - 1)$ in the upper interval $b \leq t_0 = x \leq z$. Hence the expected costs are

$$E(T(g)) = \frac{1}{\ln z} \int_1^z T(g|x) dx/x = \frac{b \cdot z^{k+1} - 1}{\ln z} = \frac{z}{\ln z} \cdot g - \frac{1}{\ln z}, \quad (1)$$

which yields (ii), minimizing that factor $z/\ln z$ by choosing $z = e$.

The *only-if*-part of (i), to show that F has no strategy to win for any value $c < e$, is relying on the same idea: we exhibit a suitable mixed strategy for the adversary G enforcing sufficiently large expected costs with any strategy of F . Asymptotically, admitting arbitrarily large values of g , such a good mixed strategy for G is furnished by playing $g = y$ with probability dy/y^2 , but we cannot argue with this distribution directly, since that density $1/y^2$ would lead to infinite expectation $E(g)$, and due to $T(g) \geq g$ also to $E(T(g)) = \infty$.

To circumvent this difficulty, we approximate that infinite case considering mixed strategies for G with the same distribution dy/y^2 now restricted to some bounded interval $(1, B]$, and playing the top value $g = B$ with the remaining probability $\int_B^\infty dy/y^2 = 1/B$. As we are going to prove *lower* bounds for the expected costs $E(T(g))$ of the finder F , we may additionally favor F by assuming that F *knows* this interval bound B . Then it will suffice to consider *finite* sequences $1 < t_0 < t_1 < \dots < t_l = B$ as strategies of F , with the quotients $q_0 = t_0$, $q_1 = t_1/t_0$, $\dots$, $q_l = t_l/t_{l-1}$ satisfying the side condition

$$q_0 \cdot q_1 \cdots q_l = B. \quad (2)$$

With respect to that distribution dy/y^2 on $(1, B]$ plus the atom $1/B$ in its endpoint, the expected deposit of G is $E(c \cdot g) = c \cdot \ln B + c$, and the expected costs for such a strategy of F are readily calculated as

$$E(T(g)) = t_0 + t_1 \cdot 1/t_0 + \dots + B \cdot 1/t_{l-1} = q_0 + \dots + q_l.$$

Since the arithmetic mean of the q 's, $E(T(g))/(l+1)$, is never less than their geometric mean known from (2), we thus obtain the lower bound

$$E(T(g)) \geq (l+1)B^{1/(l+1)} \geq e \cdot \ln B$$

for any strategy of F , where setting $z = B^{1/(l+1)}$ with $l+1 = \ln B / \ln z$ again amounts to that crucial quotient $z / \ln z \geq e$. — Altogether this shows for any value $c < e$, that F will suffer an expected loss of at least $(e - c) \cdot \ln B - c > 0$ whenever G is playing such a mixed strategy with a sufficiently large B . $\square$

In addition we point out that these geometric search strategies with factor $z = e$ optimizing the relative efficiency in the game theoretic sense are still fairly efficient also with respect to their worst case behavior, as the resulting worst case ratio $e^2/(e-1) < 4.3003$ is not much greater than the optimal factor 4. Conversely, (1) shows that playing a mixed doubling strategy with $z = 2$, optimizing the worst case behavior, will at the same time achieve an *expected* competitive factor $2/\ln 2 < 2.8854$ less than $1.062 \cdot e$.

In [2], a similar game theoretic approach has been worked out for the “linear search problem” (see [3]), also using such a distribution $\gamma \cdot dy/y^2$ on large finite intervals. The same issue has then found renewed interest as so-called “cow-path problem” (more generally also on $m > 2$ lanes), see [6] and [5], although these papers are using mixed adversary strategies with a different family of distributions, apparently because they are optimizing the *expected competitive ratio* (cf. Lemma 4.2 of [6]) rather than the ratio of expectations.

2 Adaptive Raising on Several Tracks

Now we consider the more general problem of adaptive raising with some goal hidden in any of m “tracks” $R_0, \dots, R_{m-1}$, modeled as m copies of $[1, \infty) \subset \mathbb{R}$. So the goal is now some $g > 1$ plus some *selection index* $s \in \{0, \dots, m-1\}$. Then the pure strategies for finder F (“ F -strategies”, for short) are infinite sequences of pairs (s_j, t_j) of real numbers $t_j > 1$ plus selection indices $s_j < m$ such that the m classes $J_\mu = \{j \in \mathbb{N} : s_j = \mu\}$ are infinite and the subsequences $(t_j : j \in J_\mu)$ are strictly increasing and unbounded for all $\mu < m$. With such a strategy, F is said to have found (g, s) after having arrived at the minimal *stopping index* n with $t_n \geq g$ and $s_n = s$, and with total cost $T(g, s) = t_0 + t_1 + \dots + t_n$. — How should finder F choose these sequences $st = ((s_j, t_j) : j \in \mathbb{N})$ to achieve good relative efficiency, i. e. to keep $T(g, s)/g$ as small as possible?

2.1 Worst Case Analysis for m Tracks

Let us first have a brief look at the worst case scenario, where F chooses such a pure strategy, here assumed to be known to adversary G who then can pick some (g, s) to make $T(g, s)/g$ especially large. Again we begin with the case of geometric strategies $t_j = z^{j+1}$ with a fixed factor $z > 1$, cyclically alternating the tracks by choosing $s_j = (j \bmod m)$. Then the disadvantageous goals are $g = z^{k+1} + \varepsilon$ with $s \equiv k \pmod{m}$ and small $\varepsilon > 0$, leading to stopping index $n = k + m$ and thus to total cost $T(g, s) = (z^{k+m+2} - z)/(z - 1)$. With $\varepsilon \rightarrow 0$, $k \rightarrow \infty$, the resulting quotients $T(g, s)/g$ are increasing to their limit $z^{m+1}/(z - 1)$. This becomes minimal for $z = 1 + 1/m$, with minimum value

$$\beta_m = (m + 1)((m + 1)/m)^m < (m + \tfrac{1}{2}) \cdot e < \beta_m + 1/8m. \quad (3)$$

So that cyclic strategy with $t_j = (1 + 1/m)^{j+1}$ guarantees the upper bound $T(g, s) < \beta_m \cdot g - (m + 1)$, analogous to the doubling strategy for $m = 1$.

Closely related constants $2\beta_m + 1$ have been found in [4] (see also [1]) as optimal worst case competitive ratios for the linear search problem generalized to $m + 1$ rays. As we shall prove in Sect. 3, analogously (worst case) optimality of these β_m does hold for our problem of adaptive raising on m tracks.

Theorem 2.1. *For any F -strategy $((s_j, t_j) : j \in \mathbb{N})$ for adaptive raising on m tracks and $g > 1$, $s < m$, use the minimum n with $t_n \geq g$ and $s_n = s$ to define $T(g, s) = t_0 + \dots + t_n$. Then $\rho = \limsup_{g \rightarrow \infty} T(g, s)/g$ satisfies $\rho \geq \beta_m$.*

2.2 The Game Theoretic Version

Let us now turn to the issue of mixed strategies in corresponding games $\Gamma_m(c)$ under the more natural condition that first player G chooses some secret goal (g, s) and deposits $c \cdot g$ for F , and only then finder F plays some F -strategy $((s_j, t_j) : j \in \mathbb{N})$ until $t_n \geq g$ with $s_n = s$, and finally pays $T(g, s) = t_0 + \dots + t_n$ to G . As for the simple case $m = 1$ of Sect. 1, again the crucial question is: *For which values of c are the games $\Gamma_m(c)$ advantageous for F ?*

We are able to extend our method of proof for Theorem 1.2 such that we shall obtain a complete answer to this question, quite analogous to Theorem 1.2, although things are more complicated by additional technicalities. For the sake of motivation, we therefore first describe and analyze certain mixed F -strategies that will lead to particular functions $f_m(z)$ analogous to the factor $f_1(z) = z/\ln z$ of Sect. 1. Then we shall state the main theorem and continue in 2.3 with the lower bound proof by analyzing corresponding mixed strategies for player G , while all technical details concerning the minimum points z_m of the f_m and the optimal ratios $\alpha_m = f_m(z_m)$ are postponed to Subsection 2.4, including an analysis of the asymptotical behavior of the z_m and of the α_m for $m \rightarrow \infty$.

Definition 2.1. For any $z > 1$, $S(m, z)$ shall denote the mixture of F -strategies with initial index $s_0 = i$ at random, any $i < m$ with probability $1/m$, and $t_0 = x \in (1, z]$ at random with distribution $1/\ln z \cdot dx/x$, then F proceeds geometrically by $t_{j+1} = z \cdot t_j$ with cyclic track selection $s_j = (j + i \bmod m)$.

In order to compute the expectation $E(T(g, s))$ with respect to $S(m, z)$ for any given goal (g, s) , we write $g = b \cdot z^k$ with unique $b \in (1, z]$ and $k \in \mathbb{N}$. Since $S(m, z)$ is symmetric with respect to the probabilities for $s_0 = i$, we may without restriction assume that $s \equiv k \pmod{m}$. Then the stopping index n with $t_n \geq g$ and $k \equiv s = s_n \equiv n + i \pmod{m}$ is easily determined as

$$\begin{aligned} n(i, x) &= k + m - i \quad \text{for } i > 0, & n(0, x) &= k + m \quad \text{for } x < b, \\ & & n(0, x) &= k \quad \text{for } x \geq b. \end{aligned}$$

So we obtain $T(g, s)$ with analogous dependencies on i, x , namely

$$\begin{aligned} T(g, s|i, x) &= x(z^{k+m-i+1} - 1)/(z - 1) \quad \text{for } 0 < i < m, \\ T(g, s|0, x) &= x(z^{k+m+1} - 1)/(z - 1) \quad \text{for } x < b, \\ T(g, s|0, x) &= x(z^{k+1} - 1)/(z - 1) \quad \text{for } x \geq b. \end{aligned}$$

Based on these expressions, an elementary calculation yields

$$\begin{aligned} E(T(g, s)) &= 1/m \sum_{i=0}^{m-1} \frac{1}{\ln z} \int_1^z T(g, s|i, x) dx/x \\ &= \frac{b \cdot (z^{k+m} + \dots + z^{k+1})}{m \cdot \ln z} - \frac{1}{\ln z} = f_m(z) \cdot g - \frac{1}{\ln z}, \\ \text{with } f_m(z) &= 1/m \cdot (z^m + \dots + z)/\ln z = \frac{z^{m+1} - z}{m(z - 1) \cdot \ln z} \quad (z > 1). \end{aligned} \quad (4)$$

In 2.4 we shall prove that these functions have unique minimum points z_m , with $w_m = z_m^m$ converging to the unique solution ω of the equation $\ln w = 2 - 2/w$ in $w > 1$, with $\omega = 4.92155\dots$, and shall analyze the growth of the minima $\alpha_m = f_m(z_m)$ for $m \rightarrow \infty$. Prepared in this way, we now state our main result.

Theorem 2.2. (i) Game $\Gamma_m(c)$ is advantageous for F iff $c \geq \alpha_m = f_m(z_m)$, where $z_m > 1$ is the unique minimum point of the function f_m in (4).

(ii) For any goal (g, s) , $g > 1$, $s < m$, finder F achieves the expected value $E(T(g, s)) = \alpha_m \cdot g - 1/\ln z_m$ by playing strategy $S(m, z_m)$ as defined above.

Part (ii) and the *if*-part of (i) are clear from the analysis of the strategies $S(m, z)$ that have led us to the f_m . Our lower bound proof for the *only-if*-part, that F cannot achieve any smaller competitive factor, follows in the next section.

Very similar functions $R(m, z) = 1 + f_m(z) \cdot 2/z$ occur in Theorem 3.1 of [6] as competitive factors for analogous probabilistic algorithms for the “cow-path” problem on m lanes ($m \geq 2$), where the minima $r_m^* = \min_{z>1} R(m, z)$ have turned out to be the *optimal* competitive factors for that problem. For $m = 2$, that optimality proof is in [6] (previously also in [2]); for general $m > 2$ we refer to Sect. 3.2 and to Appendix B of [5].

2.3 Good Mixed Strategies for Player G

In obvious generalization of our proof for Theorem 1.2 we let player G select any of the tracks R_s ($s < m$) with probability $1/m$ and then choose $g = y > 1$ with distribution dy/y^2 , but again restricting $g = y$ to some large finite interval $(1, B]$ with that distribution for $y < B$, or playing $g = B$ with probability $1/B$. Then the expectation of G ’s deposit is easily determined as $E(c \cdot g) = c \cdot \ln B + c$.

In the sequel we consider any F -strategy st and derive a lower bound for the expectation of the corresponding random variable $T(g, s)$. Again we may assume that F knows this B , hence it will suffice to discuss *finite* F -strategies $st = ((s_j, t_j) : 0 \leq j < l)$ of any length $l \geq m$ with $s_j < m$, $t_j > 1$, and nonempty index classes $J_\mu = \{j < l : s_j = \mu\}$ for all $\mu < m$, where the subtuples $(t_j : j \in J_\mu)$ are strictly increasing and ending with the maximal value $t_j = B$. Moreover it is convenient to extend such strategies at their lower end by a “preamble” of m dummy pairs $(i - m, t_{i-m})$ with $t_{i-m} = 1$ for $0 \leq i < m$, and to extend the index classes accordingly, setting $J'_i = \{i - m\} \cup J_i$. This admits to define the *preceding selections* for any $j \geq 0$ as

$$p(i, j) = \max\{k \in J'_i : k < j\}. \quad (5)$$

Then the expectation of $T(g, s) = t_0 + t_1 + \dots + t_n$ with stopping index n as random variable under that mixed strategy of G is easily calculated from the probabilities $\text{prob}(n \geq j)$, namely

$$E(T(g, s)) = \sum_{j=0}^{l-1} t_j \cdot \text{prob}(n \geq j) = 1/m \sum_{j=0}^{l-1} t_j \cdot \sum'_{i < m} 1/t_{p(i, j)}, \quad (6)$$

where the prime at the last summation sign shall indicate to omit all terms $1/B$ resulting from $p(i, j) = \max J_i$. Our subsequent analysis requires to generalize formula (6) and such finite strategies to preamble pairs $(i - m, u_i)$ with any $u = (u_0, \dots, u_{m-1}) \in [1, B]^m$, then of course with $t_j > u_i$ for all $j \in J_i$. Let $FF(u, B)$ denote the set of such finite F -strategies st . Imitating (6) we define a *cost function* C and infima $V(u)$,

$$C(u, st) = \sum_{j=0}^{l-1} t_j \cdot \sum'_{i < m} 1/t_{p(i, j)} \quad \text{for } st \in FF(u, B) \quad (7)$$

$$V(u) = \inf\{C(u, st) : st \in FF(u, B)\}, \quad (8)$$

which will turn out to be *minima* attained for certain optimal strategies st . Finally a lower bound for $V(1, \dots, 1)$ will yield the desired lower bound for $E(T(g, s))$ in (6), but that requires first to show (in a sequence of several “steps”) that the optimal strategies for $u = (1, \dots, 1)$ must necessarily be *cyclic* most of the time, with $s_j = (j \bmod m)$ as long as $t_j < B$, after some initial permutation.

Step 1. Let us call $st \in FF(u, B)$ “close to optimal” iff $C(u, st) < V(u) + \frac{1}{2}$. Such st have *bounded length* $l < mB + 1$, since each term of the outer sum in (7) is > 1 , whence $l < C(u, st) < V(u) + \frac{1}{2}$, and the shortest $st_0 \in FF(u, B)$ with just one pair (i, B) for each i with $u_i < B$ shows $V(u) \leq C(u, st_0) \leq mB$.

Step 2. With $\delta = 1/2mB$, any $st \in FF(u, B)$ close to optimal with trials t_j , index classes J_i , and preceding selections as in (5) satisfies

$$j \in J_i \text{ and } (t_j < B \text{ or } p(i, j) \geq 0) \implies t_j - t_{p(i, j)} \geq \delta. \quad (9)$$

Proof. Let $j \in J_i$, $k = p(i, j)$, and assume $t_j < t_k + \delta$. With $t_j < B$, we could then alter st into a shorter strategy st' by omitting the pair (i, t_j) , which in (7) would save $t_j/t_k > 1$ at least, and increase less than $l < mB + 1$ other terms t_h/t_j to t_h/t_k by a factor $t_j/t_k < 1 + \delta$. This would imply

$$C(u, st') < (C(u, st) - 1)(1 + \delta) < V(u) - \frac{1}{2} + mB \cdot \delta < V(u),$$

contradicting (8). In the other case $t_j = B$ and $k \geq 0$, we could then replace the k -th pair of st by (i, B) and omit the j -th pair, thereby in (7) saving

$$t_k/t_{p(i, k)} + t_j/t_k - t_j/t_{p(i, k)} > 1 - \delta > \frac{1}{2}$$

at least, which is impossible for st being close to optimal. $\square$

The main conclusion from Steps 1, 2 is that we can restrict definition (8) to the *compact* subset of strategies $st \in FF(u, B)$ of length $l < mB + 1$ and satisfying (9), compact when considered as a finite collection of closed bounded subsets of some $\mathbb{R}^l$. Since $C(u, st)$ is continuous (and even analytic) in the t_j on each of these components, $V(u)$ is therefore a *minimum*.

Step 3. For any optimal strategy st with $V(u) = C(u, st)$ and for any of its intermediate pairs (i, t_j) with $u_i < t_j < B$, we must have $\partial C(u, st)/\partial t_j = 0$. Moreover, permuting the track selection, i.e. the index classes of an optimal strategy st , we find that $V(u_0, \dots, u_{m-1})$ is a *symmetric* function.

Step 4. For $i < m$, $1 \leq u_i < B$, $V(u_0, \dots, u_{m-1})$ is strictly decreasing in u_i .

Proof. Consider an st with $V(u) = C(u, st)$ and for $u_i < u'_i < \min(B, u_i + \delta)$ the strategy $st' \in FF(u', B)$ obtained from st by replacing its preamble pair $(i-m, u_i)$ with $(i-m, u'_i)$. Then (7) implies $C(u, st) - C(u', st') = \sigma \cdot (1/u_i - 1/u'_i)$ with a nonzero sum σ of certain t_j , hence $V(u') \leq C(u', st') < C(u, st) = V(u)$.

Step 5. For any t_0 with $u_0 < t_0 \leq B$, we have

$$V(u_0, u_1, \dots, u_{m-1}) \leq t_0 \cdot \sum'_{i < m} 1/u_i + V(t_0, u_1, \dots, u_{m-1}), \quad (10)$$

where equality in (10) implies $u_0 \leq u_i$ for all $i < m$.

Proof. That inequality (10) follows from the definitions (7) and (8), since an initial pair $(0, t_0)$ continued with an optimal strategy st' for $V(t_0, u_1, \dots, u_{m-1})$ yields a strategy $st \in FF(u, B)$. If such st happens to be optimal with equality in (10), the additional claim $u_0 \leq u_i$ is obvious for $u_i \geq t_0$, else consider $u_1 < t_0$, for example. Then the symmetry of V in u_0, u_1 combined with (10) yields also

$$V(u_0, u_1, u_2, \dots) \leq t_0 \cdot \sum'_{i < m} 1/u_i + V(u_0, t_0, u_2, \dots, u_{m-1}),$$

whence $V(u_1, t_0, u_2, \dots) = V(t_0, u_1, u_2, \dots) \leq V(u_0, t_0, u_2, \dots, u_{m-1})$, and because of Step 4 therefore $u_1 \geq u_0$.

Step 6. *If an optimal strategy $st \in FF(u, B)$ has the initial pairs $(0, t_0)$, $(1, t_1)$ with $t_0 < B$ and $t_1 < B$, then $t_0 < t_1$. By symmetry, the same will hold for other index pairs as well.*

Proof. Because of $t_0 < B$ and $s_0 = 0$, there exists $k = \min\{j \in J_0 : j > 0\}$, and by $t_1 < B$ also $h = \min\{j \in J_1 : j > 1\}$, with $t_h > t_1$. For this optimal st the terms in (7) depending on t_0 are

$$t_0(1/u_0 + w) + (t_1 + \dots + t_k)/t_0 \quad \text{with} \quad w = \sum'_{0 < i < m} 1/u_i,$$

where the partial derivative with respect to t_0 must be zero (cf. Step 3) which implies $t_0^2 = (t_1 + \dots + t_k)/(1/u_0 + w)$. In the same manner we obtain $t_1^2 = (t_2 + \dots + t_h)/(1/t_0 + w)$, and with this t_1 the t_1 -contribution to (7) becomes $H = (t_2 + \dots + t_h) \cdot 2/t_1$. If $h > k$, then $t_h > t_1$ and $t_0 > u_0$ imply $t_1^2 > t_0^2$, hence $t_1 > t_0$. In any case this argument excludes that $t_0 = t_1$, because otherwise we could exchange the roles of J_0 and J_1 to obtain $h > k$ with $t_1 > t_0$.

To falsify $t_0 > t_1$, we apply (10) (with equality) to st , and to the strategy st' with alternate initial pairs $(0, t_1)$, $(1, t_0)$, obtaining

$$\begin{aligned} V(u_0, u_1, u_2, \dots) &= t_0(1/u_0 + w) + t_1(1/t_0 + w) + V(t_0, t_1, u_2, \dots) \\ &\leq t_1(1/u_0 + w) + t_0(1/t_1 + w) + V(t_1, t_0, u_2, \dots). \end{aligned}$$

By symmetry of V therefore $t_0/u_0 + t_1/t_0 \leq t_1/u_0 + t_0/t_1$, and so dividing $(t_0 - t_1)/u_0 \leq (t_0^2 - t_1^2)/t_0 t_1$ by $t_0 - t_1 > 0$ yields $t_0 t_1 \leq (t_0 + t_1)u_0 < 2t_0 u_0$, $t_1 < 2u_0 \leq 2u_1$ by Step 5. This, however, would decrease the t_1 -contribution to (7) to $(t_2 + \dots + t_h)/u_1 < H$, which would replace that H if we omit the pair $(1, t_1)$ from strategy st , contradicting the optimality of st .

Step 7. *If an optimal strategy $st \in FF(u, B)$ is beginning with a pair (μ, B) , then $u_i \geq B/4m$ for all $i < m$.*

Proof. It suffices to show $u_0 \geq B/4m$ for $\mu = 0$, say; then Step 5 implies that lower bound for all i . In case of $B/u_0 > 4m$ we could replace that pair $(0, B)$ which contributes $B(1/u_0 + w)$ to (7) with two pairs $(0, t_0)$, $(0, B)$ contributing $D = t_0(1/u_0 + w) + B/t_0 + Bw$. Since $u_0 \leq u_i$ for all i implies $1/u_0 + w \leq m/u_0$, choice of $t_0 = B/2m$ would then lead to $D \leq B/2u_0 + 2m + Bw < B/u_0 + Bw$, contradicting the optimality of st . $\square$

Now we are prepared to derive the desired lower bound for $V(1, \dots, 1)$, for large $B \gg 4m$. According to Step 5, the selection indices $s_0, \dots, s_{m-1}$ of the

first m pairs of an optimal strategy for $V(1, \dots, 1)$ (all $u_i = 1$) must attain m different values, after suitable permutation of the tracks we can therefore assume $s_i = i$ for $i < m$. Then repeated use of Step 6 shows $t_0 < t_1 < \dots < t_{m-1}$, and combined with Step 5 continuation $t_{m-1} < t_m < t_{m+1} < \dots$ with cyclic selection $s_j = (j \bmod m)$ until we arrive at the first index k with $t_k = B$. Then Step 7 guarantees $t_j \geq B/4m$ for all $j \geq k - m$. For the quotients $q_j = t_j/t_{j-1}$ (with $q_j = 1$ for the preamble indices $j < 0$) this implies

$$q_{k-1-i} \cdot q_{k-2-i} \cdots q_{-i} \geq B/4m \quad \text{for } 0 \leq i \leq m-1. \quad (11)$$

In formula (7) for this strategy st and $u = (1, \dots, 1)$, the preceding selections $p(i, j)$ for any $j < k$ are just the indices $j-1, \dots, j-m$ in some cyclic permutation. Therefore we have

$$V(1, \dots, 1) > \sum_{j=0}^{k-1} \sum_{i=0}^{m-1} t_j/t_{j-m+i} = \sum_{i=0}^{m-1} \sum_{j=0}^{k-1} q_j \cdots q_{j-i},$$

and estimating the inner sums on the right-hand side viewed as k -fold arithmetic means by the corresponding geometric means and their lower bounds resulting from (11), we thus obtain $V(1, \dots, 1) > \sum_{\mu=1}^m k \cdot (B/4m)^{\mu/k}$ as decisive lower bound for the right-hand side of (6). Finally we set $(B/4m)^{1/k} = z$, with $k = (\ln B - \ln(4m))/\ln z$, which yields

$$\begin{aligned} E(T(g, s)) &> 1/m \cdot ((z + z^2 + \dots + z^m)/\ln z)(\ln B - \ln(4m)) \\ &\geq f_m(z_m) \cdot (\ln B - \ln(4m)). \end{aligned}$$

Comparing this with the expected deposit determined at the beginning of this subsection, we see that for any $c < \alpha_m = f_m(z_m)$ player F will loose

$$E(T(g, s)) - E(c \cdot g) > \alpha_m(\ln B - \ln(4m)) - c \cdot \ln B - c > 0$$

for sufficiently large B . This completes our proof of Theorem 2.2.

2.4 Analysis of the Cost Functions and Optimal Ratios

Let us now have a closer look at the functions f_m defined by (4). Taking the logarithmic derivative and multiplying with z we obtain

$$\frac{z \cdot f'_m(z)}{f_m(z)} = \frac{(m+1)z^m - 1}{z^m - 1} - \frac{z}{z-1} - \frac{1}{\ln z} = \frac{m \cdot z^m}{z^m - 1} - \frac{1}{z-1} - \frac{1}{\ln z}, \quad (12)$$

which has a unique zero $z_m \in (1, \infty)$, where f_m attains its unique minimum, since on this domain $1/\ln z$ is decreasing from ∞ to 0, while the other part $h(z) = m + m/(z^m - 1) - 1/(z - 1)$ is monotonically *increasing* to its limit m , hence $1/\ln z_m < m$, $z_m^m > e$. Here $h'(z) = -m^2 z^{m-1}/(z^m - 1)^2 + 1/(z - 1)^2 > 0$ follows from $h'(z)(z^m - 1)^2(z - 1)^2/z^m = -m^2(z - 2 + 1/z) + (z^m - 2 + 1/z^m)$, where we use $4m^2 \cdot \sinh^2(y) < 4 \cdot \sinh^2(my)$ with $y = \ln z$.

To study the behavior of these z_m and of the $\alpha_m = f_m(z_m)$ in greater detail, we use $w_m = z_m^m$, so the right-hand side of (12) set to zero yields the equation

$$\frac{w}{w-1} - \frac{1}{m(w^{1/m}-1)} = \frac{1}{\ln w} \quad \text{for } w = w_m = z_m^m, \quad (13)$$

and $\ln w = v$, $w^{1/m} = \exp(v/m)$ with $v < m(w^{1/m}-1) < v/(1-v/2m)$ show

$$1/(m(w^{1/m}-1)) = 1/\ln w - \theta/2m \quad \text{with some } \theta \in (0, 1). \quad (14)$$

In the limit for $m \rightarrow \infty$, (13) thus becomes $w/(w-1) = 2/\ln w$, and so the solutions w_m of (13) are converging to the simple zero $\omega > 1$ of the function $d(w) = \ln w - 2 + 2/w$, with $d'(w) = 1/w - 2/w^2 \geq 0$ for $w \geq 2$. Leaving aside the zero at 1 and searching between the minimum value $d(2) = -0.306\dots$ and $d(5) = \ln 5 - 1.6 > 0$, one finds the unique zero $\omega = 4.9215536\dots$, lying above the inflection point at 4, with $d(4) = -0.1137\dots$ and $d'(w) < d'(4) = 1/8$ for all $w > 4$. More precisely, (13) and (14) yield $w_m/(w_m-1) = 2/\ln w_m - \theta/2m$,

$$d(w_m) = \ln w_m - 2 + 2/w_m = -\ln(w_m)(1 - 1/w_m) \cdot \theta/2m < 0,$$

thus $w_m < \omega$ and $d(w_m) > -(\ln \omega)(1 - 1/\omega)/2m = -(\ln \omega)^2/4m > -0.635/m$. For $m \geq 6$, this implies $w_m > 4$, therefore $w_m > \omega - (0.635/m)/d'(4)$, and the same holds for $m < 6$ (as verified numerically), hence

$$\omega - 5.08/m < w_m < \omega \quad \text{for all } m, \quad (15)$$

and $z_m = \sqrt[m]{w_m} = 1 + c/m + O(1/m^2)$ with $c = \ln \omega = 1.593624\dots$.

Finally let us analyze the growth of the $\alpha_m = f_m(z_m)$ for $m \rightarrow \infty$. Since (13) implies $m(z_m - 1) \ln w_m/(w_m - 1) = m(z_m - 1) + \ln w_m - m(z_m - 1) \ln w_m$, we can rewrite the expression $\alpha_m = f_m(z_m) = z_m \cdot (z_m^m - 1)/((z_m - 1) \ln(z_m^m))$ resulting from (4) as

$$\alpha_m = \frac{m \cdot z_m}{m(z_m - 1) + \ln w_m - m(z_m - 1) \ln w_m}.$$

With $m(z_m - 1) = \ln w_m + O(1/m)$ from (14) and $\ln w_m = \ln \omega - O(1/m)$ from (15), this yields the following quantitative supplement to our Theorem 2.2, stating *asymptotically linear growth of the optimal ratios*

$$\alpha_m = \frac{m + O(1)}{2 \cdot \ln \omega - \ln^2 \omega + O(1/m)} = m \cdot \tau + O(1), \quad (16)$$

$$\text{with } \tau = 1/(2 \cdot \ln \omega - \ln^2 \omega) = \omega/(2 \cdot \ln \omega) = 1.54413865\dots$$

A similar linear growth result has been obtained in Sect. 6 of [6] for the cow-path problem on m lanes, in that case with the doubled factor $\kappa = 2\tau$.

By pursuing the foregoing analysis with greater care, one can show that the convergence $w_m \rightarrow \omega$, see (15), is indeed not faster than of order $1/m$, and the final $O(1)$ in (16) can be sharpened to the form $\sigma - O(1/m)$ with a certain constant $\sigma = 1.230388\dots$, where its approximate value appears as a byproduct of the numerical data given in Table 2 of the Appendix. Moreover we can also infer from that table that the convergence of the values of $m(z_m - 1)$ to their limit $\ln \omega = 1.59362\dots$ is *from above*.

3 Worst Case Lower Bound Proofs

Our proof of Theorem 2.1 (including that of Theorem 1.1) will be indirect: assuming $\rho < \beta_m$ will lead to a contradiction. We begin with a technical reformulation of that $\limsup \rho$ belonging to any given F -strategy $((s_j, t_j) : j \in \mathbb{N})$. The quotients $T(g, s)/g$ are becoming large for goals $g = t_j + \varepsilon$ with positive ε tending to zero and $s = s_j$, then approaching $T_n(j)/t_j$, where we define

$$n(l) := \min\{\nu > l : s_\nu = s_l\}, \quad (17)$$

and write $T_n = t_0 + \cdots + t_n$ for any n . By $t_j \rightarrow \infty$, we therefore also have $\rho = \limsup_{j \rightarrow \infty} T_n(j)/t_j$. Moreover we shall exploit that we may restrict our indirect proof to F -strategies satisfying $t_j \leq t_{j+1}$ for all j , due to the following

Lemma 3.1. *To every F -strategy $st = ((s_j, t_j) : j \in \mathbb{N})$ with limit ratio $\limsup_{j \rightarrow \infty} T_n(j)/t_j = \rho$ there exists some $st^* = ((s_j^*, t_j^*) : j \in \mathbb{N})$ with monotonicity $t_j^* \leq t_{j+1}^*$ for all j and $\limsup_{j \rightarrow \infty} T_{n^*}^*(j)/t_j^* = \rho^* \leq \rho$.*

Proof. If strategy st does not satisfy this monotonicity, we consider minimal j with $t_j > t_{j+1}$ with selection indices $s_j = i$ and $s_{j+1} = i' \neq i$ (since the subsequence $(t_j : j \in J_i)$ is strictly increasing) and alter st to a new F -strategy st' defined as follows: we exchange t_j and t_{j+1} setting $t'_j := t_{j+1}$, $t'_{j+1} := t_j$, while keeping $t'_l = t_l$ for all other l , and exchange the roles of i and i' above $j+1$, setting $s'_l := i'$ for $l > j+1$, $s_l = i$, $s'_l := i$ for $l > j+1$, $s_l = i'$, and $s'_l := s_l$ for all other l . Via (17), this s' will then induce new “next” indices $n'(l)$, but such that most of the quotients $T'_{n'(l)}/t'_l$ remain the same as before, including $T'_{n'(j)}/t'_j = T_{n(j+1)}/t_{j+1}$ and $T'_{n'(j+1)}/t'_{j+1} = T_{n(j)}/t_j$, with the only possible exception of an index $r < j$ with $n(r) = j$, but in that case

$$T'_j/t'_r = (t_0 + \cdots + t_{j-1} + t_{j+1})/t_r < T_{n(r)}/t_r.$$

It may be necessary to carry out an infinite sequence of such changes (then formally to be specified as a recursive definition), but as the t_j are tending to infinity, this will establish that monotonicity up to any fixed index j within a finite number of such steps. Since none of the quotients $T_{n(j)}/t_j$ gets ever increased, the $\limsup \rho^*$ of the quotients $T_{n^*}^*(j)/t_j^*$ of the resulting *limit strategy* st^* cannot be greater than the initial ρ . $\square$

So let us now consider an F -strategy $st = ((s_j, t_j) : j \in \mathbb{N})$ for adaptive raising on m tracks with $t_j \leq t_{j+1}$ for all j and such $\limsup \rho < \beta_m$. Then we pick some β with $1 \leq \rho < \beta < \beta_m$, choose some large $k \geq \max_i \min J_i$ with $n(l) > k \implies T_{n(l)}/t_l \leq \beta$, and consider for any fixed $j \geq k$ the m indices $q(i) := \max\{q \in J_i : q < j + m\}$, for $i < m$, satisfying $n(q(i)) \geq j + m$. As the J_i form a partition of $\mathbb{N}$, these $q(i)$ are m different integers $< j + m$, so we can use the minimal $\mu < m$ with $q(\mu) \leq j$ to infer from $T_{j+m} \leq T_{n(q(\mu))}$ and $t_{q(\mu)} \leq t_j$ that $T_{j+m}/t_j \leq T_{n(q(\mu))}/t_{q(\mu)} \leq \beta$. Altogether, this implies

$$t_0 + \cdots + t_{k+n+m} \leq \beta \cdot t_{k+n} \quad \text{for all } n \geq 0. \quad (18)$$

From these inequalities we will now conclude that there exists an $a > 0$ and a positive real sequence $x_0, x_1, x_2, \dots$ satisfying corresponding equations

$$a + x_0 + \cdots + x_{n+m} = \beta \cdot x_n \quad \text{for all } n \geq 0. \quad (19)$$

First we normalize (18) by setting $a = (t_0 + \cdots + t_{k-1})/t_k$ and $x_l = t_{k+l}/t_k$ for $l \geq 0$, whence the set $X \subset \mathbb{R}_{>0}^\infty$ of positive sequences $(x_l : l \in \mathbb{N})$ satisfying the linear inequalities $x_0 \leq 1$ and

$$a + x_0 + \cdots + x_{n+m} \leq \beta \cdot x_n \quad \text{for all } n \geq 0 \quad (20)$$

is not empty. Moreover this X is *compact*, since that constant $a > 0$ and $x_0 \leq 1$ imply the bounds $a/(\beta - 1) \leq x_n$ and $x_{n+1} < \beta \cdot x_n$, thus $x_n \leq \beta^n$ by induction, and (20) describes intersections with closed half spaces. So X also contains a sequence $(x_l : l \in \mathbb{N})$ with x_0 being *minimal*, which then must satisfy the equations in (19), because otherwise any defect “ $< \beta \cdot x_n$ ” in (20) for some n would admit to decrease x_n a bit, thereby causing a defect “ $< \beta \cdot x_{n-1}$ ” in the preceding condition, and so forth down to a decrease of x_0 which, however, is excluded by its minimality.

We finish the indirect proof of Theorem 2.1 by subtracting the equations (19) for $n+1$ and n , which yields the linear recurrence $x_{n+m+1} = \beta \cdot x_{n+1} - \beta \cdot x_n$, and proceed as in Sect. 2 of [1] to arrive at the desired contradiction: For $\beta < \beta_m$, see (3), the characteristic polynomial $y^{m+1} - \beta \cdot y + \beta$ of this recurrence has only *simple* roots $v_0, v_1, \dots, v_m$, and no positive real root; more precisely these are $\lfloor (m+1)/2 \rfloor$ pairs of conjugate roots of distinct moduli, since $|v_\mu| = |v_\nu|$ implies $|v_\mu - 1| = |v_\nu - 1|$, plus one negative root, if m is even. Expressing the x_n from (19) as linear combination $x_n = \sum_{\mu=0}^m \gamma_\mu \cdot v_\mu^n$ of the standard solutions thus shows that $x_n > 0$ for all n is impossible. $\square$

References

- [1] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins, *Searching in the plane*. Information and Computation **106** (1993), 234–252.
- [2] A. Beck and D. J. Newman, *Yet more on the linear search problem*. Israel J. Math. **8** (1970), 419–429.
- [3] R. Bellman, *A minimization problem*. Bull. AMS **62** (1956), 270. — *An optimal search problem*. — problem 63–9 in SIAM Rev. **5** (1963), 274.
- [4] S. Gal, *Minimax solutions for linear search problems*. SIAM J. Appl. Math. **27** (1974), 17–30.
- [5] M. Y. Kao, Y. Ma, M. Sipser, and Y. Yin, *Optimal constructions of hybrid algorithms*. J. of Algorithms **29** (1998), 142–164; — also in *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms* (1994), 372–381.
- [6] M. Y. Kao, J. H. Reif, and S. R. Tate, *Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem*. Information and Computation **131** (1997), 63–80; — also in *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms* (1993), 441–447.

Appendix

Here we display some numerical data illustrating the analysis of Subsection 2.4. Table 1 compares the game theoretic optimal ratios α_m with the worst case ratios β_m for small m , showing savings of about 32 to 41 percent; (16) and (3) imply that the quotients α_m/β_m are converging to $\tau/e = 0.56805\dots$. Table 2 shows the minimum points z_m of the f_m , the solutions w_m of equation (13), their logarithms and the “approximate logarithms” $m(z_m - 1)$ for selected values of m , especially also for large m in order to demonstrate the limiting behavior of the deviations $\sigma_m = \alpha_m - m \cdot \tau$ in (16), and of those $m(z_m - 1)$.

Table 1. Optimal ratios compared

m	α_m	β_m
1	2.7182818	4.0000000
2	4.2848795	6.7500000
3	5.8385908	9.4814814
4	7.3880617	12.2070312
5	8.9356038	14.9299200
6	10.4821048	17.6513846
7	12.0279794	20.3719975

Table 2. Adaptive raising on m tracks: Optimal factors for the geometric progressions and some related quantities with their asymptotical behavior for large values of m .

m	z_m	$w_m = z_m^m$	$\ln w_m$	$m(z_m - 1)$	σ_m
1	2.71828183	2.71828183	1.00000000	1.71828183	1.17414318
2	1.83503707	3.36736104	1.21412936	1.67007413	1.19660227
3	1.54962339	3.72116127	1.31403579	1.64887018	1.20617489
4	1.40922432	3.94385120	1.37215771	1.63689729	1.21150715
5	1.32583930	4.09689133	1.41022847	1.62919651	1.21491062
6	1.27063753	4.20852654	1.43711260	1.62382518	1.21727291
7	1.23140919	4.29355217	1.45711440	1.61986436	1.21900889
8	1.20210283	4.36046644	1.47257903	1.61682262	1.22033866
9	1.17937924	4.41449781	1.48489408	1.61441315	1.22138997
10	1.16124573	4.45903929	1.49493334	1.61245729	1.22224204
20	1.08016646	4.67534563	1.54230309	1.60332911	1.22620345
50	1.03195159	4.81910205	1.57258761	1.59757925	1.22868616
100	1.01595614	4.86963100	1.58301816	1.59561428	1.22953245
1000	1.00159382	4.91629704	1.59255561	1.59382440	1.23030237
10000	1.00015936	4.92102732	1.59351731	1.59364429	1.23037980
100000	1.00001594	4.92150100	1.59361356	1.59362626	1.23038755
1000000	1.00000159	4.92154837	1.59362319	1.59362446	1.23038832

A Competitive Algorithm for the General 2-Server Problem

René A. Sitters¹, Leen Stougie^{1,3}, and Willem E. de Paepe²

¹ Department of Mathematics and Computer Science

² Department of Technology Management

Technische Universiteit Eindhoven

P.O.Box 513, 5600 MB Eindhoven, The Netherlands

{r.a.sitters, l.stougie, w.e.d.paepe}@tue.nl

³ CWI, P.O.Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract. We consider the general on-line two server problem in which at each step both servers receive a request, which is a point in a metric space. One of the servers has to be moved to its request. The special case where the requests are points on the real line is known as the CNN-problem. It has been a well-known open question if an algorithm with a constant competitive ratio exists for this problem. We answer this question in the affirmative sense by providing the first constant competitive algorithm for the general two-server problem on any metric space.

1 Introduction

In the *general k -server problem* we are given servers $s_1, \dots, s_k$, each of which moving in a metric space $\mathbb{M}_i$. Requests $r \in \mathbb{M}_1 \times \mathbb{M}_2 \times \dots \times \mathbb{M}_k$ are presented on-line one by one. Thus, a request is a k -tuple $r = (z_1, z_2, \dots, z_k)$ and it is served by moving one of the servers s_i to his corresponding point z_i . The decision which server to move is irrevocable and has to be taken without any knowledge about future requests. The cost of moving server s_i to z_i is equal to the distance travelled by s_i from his current location to z_i . The objective is to minimize the total cost to serve all given requests. The performance of an on-line algorithm is measured through competitive analysis. An online algorithm is c -competitive if, for any request sequence σ , the algorithm's cost are at most c times the cost of the optimal solution of the corresponding off-line problem plus some additive constant independent of σ .

The general k -server problem is a natural generalization of the well-known *k -server problem* for which $M_1 = M_2 = \dots = M_k$ and $z_1 = z_2 = \dots = z_k$ at each time step. The k -server problem was introduced by Manasse, McGeoch and Sleator [9], who proved a lower bound of k on the competitive ratio of any deterministic algorithm for any metric space with at least $k + 1$ points and posed the well-known k -server conjecture saying that there exists a k -competitive algorithm for any metric space. The conjecture has been proved for $k = 2$ [9] and some special metric spaces [2][3]. For $k \geq 3$ the current best upper bound of $2k - 1$ is given by Koutsoupias and Papadimitriou [7].

The *weighted k -server problem* turns out to be much harder. In this problem a weight is assigned to each server and the total cost is the sum of the weighted distances. Fiat and Ricklin [6] prove that for any metric space there exists a set of weights such that the competitive ratio of any deterministic algorithm is at least $k^{\Omega(k)}$. For a uniform metric space, on which the problem is called the weighted paging problem, Feuerstein et al. [5] give a 6.275-competitive algorithm. For $k = 2$ Chrobak and Sgall [4] provided a 5-competitive algorithm and proved that no better competitive ratio is possible.

A weighted k -server algorithm is called competitive if the competitive ratio is independent of the weights. For a general metric space no competitive algorithm is known yet even for $k = 2$. It is easy to see that the general k -server problem is a generalization of the weighted k -server problem as well.

The general 2-server problem in which both servers move on the real line has become well-known as the CNN-problem¹. Koutsoupias and Taylor [8] emphasize the importance of the CNN-problem as one of the simplest problems in a rich class of so-called sum of task systems [1]. In the sum-problem each system gets a task (request) and only one system has to fulfill its task. Such problems form a richer class than the k -server problem for modelling purposes (see [8]).

Koutsoupias and Taylor [8] prove a lower bound of $6 + \sqrt{17}$ on the competitive ratio of any deterministic on-line algorithm for the general 2-server problem, through an instance of the weighted 2-server problem on the real line. They also conjecture that the *work function algorithm* has constant competitive ratio for the general 2-server problem. It seems to be a bad tradition of multiple-server problems to keep unsettled conjectures. For the general 2-server problem the situation was even worse than for the k -server problem: the question if any algorithm exists with constant competitive ratio remained unanswered.

In this paper we answer this question affirmatively, by designing an algorithm and prove an upper bound of 100,000 on its competitive ratio. The constant is huge, but our goal was indeed to settle the question. We believe that our result gives new insight in the problem and will lead to more and much better algorithms for the general k -server problem in the near future.

Optimal off-line solutions of metrical task systems can easily be found by dynamic programming (see [1]), which yields a $O(n^2)$ time algorithm for the general two server problem. As a result our algorithm can be implemented to work in polynomial time.

2 A Competitive Algorithm

A request is given by a pair $r_i = (x_i, y_i)$ with x_i a point in metric space $\mathbb{M}_1 = (\mathbb{X}, d_x)$ and y_i in $\mathbb{M}_2 = (\mathbb{Y}, d_y)$. We suppress the sub-indices on the distance since it will always be clear from the context which of the two measures is meant. We denote the two servers as the x - and y -server. The distance $\delta : (\mathbb{M}_1 \times \mathbb{M}_2)^2 \rightarrow \mathbb{R}$ is defined as $\delta((x_1, y_1), (x_2, y_2)) = d(x_1, x_2) + d(y_1, y_2)$.

¹ The name CNN-problem was suggested by Gerhard Woeginger.

We say that an online algorithm for the general two server problem is *lazy* if at any request only the server that services the request moves, and halts in the request. Our online algorithm is not lazy but it is easy to make it into a lazy algorithm by treating all moves made by the algorithm as virtual, and move a server only for real when it serves the next request. The triangle inequality ensures that the real movement is no more than the sum of the virtual moves. Moreover, we allow that the virtual moves are made to points outside the metric space. This is useful when we want to make a virtual move to a point between two points a_1 and a_2 of the metric space. This can easily be done by adding a new point a to the metric space and defining for any other point z in the metric space $d(z, a) = \min\{d(z, a_1) + d(a_1, a), d(z, a_2) + d(a_2, a)\}$, where choose $d(a, a_1)$ and $d(a, a_2)$ such that their sum is $d(a_1, a_2)$.

A tour is defined as a directed path in the product space $\mathbb{M}_1 \times \mathbb{M}_2$, and we denote the length of a tour T by $|T|$. We say that a tour T serves the request sequence $r_1, \dots, r_n$ if there is a sequence of pairs $(\bar{x}_1, \bar{y}_1), \dots, (\bar{x}_n, \bar{y}_n)$, that lie on T in this order, such that for all $j \in \{1, \dots, n\}$, $\bar{x}_j = x_j$ or $\bar{y}_j = y_j$.

Given a configuration $(\hat{x}_0, \hat{y}_0)$ and a request sequence $r_1, r_2, \dots, r_n$ we denote X_{0j} ($0 < j \leq n$) as the length of the path $\hat{x}_0, x_1, \dots, x_j$, and X_{ij} ($1 \leq i < j \leq n$) as the length of the path $x_i, x_{i+1}, \dots, x_j$. We denote Y_{ij} ($0 \leq i < j \leq n$) in a similar way.

2.1 Basic Properties and a Sketch

We state two important properties of the general 2-server problem, which have inspired the design of our on-line algorithm. They are stated in Lemmas 1 and 2 and illustrated in Figure 2.1. The figure shows a part of an instance of the CNN-problem consisting of 7 requests. Five possible tours are shown. Tour T_D serves all request with the the x-server, starting from the x-coordinate of the first request, hence $|T_D| = X_{1,7}$. Similarly, $|T_E| = Y_{1,7}$. The other three tours each have length much smaller than $\min\{X_{1,7}, Y_{1,7}\}$. Tour T_A is relative far apart from tours T_B and T_C , whereas the tours T_B and T_C are relative close to one another. Lemma 1 states the impossibility of the existence of more than two



Fig. 1. Part of an instance of the CNN-problem and several feasible tours.

small tours, serving the same request sequence, which are mutually relatively far apart. First we give a notion of closeness of two tours which we employ in this paper. Let T_A and T_B be tours in $\mathbb{M}_1 \times \mathbb{M}_2$. We say that they are *connected* if there are points $x \in \mathbb{X}$ and $y \in \mathbb{Y}$ such that x and y are points on T_A as well as on T_B . (We do not impose that (x, y) is on the tours).

Lemma 1. *Given are three tours T_A , T_B and T_C , each serving a request sequence $r_1, r_2, \dots, r_n$. If non of the three pairs $\{T_A, T_B\}$, $\{T_B, T_C\}$ and $\{T_A, T_C\}$ is connected, then $|T_A| + |T_B| + |T_C| \geq \min\{X_{1n}, Y_{1n}\}$.*

Proof. Assume without loss of generality that the x-server of T_A shares no request with the x-server of T_B and no request with the x-server of T_C . If the y-server of T_A serves all requests then the lemma obviously holds. So assume request r_i is served by the x-server of T_A . Then T_B and T_C must serve point y_i . But this means that the x-servers of T_B and T_C do not share a request. Hence, the three x-servers share no request. Thus, each request must be served by at least two y-servers and for each two consecutive requests there is a y-server that serves both requests. $\square$

Lemma 2. *If T_A and T_B are connected and (x_{j_1}, y_{j_1}) and (x_{j_2}, y_{j_2}) are points on respectively T_A and T_B , then $\delta((x_{j_1}, y_{j_1}), (x_{j_2}, y_{j_2})) \leq |T_A| + |T_B|$.*

Proof. Let $x \in \mathbb{X}$ and $y \in \mathbb{Y}$ be points that connect both tours. Then,

$$\begin{aligned} \delta((x_{j_1}, y_{j_1}), (x_{j_2}, y_{j_2})) &= d(x_{j_1}, x_{j_2}) + d(y_{j_1}, y_{j_2}) \\ &\leq d(x_{j_1}, x) + d(x, x_{j_2}) + d(y_{j_1}, y) + d(y, y_{j_2}) \\ &\leq |T_A| + |T_B|. \end{aligned}$$

$\square$

The idea behind our competitive algorithm is to try to remain close to an optimal tour, unless the optimal tour is relatively large. The algorithm works in phases, that are separate except that the end positions of the servers in one phase are their starting positions in the next phase. The algorithm is defined such that in each phase it is successful with respect to at least one of the following two goals:

Keeping the tour relatively short in comparison to an optimal tour that serves the same request sequence presented in the phase;

A substantial decrease in distance between the position of its own servers and those of an optimal tour at the end of the phase in comparison to the start of the phase.

In the beginning of each phase the algorithm chooses a reasonable strategy, which we call **BALANCE** and which is presented in the following subsection. While applying this strategy the algorithm keeps track of short tours for serving the requests in the phase. If such a short tour emerges then the algorithm tends to move its servers to the server positions of this short tour. If only one such a

tour exists or if all short tours are relatively near to each other then the second goal stated above is reached and the phase stops. We know from Lemma 1 that at most two of them may be relatively far apart. In case indeed two such tours exist then the algorithm needs to move its servers to the positions the servers have on one of the short tours. It chooses the one that it is in a certain way nearest.

This choice may turn out to be unfortunate, which is illustrated in Figure 2.1: all requests are given in turn in points 1 and 2. To stay competitive an algorithm, starting the phase from v , must either move to point a or to point b . On an optimal tour the requests could be served at zero cost if this tour started the phase in a or b . If the algorithm moves its servers to a while the servers on an optimal tour appear to be in b then the distance to the optimal tour has even increased after this move. Similarly, if the algorithm moves its servers to b , the optimal servers may turn out to be in a .

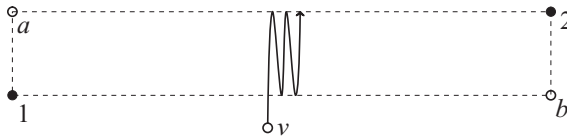


Fig. 2. A difficult choice.

We define the strategy COMPETE, described in Section 2.3, to avert this potential danger. The achievement of COMPETE is that, at the end of the phase, small tours, if any exist, have their servers on positions which are concentrated around a single point in $\mathbb{M}_1 \times \mathbb{M}_2$. Once this is achieved the phase is finished by moving the servers in the direction of the positions on the shortest tour at the end of the phase.

It will be clear from the sketch of the algorithm that in each phase, ONLINE carefully chooses its steps in order to stay competitive. If at some moment the requested points are relatively far away for both servers, then the algorithm is forced to make a large step. If this step is much larger than the sum of all preceding steps in the phase, then the phase is terminated immediately and the request is considered as the first request in the new phase.

The precise description of the algorithm, which is found in Section 2.4, is rather technical. The basic ideas have just been sketched, but their implementation allows freedom for specific choices. The choices we made are motivated by no more and no less than the fact that they allowed us to prove the desired competitiveness. Many other choices are possible, also alternatives for BALANCE and COMPETE and may give even better competitive ratios. However, the main goal of our research was to prove the conjecture that a constant competitive algorithm for the general two-server problem exists. We leave it to future research to find better competitive ratios.

2.2 Algorithm Balance

Algorithm BALANCE is applied at the beginning of each phase of ONLINE and within the subroutine COMPETE. We describe it on a request sequence $r_1, \dots, r_n$ starting from the positions $(\hat{x}_0, \hat{y}_0)$. Let S_j^x and S_j^y be the total costs made by, respectively, the x- and the y-server after serving request r_j , and $S_j := S_j^x + S_j^y$. Let $(\hat{x}_j, \hat{y}_j)$ be the server positions after serving request r_j .

BALANCE:

If $S_j^x + d(\hat{x}_j, x_{j+1}) \leq S_j^y + d(\hat{y}_j, y_{j+1})$, then move the x-server to request x_{j+1} . Else move the y-server to request y_{j+1} .

The following lemma gives an upper bound on the cost of BALANCE.

Lemma 3. $S_j \leq 2 \max\{S_j^x, S_j^y\} \leq \min\{X_{0j}, Y_{0j}\} \quad \forall j \in \{0, \dots, n\}$.

Proof. Clearly, $S_j^x \leq X_{0j}$ and $S_j^y \leq Y_{0j}$. Let request r_i , $i \leq j$, be the last request served by the x-server. Then, by definition, $S_j^x = S_i^x \leq S_{i-1}^y + d(\hat{y}_{i-1}, y_i) \leq Y_{0i} \leq Y_{0j}$. Hence, $S_j^x \leq \min\{X_{0j}, Y_{0j}\}$. Similarly it is shown that $S_j^y \leq \min\{X_{0j}, Y_{0j}\}$. $\square$

Lemma 4. $S_{j+1} \leq 3S_j + \min\{d(\hat{x}_0, x_{j+1}), d(\hat{y}_0, y_{j+1})\}$, $\forall j \geq 1$.

Proof. Without loss of generality we assume that $d(\hat{x}_0, x_{j+1}) \leq d(\hat{y}_0, y_{j+1})$. By definition of BALANCE we have $S_{j+1} \leq S_j + \min\{S_j^x + d(\hat{x}_j, x_{j+1}), S_j^y + d(\hat{y}_j, y_{j+1})\} \leq S_j + S_j^x + d(\hat{x}_j, x_{j+1}) \leq S_j + 2S_j^x + d(\hat{x}_0, x_{j+1}) \leq 3S_j + d(\hat{x}_0, x_{j+1})$. $\square$

2.3 Algorithm Compete

We denote the positions of the servers at the beginning of algorithm COMPETE by $(\hat{x}, \hat{y})$. The behavior of the algorithm depends on a parameter $(x^*, y^*) \in \mathbb{M}_1 \times \mathbb{M}_2$, which is regarded as the position of the servers on the alternative short tour. Define $\Delta = \frac{1}{2} \max\{d(\hat{x}, x^*), d(\hat{y}, y^*)\}$. We describe the algorithm in case $\Delta = \frac{1}{2} d(\hat{x}, x^*)$. Interchanging the role of x and y gives the description in case $\Delta = \frac{1}{2} d(\hat{y}, y^*)$. The algorithm works in phases. The only information it takes to the next phase is the current position of its servers. We describe a generic phase on a sequence of requests $r_1, r_2, \dots$. Occasionally both servers make a move after the release of a request. Let S_j^x and S_j^y be the distance travelled by respectively the x- and y-server during the current phase, after both servers made their moves upon the release of request r_j and $(\hat{x}_j, \hat{y}_j)$ their positions at the same time.

A phase of COMPETE(x^*, y^*) :

- (1) Apply BALANCE, until the release of a request r_j with $d(\hat{x}, x_j) \geq \Delta$ in which case go to Step (2).

- (2) Apply the following three steps.
- a. If $d(\hat{y}, y_j) < d(\hat{x}, x_j)$, then serve y_j , else serve x_j .
 - b. If the x-server has not served any request in the phase, and $j > 1$, then move the x-server over a distance S_{j-1}^y towards x_{j-1} .
 - c. Start a new phase.

The following lemma shows that if the two alternative short tours remain small, then COMPETE remains competitive with the sum of the two short tours. We exploit that COMPETE starts with its servers in the same position as one of the two short tours.

Lemma 5. *Given any request sequence let T be the tour made by Compete (x^*, y^*) starting from position $(\hat{x}, \hat{y})$. Let T_1 and T_2 be tours, both serving the same request sequence and starting in respectively $(\hat{x}, \hat{y})$ and (x^*, y^*) . If $|T_1| + |T_2| < \Delta$, then $|T| \leq 10(|T_1| + |T_2|)$.*

Proof. We assume that $\Delta = \frac{1}{2}d(\hat{x}, x^*)$, the other case being similar. Let $(x^{(1)}, y^{(1)})$ and $(x^{(2)}, y^{(2)})$ be the final positions of respectively T_1 and T_2 . We give an auxiliary request $(x^{(2)}, y^{(1)})$ at the end, which is served in T_1 and T_2 at no extra cost. Since we assume $|T_1| + |T_2| < \Delta$ we have $d(\hat{x}, x^{(2)}) \geq 2\Delta - d(x^*, x^{(2)}) > \Delta$ and therefore the last phase will end properly, i.e. with a $\langle 2 \rangle$ step.

Consider an arbitrary phase in the algorithm, and suppose this phase contains n requests. We use $(\hat{x}_0, \hat{y}_0)$ for the positions of the servers at the beginning of the phase (in the first phase $(\hat{x}_0, \hat{y}_0) = (\hat{x}, \hat{y})$). Define $S = S_n^x + S_n^y$. Let C_1^x (C_1^y) and C_2^x (C_2^y) be the total cost of the x-server (y-server) on, respectively, T_1 and T_2 in the phase and define $C_1 = C_1^x + C_1^y$, $C_2 = C_2^x + C_2^y$, and $C = C_1 + C_2$. The positions of the servers in T_1 after serving request r_j , $j \in \{0, \dots, n\}$ are denoted by (x'_j, y'_j) .

We define a potential at the beginning of the phase as $\Psi = 3d(\hat{x}_0, x'_0)$, whence the increase in potential during this phase is $\Delta\Psi = 3d(\hat{x}_n, x'_n) - 3d(\hat{x}_0, x'_0)$. We will prove that $S \leq 10C - \Delta\Psi$. This proves the lemma since taking the sum over all phases yields $|T| \leq 10(|T_1| + |T_2|) - 3d(\hat{x}_N, x'_N) \leq 10(|T_1| + |T_2|)$, where $(\hat{x}_N, x'_N)$ denote the final positions of the x-servers of T and T_1 in the last phase.

Given the condition of the lemma, request r_n is served by the y-servers of T_1 , since otherwise $|T_1| \geq d(\hat{x}, x_n) \geq \Delta$. For the same reason, r_n is served by the y-server of T , since otherwise, by definition of Step 2a of COMPETE, $d(\hat{y}, y_n) \geq d(\hat{x}, x_n) \geq \Delta$, implying again $|T_1| \geq \Delta$. Hence, $\hat{y}_0 = y'_0$ and $\hat{y}_n = y'_n = y_n$. To simplify notation we define $y_0 = \hat{y}_0$ ($= y'_0$). By a similar argument $r_1, \dots, r_{n-1}$ must be served by the y-server of T_2 . We distinguish three cases.

Case 1. The y-server of T_1 serves a request r_k with $k \in \{1, \dots, n-1\}$. In this case, $C_1^y \geq d(y_0, y_k) + d(y_k, y_n)$ and $C_2^y \geq d(y_1, y_k) + d(y_k, y_{n-1})$. By the triangle inequality we have $d(y_0, y_k) + d(y_k, y_n) + C_2^y \geq d(y_0, y_1) + d(y_{n-1}, y_n)$. Hence, $C_1^y + 2C_2^y \geq d(y_0, y_1) + d(y_{n-1}, y_n) + C_2^y \geq S_n^y$. By the use of BALANCE also $d(y_0, y_1) + C_2^y \geq S_n^x$. Clearly the increase in potential is bounded by $\Delta\Psi \leq 3(C_1^x + S^x)$. We obtain

$$\begin{aligned}
S &= 4S_n^x + S_n^y - 3S_n^x \\
&\leq 5C_1^y + 10C_2^y + 3C_1^x - \Delta\Psi \\
&\leq 10C - \Delta\Psi.
\end{aligned}$$

Case 2. The y -server of T_1 serves only r_0 and r_n and the x -server of COMPETE serves no request in the phase. In this case $C_1^y = d(y_0, y_n)$, $x'_n = x_{n-1}$ (for $n \geq 2$), and $S_n^x = S_{n-1}^y$. The increase in potential is $\Delta\Psi \leq 3(C_1^x - S_n^x)$. Therefore,

$$\begin{aligned}
S &= S_n^x + S_n^y \\
&\leq S_n^x + 2S_{n-1}^y + d(y_0, y_n) \\
&= 3S_n^x + d(y_0, y_n) \\
&\leq 3C_1^x - \Delta\Psi + d(y_0, y_n) \\
&\leq 3C - \Delta\Psi.
\end{aligned}$$

Case 3. The y -server of T_1 serves only r_0 and r_n and the x -server of COMPETE serves a request r_j with $j \in \{1, \dots, n-1\}$. Again, $C_1^y = d(y_0, y_n)$ and $x'_n = x_{n-1}$ (for $n \geq 2$). The increase in potential is $\Delta\Psi \leq 3(C_1^x - d(\hat{x}_0, x_1))$. Clearly $S_n^x \leq d(\hat{x}_0, x_1) + C_1^x$, and by definition of BALANCE also $S_{n-1}^y \leq d(\hat{x}_0, x_1) + C_1^x$. We obtain,

$$\begin{aligned}
S &= S_n^x + S_n^y \\
&\leq S_n^x + 2S_{n-1}^y + d(y_0, y_n) \\
&\leq 6C_1^x - \Delta\Psi + d(y_0, y_n) \\
&\leq 6C - \Delta\Psi.
\end{aligned}$$

□

2.4 Algorithm Online

Algorithm ONLINE works in phases. The only information taken from one phase to the next phase is the position of the servers at the end of the phase. Each phase starts by applying the simple BALANCE routine, until it becomes clear that there exists a short tour whose servers positions are not far from the starting position of ONLINE in the phase. At that moment, it makes an extra move to the positions of the servers on a short tour. If there is only one such a tour or every two such tours are relatively near to each other, then the phase ends. Otherwise ONLINE switches to the subroutine COMPETE. We notice that all these short tours do not need to start the phase in the same position as ONLINE. The phase ends with again an extra move. As announced in the sketch of the algorithm the phase can also end prematurely as soon as a request is presented that requires a relatively large move, which is defined in the algorithm by the *exception rule*.

We denote $v_0 = (O, O)$ as the starting position in phase 1, and v_i as the end position in phase i ($i \geq 1$), and hence the starting position in phase $i + 1$. The requests given in phase i are denoted by $r_{i1}, r_{i2}, \dots$. We describe a generic phase of ONLINE and to facilitate the exposition we suppress the subindex i of the phase. Thus, we denote the requests in the generic phase by $r_1, r_2, \dots$, the starting position by v_{-1} and the end position by v . We denote S_j ($j \geq 1$) as the cost of serving $r_1, r_2, \dots, r_j$ by BALANCE in Step (1) of the algorithm. Similarly, Z_j we denote the length of the tour by COMPETE in Step (2).

As indicated above the algorithm occasionally makes more moves than necessary to serve a next request r_j . We denote $(\hat{x}_j, \hat{y}_j)$ as the position of ONLINE after all moves are made, and we denote A_j as the cost of ONLINE until this moment. Additionally, we use the notation $(\hat{x}_0, \hat{y}_0)$ for the initial positions v_{-1} of the ONLINE servers in the phase. The constant η appearing in the description has value $\eta = 1/120$. For the tours T_A, T_B, T_C , and T_D in the description, v_A, v_B, v_C , and v_D represent their end points (end position of their servers).

A phase of ONLINE :

Apply the Steps (1),(2) and (3), with the following *exception rule*: If at any moment a request $r_j = (x_j, y_j)$ ($j \geq 2$) is released for which $\min\{d(\hat{x}_0, x_j), d(\hat{y}_0, y_j)\} \geq 4A_{j-1}$, then return to $v_{-1} = (\hat{x}_0, \hat{y}_0)$ and let (x_j, y_j) be the first request in a new phase.

- (1) Apply BALANCE. At the release of a request r_j for which there is a tour T_A , with $|T_A| < \eta S_j$ and $\delta(v_{-1}, v_A) \leq S_j/3$, return to v_{-1} after serving r_j , and continue with Step (2). Let r_{k_1} be this request.
- (2) Let T_B and T_C be tours, serving $r_1, \dots, r_{k_1}$, with $\max\{|T_B|, |T_C|\} < \eta S_{k_1-1}$, and $\delta(v_B, v_C) \geq 16\eta S_{k_1-1}$. If no such tours exist, then define $k_2 = k_1$ and continue with (3). Assume w.l.o.g. that $\delta(v_{-1}, v_B) \leq \delta(v_{-1}, v_C)$. Move the servers to v_B and apply COMPETE(v_C) until $Z_j \geq S_{k_1-1}$. Let r_{k_2} be this request. Move the servers back to v_{-1} .
- (3) Let T_D serve $r_1, \dots, r_{k_2}$ and have minimum length. If $|T_D| < \eta S_{k_1-1}$, then move from v_{-1} towards v_D over a distance $\frac{1}{3}S_{k_1-1}$. Start a new phase from this position.

We emphasize that Z_j , ($k_1 + 1 \leq j \leq k_2$) is the cost COMPETE(v_C) makes starting from v_B .

3 Competitive Analysis

In the competitive analysis we distinguish two types of phases. The phases of type II are those that terminated prematurely by the exception rule. The other phases are of type III. Notice that the last phase, which we denote by N , is of type III since any phase of type II is followed by at least one more phase.

For the analysis we introduce a potential function Φ that measures the distance of the position of the ONLINE servers to those of the optimal tour. The potential at the beginning of a phase i is $\Phi_{i-1} = 1000 \cdot \delta(v_{i-1}, v_{i-1}^*)$. We define the potential at the end of the last phase N to be zero, i.e., $\Phi_N = 0$.

We will prove in Lemma 8 that in each phase of type III either the ONLINE tour is relatively short with respect to the optimal tour over the requests in the phase or the potential function has decreased substantially over the phase. First, in the next two lemmas we will bound the length of the ONLINE tour in a type III

phase in terms of the bound from Lemma 1, which in a sense bounds the length of tours from below.

Consider an arbitrary phase of type III, and suppose it contains n requests $r_1, \dots, r_n$. As before, we denote X_{ij} ($0 \leq i < j \leq n$) as the length of the path $x_i, x_{i+1}, \dots, x_j$ (For $i = 0$ this path is $\hat{x}_0, x_1, \dots, x_j$). We denote Y_{ij} in a similar way. To simplify notation we write, in the following lemmas, S shortly for S_{k_1-1} , the length of the tour that BALANCE makes in Step (1) of ONLINE. By $|P|$ we denote the length of the ONLINE tour in the phase.

Lemma 6. *For each phase of type III, $\min\{X_{1k_1}, Y_{1k_1}\} \geq (1/12 - \eta/2)S$*

Proof. Assume w.l.o.g. $X_{1k_1} \leq Y_{1k_1}$ and notice that Lemma 3 implies $S_{k_1} \leq 2 \min\{X_{0k_1}, Y_{0k_1}\}$. Tour T_A cannot serve merely y-requests, since in that case $\delta(v_{-1}, v_A) \geq d(\hat{y}_0, y_1) - |T_A| = Y_{0k_1} - Y_{1k_1} - |T_A| \geq Y_{0k_1} - 2|T_A| \geq (1/2 - 2\eta)S_{k_1} > S_{k_1}/3$. Now, let x_j be the last x-request served by T_A . In this case we obtain $S_{k_1}/3 \geq \delta(v_{-1}, v_A) \geq d(\hat{x}_0, x_j) - |T_A| \geq X_{0k_1} - 2X_{1k_1} - |T_A| \geq (1/2 - \eta)S_{k_1} - 2X_{1k_1}$. Hence, $X_{1k_1} \geq (1/12 - \eta/2)S_{k_1} \geq (1/12 - \eta/2)S$. $\square$

Lemma 7. *For each phase of type III, $|P| < 191S$.*

Proof. Notice that Lemma 4 and the exception rule imply $S_{k_1} \leq 3S + \min\{d(\hat{x}_0, x_{k_1}), d(\hat{y}_0, y_{k_1})\} \leq 7S$. The cost made in Step (1) is at most $2S_{k_1} \leq 14S$ and the cost in Step (3) is at most $S/3$. If COMPETE is not applied in the phase, then only Step (1) and (3) add to the cost, whence $|P| \leq 14\frac{1}{3}S$. So assume COMPETE is applied.

First we bound $\delta(v_{-1}, v_B)$. Applying Lemma 6, $|T_A| + |T_B| + |T_C| \leq 7\eta S + \eta S + \eta S < \min\{X_{1k_1}, Y_{1k_1}\}$. Hence, these three tours do not satisfy the property of Lemma 1. By Lemma 2 the tours T_B and T_C cannot be connected, whence T_A must be connected to T_B or T_C . In the first case we have $\delta(v_{-1}, v_B) \leq \delta(v_{-1}, v_A) + |T_A| + |T_B|$, applying Lemma 3. Similarly if T_A is connected to T_C then $\delta(v_{-1}, v_C) \leq \delta(v_{-1}, v_A) + |T_A| + |T_C|$. Since we assumed $\delta(v_{-1}, v_B) \leq \delta(v_{-1}, v_C)$ we have $\delta(v_{-1}, v_B) \leq \delta(v_{-1}, v_A) + |T_A| + \min\{|T_B|, |T_C|\} \leq S_{k_1}/3 + \eta S_{k_1} + \eta S < 4S$.

Next we bound the cost made by COMPETE. Since by definition of ONLINE $Z_{k_2-1} < S$ the total cost after r_{k_2-1} is served is $A_{k_2-1} < 14S + 4S + S = 19S$. If r_{k_2} is served in Step (1) of COMPETE then the last step was a BALANCE step in a phase of COMPETE. Assume the phase started in the point (x, y) . By Lemma 4,

$$\begin{aligned} Z_{k_2} &< 3S + \min\{d(x, x_{k_2}), d(y, y_{k_2})\} \\ &\leq 3S + \delta((x, y), (\hat{x}_0, \hat{y}_0)) + \min\{d(\hat{x}_0, x_{k_2}), d(\hat{y}_0, y_{k_2})\} \\ &\leq 3S + 5S + 4 \cdot 19S \\ &= 84S. \end{aligned}$$

For the third inequality we used the exception rule. Now assume the request is served in Step (2) of COMPETE. The cost of the Step in (2)a is at most

$Z_{k_2-1} + \min\{d(\hat{x}_{k_1}, x_{k_2}), d(\hat{y}_{k_1}, y_{k_2})\}$, and the cost in $\langle 2 \rangle b$ is no more than Z_{k_2-1} . Therefore,

$$\begin{aligned} Z_{k_2} &\leq 3Z_{k_2-1} + \min\{d(\hat{x}_{k_1}, x_{k_2}), d(\hat{y}_{k_1}, y_{k_2})\} \\ &< 3S + \delta((\hat{x}_0, \hat{y}_0), (\hat{x}_{k_1}, \hat{y}_{k_1})) + \min\{d(\hat{x}_0, x_{k_2}), d(\hat{y}_0, y_{k_2})\} \\ &\leq 3S + 4S + 4 \cdot 19S \\ &= 83S. \end{aligned}$$

We conclude that the total cost in the phase is no more than $14S + 2(4S + 84S) + S/3 = 190\frac{1}{3}S$. $\square$

In the following crucial lemma we use $|P^*|$ as the length of an optimal tour to serve the request in the phase considered. We use v_{-1}^* and v^* for the starting and finishing positions of the servers on an optimal tour in the phase. In accordance with suppressing the subindex for the phase, we also write Φ_{-1} and Φ for the potential function, respectively at the beginning and at the end of the phase.

Lemma 8. *For each phase of type III, $2|P| < 10^5|P^*| - \Phi + \Phi_{-1}$.*

Proof. First assume the phase considered is not the last phase. We have to show that

$$F \equiv c_1|P^*| + c_2(\delta(v_{-1}, v_{-1}^*) - \delta(v, v^*)) > 2|P|, \quad (1)$$

with $c_1 = 10^5$ and $c_2 = 10^3$. We distinguish three cases.

Case 1. $|P^*| \geq \eta S$. In this case $\delta(v_{-1}, v_{-1}^*) - \delta(v, v^*) \geq -\delta(v, v_{-1}) - \delta(v^*, v_{-1}^*) \geq -S/3 - |P^*|$. Inequality (1) becomes in this case $F \geq c_1|P^*| - c_2(S/3 + |P^*|) = (c_1 - c_2)|P^*| - c_2S/3 > 382S > 2|P|$.

Case 2. $|P^*| < \eta S$ and COMPETE was not applied. From the proof of Lemma 7 $|P| \leq 14\frac{1}{3}S$. By definition of ONLINE the endpoint of any tour serving requests $r_1, \dots, r_{k_1-1}$ and with length smaller than ηS , must be at a distance greater than $S/3$ from point v_{-1} . In particular $\delta(v_{-1}, v_D) > S/3$. Since COMPETE was not applied $\delta(v_D, v^*) < 16\eta S$. By the triangle inequality

$$\delta(v_{-1}, v^*) \geq \delta(v_{-1}, v_D) - \delta(v_D, v^*) > \delta(v_{-1}, v_D) - 16\eta S,$$

and

$$\delta(v, v^*) \leq \delta(v_{-1}, v_D) - S/3 + \delta(v_D, v^*).$$

Hence,

$$\begin{aligned} \delta(v_{-1}, v_{-1}^*) - \delta(v, v^*) &\geq \delta(v_{-1}, v^*) - \delta(v, v^*) - \delta(v_{-1}^*, v^*) \\ &> S/3 - 32\eta S - |P^*| \\ &= 8\eta S - |P^*|. \end{aligned}$$

Hence, $F > c_1|P^*| + c_2(8\eta S - |P^*|) = (c_1 - c_2)|P^*| + 8c_2\eta S \geq 8c_2\eta S > 29S > 2|P|$.

Case 3. $|P^*| < \eta S$ and COMPETE was applied. Let T'_B be an optimal extensions of T_B , i.e. it starts in v_B , serves the requests $r_{k_1+1}, \dots, r_{k_2}$ and has

minimum length. Define T'_C similar as T'_B with respect to T_C . Now we apply Lemma 5 with the parameters $(\hat{x}, \hat{y}) = v_B$, $(x^*, y^*) = v_C$, $T_1 = T'_B$, $T_2 = T'_C$, and $\Delta = \frac{1}{2} \max\{d(\hat{x}, x^*), d(\hat{y}, y^*)\} \geq \delta(v_B, v_C)/4 \geq 4\eta S$. Lemma 5 implies

$$|T'_B| + |T'_C| \geq \min\{\Delta, S/10\} \geq 4\eta S. \quad (2)$$

Now let $\hat{T}_B$ and $\hat{T}_C$ be arbitrary tours that serve $r_1, \dots, r_{k_2}$ and are connected with T_B and T_C respectively. Assume that T_B and $\hat{T}_B$ both serve the requests x_i and y_j for some $i, j \in \{1, \dots, k_1\}$. A possible extension of T_B is to move the servers to (x_i, y_j) and serve the requests $r_{k_1+1}, \dots, r_{k_2}$ similar to $\hat{T}_B$. This implies $|T'_B| \leq |T_B| + |\hat{T}_B|$. Similarly $|T'_C| \leq |T_C| + |\hat{T}_C|$. Together with (2) this yields

$$|\hat{T}_B| + |\hat{T}_C| \geq |T'_B| + |T'_C| - |T_B| - |T_C| \geq 2\eta S. \quad (3)$$

Let T be an arbitrary tour that serves $r_1, \dots, r_{k_2}$ with $|T| < \eta S$. Since $|T_B| + |T_C| + |T| < 3\eta S < \min\{X_{1k_1}, Y_{1k_1}\}$, these three tours do not satisfy the property of Lemma 1. (To apply Lemma 1 strictly we should consider T restricted to the first k_1 requests.) Since T_B and T_C are not connected (using Lemma 2) tour T must be connected with either T_B or T_C . With (3) this implies that either any such tour T is connected with T_B or any such tour is connected with T_C . Hence the optimal tour P^* , and the tour T_D defined by ONLINE, are both connected with T_B or are both connected with T_C . This implies $\delta(v_D, v^*) \leq |T_D| + \max\{|T_B|, |T_C|\} + |P^*| < 2\eta S + |P^*|$. With the triangle inequality we obtain

$$\begin{aligned} & \delta(v_{-1}, v_{-1}^*) - \delta(v, v^*) \\ & \geq \delta(v_{-1}, v^*) - \delta(v, v^*) - \delta(v_{-1}^*, v^*) \\ & \geq (\delta(v_{-1}, v_D) - \delta(v, v_D)) - 2\delta(v_D, v^*) - |P^*| \\ & > S/3 - 2(2\eta S + |P^*|) - |P^*| \\ & = 2S/5 - 3|P^*| \end{aligned}$$

Inequality (1) becomes $F > c_1|P^*| + c_2(2S/5 - 3|P^*|) > c_2 \cdot 2S/5 > 382S > 2|P|$.

It is clear that the inequality of the lemma also holds for the last phase of ONLINE, phase N , in case this phase finishes with Step (3). That the inequality also holds if this phase finishes in one of the two other steps is a matter of case checking, which we omit in this extended abstract. If the phase finishes in one of the two other steps, then a much better inequality can be obtained by going through the analysis above. We omit this in this extended abstract. $\square$

Constant competitiveness of ONLINE is now an easy consequence. We use P_i and P_i^* for the ONLINE and the optimal tour in phase i , respectively.

Theorem 1. *ONLINE is 100.000-competitive for the general two server problem.*

Proof. Consider a phase j of type II. Since ONLINE ends the phase in the same position as it started, any increase in potential is caused by the change of positions of the servers on the optimal tour only. Hence, $10^5 \cdot |P_j^*| - \Phi_j + \Phi_{j-1} \geq$

$10^5 \cdot |P_j^*| - 10^3 |P_j^*| \geq 0$. On the other hand, any phase j of type II is followed by a phase $j + 1$ in which the cost of the first step, is at least twice the total cost $|P_j|$ of phase j . The last phase is of type III implying that the ONLINE cost over all phases of type II is at most $\frac{1}{2} + \frac{1}{4} + \dots = 1$ times the cost over all phases of type III. We conclude that

$$\begin{aligned} \sum_{j \in \text{I} \cup \text{III}} |P_j| &\leq 2 \sum_{j \in \text{III}} |P_j| < \sum_{j \in \text{III}} 10^5 \cdot |P_j^*| - \Phi_j + \Phi_{j-1} \leq \\ \sum_{j \in \text{I} \cup \text{III}} 10^5 \cdot |P_j^*| - \Phi_j + \Phi_{j-1} &= \sum_{j \in \text{I} \cup \text{III}} 10^5 \cdot |P_j^*|. \end{aligned}$$

□

References

1. Allan Borodin, Nathan Linial, and Michael Saks, *An optimal online algorithm for metrical task system*, Journal of the ACM **39** (1992), 745–763.
2. Marek Chrobak, Howard Karloff, Tom H. Payne, and Sundar Vishwanathan, *New results on server problems*, SIAM Journal on Discrete Mathematics **4** (1991), 172–181.
3. Marek Chrobak and Lawrence L. Larmore, *An optimal online algorithm for k servers on trees*, SIAM Journal on Computing **20** (1991), 144–148.
4. Marek Chrobak and Jiří Sgall, *The weighted 2-server problem*, The 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS), LNCS 1770, Springer-Verlag, 2000, pp. 593–604.
5. Esteban Feuerstein, Steve Seiden, and Alejandro Strejlevich de Loma, *The related server problem*, Unpublished manuscript, 1999.
6. Amos Fiat and Moty Ricklin, *Competitive algorithms for the weighted server problem*, Theoretical Computer Science **130** (1994), 85–99.
7. Elias Koutsoupias and Christos Papadimitriou, *On the k -server conjecture*, Journal of the ACM **42** (1995), 971–983.
8. Elias Koutsoupias and David Taylor, *The CNN problem and other k -server variants*, The 17th Annual Symposium on Theoretical Aspects of Computer Science 2000 (STACS), LNCS 1770, Springer-Verlag, 2000, pp. 581–592.
9. Mark Manasse, Lyle A. McGeoch, and Daniel Sleator, *Competitive algorithms for server problems*, Journal of Algorithms **11** (1990), 208–230.

On the Competitive Ratio for Online Facility Location^{*}

Dimitris Fotakis

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
fotakis@mpi-sb.mpg.de

Abstract. We consider the problem of Online Facility Location, where demands arrive online and must be irrevocably assigned to an open facility upon arrival. The objective is to minimize the sum of facility and assignment costs. We prove that the competitive ratio for Online Facility Location is $\Theta(\frac{\log n}{\log \log n})$. On the negative side, we show that no randomized algorithm can achieve a competitive ratio better than $\Omega(\frac{\log n}{\log \log n})$ against an oblivious adversary even if the demands lie on a line segment. On the positive side, we present a deterministic algorithm achieving a competitive ratio of $O(\frac{\log n}{\log \log n})$. The analysis is based on a hierarchical decomposition of the optimal facility locations such that each component either is relatively well-separated or has a relatively large diameter, and a potential function argument which distinguishes between the two kinds of components.

1 Introduction

The (metric uncapacitated) Facility Location problem is, given a metric space along with a facility cost for each point and a (multi)set of demand points, to find a set of facility locations which minimize the sum of facility and assignment costs. The assignment cost of a demand point is the distance to the nearest facility. Facility Location provides a simple and natural model for network design and clustering problems and has been the subject of intensive research over the last decade (e.g., see [17] for a survey and [9] for approximation algorithms and applications).

The definition of *Online Facility Location* [16] is motivated by practical applications where either the demand set is not known in advance or the solution must be constructed incrementally using limited information about future demands. In Online Facility Location, the demands arrive one at a time and must be irrevocably assigned to an open facility without any knowledge about future demands. The objective is to minimize the sum of facility and assignment costs, where each demand's assignment cost is the distance to the facility it is assigned to.

^{*} This work was partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

We evaluate the performance of online algorithms using *competitive analysis* (e.g., [5]). An online algorithm is c -competitive if for all instances, the cost incurred by the algorithm is at most c times the cost incurred by an optimal offline algorithm, which has full knowledge of the demand sequence, on the same instance. We always use n to denote the number of demands.

Previous Work. In the offline case, where the demand set is fully known in advance, there are constant factor approximation algorithms based on Linear Programming rounding (e.g., [18]), local search (e.g., [10]), and the primal-dual schema (e.g., [12]). The best known polynomial-time algorithm achieves an approximation ratio of 1.52 [14], while no polynomial-time algorithm can achieve an approximation ratio less than 1.463 unless $\text{NP} = \text{DTIME}(n^{O(\log \log n)})$ [10].

Online Facility Location was first defined and studied in [16], where a simple randomized algorithm is shown to achieve a constant performance ratio if the demands, which are adversarially selected, arrive in random order. In the standard framework of competitive analysis, where not only the demand set but also the demand order is selected by an oblivious adversary, the same algorithm achieves a competitive ratio of $O(\frac{\log n}{\log \log n})^1$. It is also shown a lower bound of $\Omega(\log^* n)$ on the competitive ratio of any online algorithm, where $\log^*$ is the inverse Ackerman function.

Online Facility Location should not be confused with the problem of Online Median [15]. In Online Median, the demand set is fully known in advance and the number of facilities increases online. An $O(1)$ -competitive algorithm is known for Online Median [15].

Online Facility Location bears a resemblance to the extensively studied problem of Online File Replication (e.g., [4,2,1,13,8]). In Online File Replication, we are given a metric space, a point initially holding the file, and a replication cost factor. Read requests are generated by points in an online fashion. Each request accesses the nearest file copy at a cost equal to the corresponding distance. In between requests, the file may be replicated to a set of points at a cost equal to the replication cost factor times the total length of the minimum Steiner tree connecting the set of points receiving the file to at least one point already holding the file. Similarly to Facility Location, File Replication asks for a set of file locations which minimize the sum of replication and access costs. The important difference is that the cost of each facility only depends on the location, while the cost of each replication depends on the set of points which hold the file and the set of points which receive the file.

Online File Replication is a generalization of Online Steiner Tree [11]. Hence, there are metric spaces in which no randomized online algorithm can achieve a competitive ratio better than $\Omega(\log n)$ against an oblivious adversary. They are known both a randomized [4] and a deterministic [2] algorithm achieving a competitive ratio of $O(\log n)$ for the more general problem of Online File Allocation. For trees and rings, algorithms of constant competitive ratio are known [1,13,8].

¹ Only a logarithmic competitive ratio is claimed in [16]. However, a competitive ratio of $O(\frac{\log n}{\log \log n})$ follows from a simple modification of the same argument.

Contribution. We prove that the competitive ratio for Online Facility Location is $\Theta(\frac{\log n}{\log \log n})$. On the negative side, we show that no randomized algorithm can achieve a competitive ratio better than $\Omega(\frac{\log n}{\log \log n})$ against an oblivious adversary even if the metric space is a line segment. The only previously known lower bound was $\Omega(\log^* n)$ [16]. On the positive side, we present a deterministic algorithm achieving a competitive ratio of $O(\frac{\log n}{\log \log n})$ in every metric space. To the best of our knowledge, this is the first deterministic upper bound on the competitive ratio for Online Facility Location.

As for the analysis, the technique of [2], which is based on a hierarchical decomposition/cover of the optimal file locations such that each component's diameter is not too large, cannot be adapted to yield a sub-logarithmic competitive ratio for Online Facility Location. On the other hand, it is not difficult to show that our algorithm achieves a competitive ratio of $O(\frac{\log n}{\log \log n})$ for instances whose optimal solution consists of a single facility. To establish a tight bound for general instances, we show that any metric space has a hierarchical cover with the additional property that any component either is relatively well-separated or has a relatively large diameter. Then, we prove that the sub-instances corresponding to well-separated components can be treated as essentially independent instances whose optimal solutions consist of a single facility, and we bound the additional cost incurred by the algorithm because of the sub-instances corresponding to large diameter components.

Problem Definition. The problem of Online Facility Location is formally defined as follows. We are given a metric space $\mathcal{M} = (C, d)$, where C denotes the set of points and $d : C \times C \mapsto \mathbb{R}^+$ denotes the distance function which is symmetric and satisfies the triangle inequality. For each point $v \in C$, we are also given the cost f_v of opening a facility at v . The demand sequence consists of (not necessarily distinct) points $w \in C$. When a demand w arrives, the algorithm can open some new facilities. Once opened, a facility cannot be closed. Then, w must be irrevocably assigned to the nearest facility. If w is assigned to a facility at v , w 's assignment cost is $d(w, v)$. The objective is to minimize the sum of facility and assignment costs.

Throughout this paper, we only consider unit demands by allowing multiple demands to be located at the same point. We always use n to denote the total number of demands. We distinguish between the case of uniform facility costs, where the cost of opening a facility, denoted by f , is the same for all points, and the general case of non-uniform facility costs, where the cost of opening a facility depends on the point.

Notation. A metric space $\mathcal{M} = (C, d)$ is usually identified by its point set C . For a subspace $C' \subseteq C$, $D(C') = \max_{u, v \in C'} \{d(u, v)\}$ denotes the diameter of C' . For a point $u \in C$ and a subspace $C' \subseteq C$, $d(C', u) = \min_{v \in C'} \{d(v, u)\}$ denotes the distance from u to the nearest point in C' . We use the convention that $d(u, \emptyset) = \infty$. For subspaces $C', C'' \subseteq C$, $d(C', C'') = \min_{u \in C''} \{d(C', u)\}$ denotes the

minimum distance between a point in C' and a point in C'' . For a point $u \in C$ and a non-negative number r , $B(u, r)$ denotes the ball of center u and radius r , $B(u, r) = \{v \in C : d(u, v) \leq r\}$.

2 A Lower Bound on the Competitive Ratio

In this section, we restrict our attention to uniform facility costs and instances whose optimal solution consists of a single facility. These assumptions can only strengthen the proven lower bound.

Theorem 1. *No randomized algorithm for Online Facility Location can achieve a competitive ratio better than $\Omega(\frac{\log n}{\log \log n})$ against an oblivious adversary even if the metric space is a line segment.*

Proof Sketch. We first prove that the lower bound holds if the metric space is a complete binary Hierarchically Well-Separated Tree (HST) [3]. Let T be a complete binary rooted tree of height h such that (i) the distance from the root to each of its children is D , and (ii) on every path from the root to a leaf, the edge length drops by a factor exactly m on every step. The height of a vertex is the number of edges on the path to the root. Every non-leaf vertex has exactly two children and every leaf has height exactly h . The distance from a vertex of height i to each of its children is exactly $\frac{D}{m^i}$. Let f be the cost of opening a new facility, which is the same for every vertex of T .

For a vertex v , let T_v denote the subtree rooted at v . The lower bound is based on the following property of T : The distance from a vertex v of height i to any vertex in T_v is at most $\frac{m}{m-1} \frac{D}{m^i}$, while the distance from v to any vertex not in T_v is at least $\frac{D}{m^{i+1}}$.

By Yao's principle (e.g., [5, Chapter 8]), it suffices to show that there is a probability distribution over demand sequences for which the ratio of the expected cost of any deterministic online algorithm to the expected optimal cost is $\Omega(\frac{\log n}{\log \log n})$.

We define an appropriate probability distribution by considering demand sequences divided into $h+1$ phases. Phase 0 consists of a single demand at the root v_0 . After the end of phase i , if v_i is not a leaf, the adversary proceeds to the next phase by selecting v_{i+1} uniformly at random and independently (u.i.r.) among the two children of v_i . Phase $i+1$ consists of m^{i+1} consecutive demands at v_{i+1} .

The total number of demands is at most $m^h \frac{m}{m-1}$, which must not exceed n . The optimal solution opens a single facility at v_h and, for each phase i , incurs an assignment cost no greater than $D \frac{m}{m-1}$. Therefore, the optimal cost is at most $f + hD \frac{m}{m-1}$.

Let Alg be any deterministic online algorithm. We fix the adversary's random choices $v_0, \dots, v_i$ up to phase i , $0 \leq i \leq h-1$, (equivalently, we fix T_{v_i}), and we consider the expected cost (conditional on T_{v_i}) incurred by Alg for demands and facilities not in T_{v_i} . If Alg has no facilities in T_{v_i} when the first demand at v_{i+1} arrives, the assignment cost of demands at $v_i \in T_{v_i} \setminus T_{v_{i+1}}$ is at least

```

 $A \leftarrow \emptyset; L \leftarrow \emptyset; /* \text{Initialization} */$ 
For each demand  $w$ :
   $r_w \leftarrow \frac{d(A, w)}{x}; B_w \leftarrow \{u \in L \cup \{w\} : d(w, u) \leq r_w\}; \text{Pot}(B_w) \leftarrow \sum_{u \in B_w} d(A, u);$ 
  if  $\text{Pot}(B_w) \geq f$  then    /* A new facility is opened */
    if  $d(A, w) < f$  then
      Let  $\nu$  be the smallest integer:
        either there exists exactly one  $u \in B_w$  such that
           $\text{Pot}(B_w \cap B(u, \frac{r_w}{2^\nu})) > \frac{\text{Pot}(B_w)}{2},$ 
        or, for any  $u \in B_w, \text{Pot}(B_w \cap B(u, \frac{r_w}{2^{\nu+1}})) \leq \frac{\text{Pot}(B_w)}{2}.$ 
      Let  $\hat{w}$  be any demand in  $B_w: \text{Pot}(B_w \cap B(\hat{w}, \frac{r_w}{2^\nu})) > \frac{\text{Pot}(B_w)}{2}.$ 
      else  $\hat{w} \leftarrow w;$ 
       $A \leftarrow A \cup \{\hat{w}\}; L \leftarrow L \setminus B_w;$ 
    else  $L \leftarrow L \cup \{w\};$     /*  $w$  is marked unsatisfied */
  Assign  $w$  to the nearest facility in  $A.$ 

```

Fig. 1. The algorithm Deterministic Facility Location – DFL.

mD . Otherwise, since v_{i+1} is selected u.i.r. among v_i 's children, with probability at least $\frac{1}{2}$, there is at least one facility in $T_{v_i} \setminus T_{v_{i+1}}$. Therefore, for any fixed T_{v_i} , the (conditional) expected cost incurred by Alg for demands and facilities not in $T_{v_{i+1}}$ is at least $\min\{mD, \frac{f}{2}\}$ plus the cost for demands and facilities not in T_{v_i} . Since this holds for any fixed choice of $v_0, \dots, v_i$ (equivalently, for any fixed T_{v_i}), the (unconditional) expected cost incurred by Alg for demands and facilities not in $T_{v_{i+1}}$ is at least $\min\{mD, \frac{f}{2}\}$ plus the (unconditional) expected cost for demands and facilities not in T_{v_i} . Hence, at the beginning of phase i , $0 \leq i \leq h$, the expected cost incurred by Alg for demands and facilities not in T_{v_i} is at least $i \min\{mD, \frac{f}{2}\}$. For the last phase, Alg incurs a cost no less than $\min\{mD, f\}$ inside T_{v_h} .

For $m = h$ and $D = \frac{f}{h}$, the total expected cost of Alg is at least $\frac{h+2}{2}hD$, while the optimal cost is at most $\frac{2h-1}{h-1}hD$. For the chosen value of h , the quantity $\frac{h^{h+1}}{h-1}$ must not exceed n . Setting $h = \left\lfloor \frac{\log n}{\log \log n} \right\rfloor$ yields the claimed lower bound.

To conclude the proof, we consider the following embedding of T in a line segment. The root is mapped to 0 (i.e., the center of the segment). Let v be a vertex of height i mapped to $\tilde{v}$. Then, v 's left child is mapped to $\tilde{v} - \frac{D}{m^i}$ and v 's right child is mapped to $\tilde{v} + \frac{D}{m^i}$. It can be shown that, for any $m \geq 4$, this embedding results in a hierarchically well-separated metric space. $\square$

3 A Deterministic Algorithm for Uniform Facility Costs

In this section, we present the algorithm Deterministic Facility Location – DFL (Fig. 1) and prove that its competitive ratio is $O(\frac{\log n}{\log \log n})$.

Outline. The algorithm maintains its *facility configuration* A and the set L of *unsatisfied demands*, which are the demands not having contributed towards opening a new facility so far. A new demand w is marked unsatisfied and added

to L only if no new facilities are opened when w arrives. Each unsatisfied demand $u \in L$ can contribute an amount of $d(A, u)$ to the cost of opening a new facility in its neighborhood. We refer to the quantity $d(A, u)$ as the *potential* of u . Only unsatisfied demands and the demand currently being processed have non-zero potential. For a set S consisting of demands of non-zero potential, let $\text{Pot}(S) = \sum_{u \in S} d(A, u)$ be the potential of S .

The high level idea is to keep a balance between the algorithm's assignment and facility costs. For each demand w , the algorithm computes the set B_w consisting of w and the unsatisfied demands at distance no greater than $\frac{d(A, w)}{x}$ from w , where x is a sufficiently large constant. If B_w 's potential is less than f , w is assigned to the nearest facility, marked unsatisfied and added to L . Otherwise, the algorithm opens a new facility at an appropriate location $\hat{w} \in B_w$ and assigns w to it. In this case, the demands in B_w are marked satisfied and removed from L . The location $\hat{w}$ is chosen as the center of a smallest radius ball/subset of B_w contributing more than half of B_w 's potential.

An Overview of the Analysis. For an arbitrary sequence of n demands, we compare the algorithm's cost with the cost of a fixed offline optimal solution. The optimal solution is determined by k facility locations $c_1^*, c_2^*, \dots, c_k^*$. The set of optimal facilities is denoted by C^* . Each demand u is assigned to the nearest facility in C^* . Hence, C^* defines a partition of the demand sequence into optimal clusters $C_1, C_2, \dots, C_k$. Let $d_u^* = d(C^*, u)$ denote the assignment cost of u in the optimal solution, let $S^* = \sum_u d_u^*$ be the total optimal assignment cost, let $F^* = kf$ be the total optimal facility cost, and let $\sigma^* = \frac{S^*}{n}$ be the average optimal assignment cost.

Let ρ, ψ denote a fixed pair of integers such that $\rho^\psi > n$. For any integer j , $0 \leq j \leq \psi$, let $r(j) = \rho^j \sigma^*$. We also define $r(-1) = 0$ and $r(\psi + 1) = \infty$. We observe that, for any demand u , $d_u^* < r(\psi)$. Let λ be some appropriately large constant, and, for any integer j , $-1 \leq j \leq \psi + 1$, let $R(j) = \lambda r(j)$. Throughout the analysis of DFL, we use $\lambda = 3x + 2$.

The Case of a Single Optimal Cluster. We first restrict our attention to instances whose optimal solution consists of a single facility c^* . The convergence of A to c^* is divided into $\psi + 2$ phases, where the current phase ℓ , $-1 \leq \ell \leq \psi$, starts just after the first facility within a distance of $R(\ell + 1)$ from c^* is opened and ends when the first facility within a distance of $R(\ell)$ from c^* is opened. In other words, the current phase ℓ lasts as long as $d(A, c^*) \in (R(\ell), R(\ell + 1)]$.

The demands arriving in the current phase ℓ and the demands remaining in L from the previous phase are partitioned into *inner* demands, whose optimal assignment cost is less than $r(\ell)$, and *outer* demands. The last phase ($\ell = -1$) never ends and only consists of outer demands.

For any outer demand u , $d(A, u)$ is at most $\lambda \sigma^* + (\lambda \rho + 1)d_u^*$ (Ineq. (3)). Hence, the assignment cost of an outer demand arriving in phase ℓ can be charged to its optimal assignment cost. We charge the total assignment cost of inner demands arriving in phase ℓ and the total facility cost incurred by the algorithm in phase ℓ to the optimal facility cost and the optimal assignment cost of the outer demands marked satisfied in phase ℓ .

The set of inner demands is included in a ball of center c^* and radius $r(\ell)$. If $R(\ell)$ is large enough compared to $r(\ell)$ (namely, if λ is chosen sufficiently large), we can think of the inner demands as being essentially located at c^* , because they are much closer to each other than to the current facility configuration A . Hence, we refer to the total potential accumulated by unsatisfied inner demands as the potential accumulated by c^* or simply, the potential of c^* . For any inner demand w , B_w includes the entire set of unsatisfied inner demands. Therefore, the potential accumulated by c^* is always less than f (Lemma 3).

However, a new facility may decrease the potential of c^* , because (i) it may be closer to c^* , and (ii) some unsatisfied inner demands may contribute their potential towards opening the new facility, in which case they are marked satisfied and removed from L . As a result, the upper bound of f on the potential accumulated by c^* cannot be directly translated into an upper bound on the total assignment cost of the inner demands arriving in phase ℓ as in [16].

Each time a new facility is opened, the algorithm incurs a facility cost of f and an assignment cost no greater than $\frac{f}{x}$. The algorithm must also be charged with an additional cost accounting for the decrease in the potential accumulated by c^* , which cannot exceed f . Hence, for each new facility, the algorithm is charged with a cost no greater than $\frac{2x+1}{x}f$.

Using the fact that $R(\ell)$ is much larger than $r(\ell)$, we show that if the inner demands included in B_w contribute more than half of B_w 's potential, the new facility at $\hat{w}$ is within a distance of $R(\ell)$ from c^* (Lemma 4). In this case (Lemma 8, Case Isolated.B), the current phase ends and the algorithm's cost is charged to the optimal facility cost. Otherwise (Lemma 8, Case Isolated.A), the algorithm's cost is charged to the potential of the outer demands included in B_w , which is at least $f/2$. The optimal facility cost is charged $O(\psi)$ times and the optimal assignment cost is charged $O(\lambda\rho)$ times. Hence, setting $\psi = \rho = O(\frac{\log n}{\log \log n})$ yields the desired competitive ratio.

The General Case. If the optimal solution consists of $k > 1$ facilities $c_1^*, \dots, c_k^*$, the demands are partitioned into the optimal clusters $C_1, \dots, C_k$. The convergence of A to an optimal facility c_i^* is divided into $\psi + 2$ phases, where the current phase ℓ_i , $-1 \leq \ell_i \leq \psi$, lasts as long as $d(A, c_i^*) \in (R(\ell_i), R(\ell_i + 1)]$. For the current phase ℓ_i , the demands of C_i are again partitioned into inner and outer demands, and the inner demands of C_i can be thought of as being essentially located at c_i^* .

As before, the potential accumulated by an optimal facility c_i^* cannot exceed f . However, a single new facility can decrease the potential accumulated by many optimal facilities. Therefore, if we bound the decrease in the potential of each optimal facility separately and charge the algorithm with the total additional cost, we can only guarantee a logarithmic upper bound on the competitive ratio. To establish a tight bound, we show that the average (per new facility) decrease in the total potential accumulated by optimal facilities is $O(f)$.

We first observe that as long as the distance from the algorithm's facility configuration A to a set of optimal facilities K is large enough compared to the diameter of K , the inner demands assigned to facilities in K are much closer

to each other than to A . Consequently, we can think of the inner demands assigned to K as being located at some optimal facility $c_K^* \in K$. Therefore, the total potential accumulated by optimal facilities in K is always less than f (Lemma 3). This observation naturally leads to the definition of an (optimal facility) *coalition* (Definition 2).

Our potential function argument is based on a *hierarchical cover* (Definition 1) of the subspace C^* comprising the optimal facility locations. Given a facility configuration A , the hierarchical cover determines a minimal collection of *active* coalitions which form a partition of C^* (Definition 3).

A coalition is *isolated* if it is well-separated from any other disjoint coalition, and *typical* otherwise. A new facility can decrease the potential accumulated by at most one isolated active coalition. Therefore, for each new facility, the decrease in the total potential accumulated by isolated active coalitions is at most f (Lemma 8, Case Isolated).

On the other hand, a new facility can decrease the potential accumulated by several typical active coalitions. We prove that any metric space has a hierarchical cover such that each component either is relatively well-separated or has a relatively large diameter (i.e., its diameter is within a constant factor from its parent's diameter (Lemma 1). Typical active coalitions correspond to the latter kind of components. Hence, we obtain a bound on the relative length of the interval for which an active coalition remains typical (Lemma 2), which can be translated into a bound of $O(f)$ on the total decrease in the potential accumulated by an active coalition, while the coalition remains typical (potential function component $\Xi_K^{(2)}$ and Lemma 7).

In the remaining paragraphs, we prove the following theorem by turning the aforementioned intuition into a formal potential function argument.

Theorem 2. *For any constant $x \geq 10$, the competitive ratio of Deterministic Facility Location is $O(\frac{\log n}{\log \log n})$.*

Hierarchical Covers and Optimal Facility Coalitions. We start by showing that any metric space has a hierarchical cover with the desired properties.

Definition 1. A hierarchical cover of a metric space C is a collection $\mathcal{K} = \{K_1, \dots, K_m\}$ of non-empty subsets of C which can be represented by a rooted tree $T_{\mathcal{K}}$ in the following sense:

- (A) C belongs to $\mathcal{K}$ and corresponds to the root of $T_{\mathcal{K}}$.
- (B) For any $K \in \mathcal{K}$, $|K| > 1$, $\mathcal{K}$ contains sets $K_1, \dots, K_\mu$, each of diameter less than $D(K)$, which form a partition of K . The sets $K_1, \dots, K_\mu$ correspond to the children of K in $T_{\mathcal{K}}$.

We use $\mathcal{K}$ and its tree representation $T_{\mathcal{K}}$ interchangeably. By definition, every non-leaf set has at least two children. Therefore, $T_{\mathcal{K}}$ has at most $2|C| - 1$ nodes. For a set K different from the root, we use P_K to denote the immediate ancestor/parent of K in $T_{\mathcal{K}}$. Our potential function argument is based on the following property of metric spaces.

Lemma 1. *For any metric space C and any $\gamma \geq 16$, there exists a hierarchical cover T_K of C such that for any set K different from the root, either $D(K) > \frac{D(P_K)}{\gamma^2}$ or $d(K, C \setminus K) > \frac{D(P_K)}{4\gamma}$.*

Proof Sketch. Let C be any metric space, and let $D = D(C)$. We first show that, for any integer $i \geq 0$, C can be partitioned into a collection of level i groups $G_1^i, \dots, G_m^i$ such that (i) for any $j_1 \neq j_2$, $d(G_{j_1}^i, G_{j_2}^i) > \frac{D}{4\gamma^i}$, and (ii) if $D(G_j^i) > \frac{D}{\gamma^i}$, then G_j^i does not contain any subset $G \subseteq G_j^i$ such that both $D(G) \leq \frac{D}{\gamma^{i+1}}$ and $d(G, G_j^i \setminus G) > \frac{D}{4\gamma^i}$. Since the collection of level i groups is a partition of C , for any G_j^i , $d(G_j^i, C \setminus G_j^i) > \frac{D}{4\gamma^i}$.

Level i groups are further partitioned into level i components $K_1^i, \dots, K_{m'}^i$ such that (i) $D(K_j^i) \leq \frac{D}{\gamma^i}$, and (ii) either $D(K_j^i) > \frac{D}{\gamma^{i+1}}$ or $d(K_j^i, C \setminus K_j^i) > \frac{D}{4\gamma^i}$. To ensure a hierarchical structure, we proceed inductively in a bottom-up fashion. We create a single level i component for each level i group G_j^i of diameter no greater than $\frac{D}{\gamma^i}$. We recall that $d(G_j^i, C \setminus G_j^i) > \frac{D}{4\gamma^i}$. If $D(G_j^i) > \frac{D}{\gamma^i}$, G_j^i is partitioned into level i components of diameter in the interval $(\frac{D}{\gamma^{i+1}}, \frac{D}{\gamma^i}]$. For $\gamma \geq 16$, such a partition exists, because G_j^i does not contain any well-separated subsets of small diameter. Finally, we eliminate multiple occurrences of the same component at different levels. $\square$

Definition 2. *A set of optimal facilities $K \subseteq C^*$ with representative $c_K^* \in K$ is a coalition with respect to the facility configuration A if $d(A, c_K^*) \geq \lambda D(K)$. A coalition K is called isolated if $d(K, C^* \setminus K) \geq 2d(A, c_K^*)$, and typical otherwise. A coalition K becomes broken as soon as $d(A, c_K^*) < \lambda D(K)$.*

Given a hierarchical cover T_K of the subspace C^* comprising the optimal facility locations, we choose an arbitrary optimal facility as the representative of each set K . The representative of K always remains the same and is denoted by c_K^* . Then, T_K can be regarded as a system of optimal facility coalitions which hierarchically covers C^* . The current facility configuration A defines a minimal collection of active coalitions which form a partition of C^* .

Definition 3. *Given a hierarchical cover T_K of C^* , a coalition $K \in T_K$ is an active coalition with respect to A if $d(A, c_K^*) \geq \lambda D(K)$ and for any other coalition K' on the path from K to the root of T_K , $d(A, c_{K'}^*) < \lambda D(K')$.*

Lemma 2. *For any $\gamma \geq 8\lambda$, there is a hierarchical cover T_K of C^* such that if K is a typical active coalition with respect to the facility configuration A , then $\lambda \frac{D(P_K)}{\gamma^2} < d(A, c_K^*) < (\lambda + 1)D(P_K)$.*

Proof. For some $\gamma \geq 8\lambda$, let T_K be the hierarchical cover of C^* implied by Lemma 1. We show that T_K has the claimed property. The root of T_K is an isolated coalition by definition. Hence, we can restrict our attention to coalitions $K \in T_K$ different from the root for which the parent function P_K is well-defined.

Since K is an active coalition, its parent coalition P_K must have become broken. The upper bound on $d(A, c_K^*)$ follows from the triangle inequality and the fact that c_K^* also belongs to P_K .

For the lower bound, we consider two cases. If K has a relatively large diameter ($D(K) > \frac{D(P_K)}{\gamma^2}$), the lower bound on $d(A, c_K^*)$ holds as long as K remains a coalition. If K is relatively well-separated ($d(K, C^* \setminus K) > \frac{D(P_K)}{4\gamma}$) and the lower bound on $d(A, c_K^*)$ does not hold, we conclude that $2d(A, c_K^*) < d(K, C^* \setminus K)$ (K is an isolated coalition), which is a contradiction. $\square$

Notation. The set of active coalitions with respect to the current facility configuration A is denoted by $\text{Act}(A)$. For a coalition K , ℓ_K denotes the index of the current phase. Namely, ℓ_K is equal to the integer j , $-1 \leq j \leq \psi$, such that $d(A, c_K^*) \in (R(j), R(j+1)]$. If $d(A, c_K^*) > R(\psi)$, $\ell_K = \psi$ (the first phase), while if $d(A, c_K^*) \leq R(0)$, $\ell_K = -1$ (the last phase). Let $C_K = \bigcup_{c_i^* \in K} C_i$ be the optimal cluster corresponding to K . Since $\text{Act}(A)$ is always a partition of C^* , the collection $\{C_K : K \in \text{Act}(A)\}$ is a partition of the demand sequence. For the current phase ℓ_K , the demands of C_K are partitioned into inner demands $\text{In}(K) = \{u \in C_K : d_u^* < r(\ell_K)\}$ and outer demands $\text{Out}(K) = C_K \setminus \text{In}(K)$. Let also $\Lambda_K = L \cap \text{In}(K)$ be the set of unsatisfied inner demands assigned to K .

We should emphasize that ℓ_K , $\text{In}(K)$, $\text{Out}(K)$, and Λ_K depend on the current facility configuration A . In addition, Λ_K depends on the current set of unsatisfied demands L . For simplicity of notation, we omit the explicit dependence on A and L by assuming that while a demand w is being processed, ℓ_K , $\text{In}(K)$, $\text{Out}(K)$, and Λ_K keep the values they had when w arrived.

Properties. Let K be a coalition with respect to the current facility configuration A . Then, $d(A, c_K^*) \geq \lambda \max\{D(K), r(\ell_K)\}$. The diameter of the subspace comprising the inner demands of K is $D(\text{In}(K)) < 3 \max\{D(K), r(\ell_K)\}$. We repeatedly use the following inequalities. Let u be any demand in C_K and let $c_u^* \in K$ be the optimal facility to which u is assigned. Then,

$$d(A, u) \leq d(A, c_K^*) + d(c_K^*, c_u^*) + d(c_u^*, u) \leq d(A, c_K^*) + D(K) + d_u^* \leq \frac{\lambda+1}{\lambda} d(A, c_K^*) + d_u^* \quad (1)$$

If u is an inner demand of K ($u \in \text{In}(K)$),

$$d(u, c_K^*) \leq d(u, c_u^*) + d(c_u^*, c_K^*) < r(\ell_K) + D(K) \leq 2 \max\{D(K), r(\ell_K)\} \leq \frac{2}{\lambda} d(A, c_K^*) \quad (2)$$

If u is an outer demand of K ($u \in \text{Out}(K)$),

$$d(A, u) \leq (\lambda + 1)\sigma^* + ((\lambda + 1)\rho + 1)d_u^* \quad (3)$$

Proof of Ineq. (3). Since u is an outer demand, it must be the case that $d_u^* \geq r(\ell_K)$. In addition, by Ineq. (1), $d(A, u) \leq \frac{\lambda+1}{\lambda} d(A, c_K^*) + d_u^*$. If the current phase is the last one ($\ell_K = -1$), then $d(A, c_K^*) \leq \lambda\sigma^*$, and the inequality follows. Otherwise, the current phase cannot be the first one (i.e., it must be $\ell_K < \psi$), because $d_u^* < r(\psi)$ and u could not be an outer demand. Therefore, $d(A, u) \leq R(\ell_K + 1) = \lambda\rho r(\ell_K) \leq \lambda\rho d_u^*$, and the inequality follows. $\square$

Lemma 3 and Lemma 4 establish the main properties of DFL.

Lemma 3. For any coalition K , $\text{Pot}(\Lambda_K) = \sum_{u \in \Lambda_K} d(A, u) < f$.

Proof. In the last phase ($\ell_K = -1$), $\text{Pot}(\Lambda_K) = 0$, because there are no inner demands ($\ln(K) = \emptyset$). If $\ell_K \geq 0$, for any inner demand u of K ($u \in \ln(K)$),

$$d(A, u) \geq d(A, c_K^*) - d(c_K^*, u) > 3x \max\{D(K), r(\ell_K)\} ,$$

where the last inequality follows from (i) $d(A, c_K^*) \geq \lambda \max\{D(K), r(\ell_K)\}$, because K is a coalition, (ii) $d(u, c_K^*) < 2 \max\{D(K), r(\ell_K)\}$, because of Ineq. (2), and (iii) $\lambda = 3x + 2$.

Let w be the demand in Λ_K which has arrived last, and let A_w be the facility configuration when w arrived. The last time $\text{Pot}(\Lambda_K)$ increased was when w was added to L (and hence, to Λ_K). Since $D(\ln(K)) < 3 \max\{D(K), r(\ell_K)\} < \frac{d(A, w)}{x} \leq \frac{d(A_w, w)}{x}$, B_w must have contained the entire set Λ_K (including w). $\text{Pot}(B_w)$ must have been less than f , because w was added to L . Therefore, $\text{Pot}(\Lambda_K) \leq \text{Pot}(B_w) < f$. $\square$

Lemma 4. Let w be any demand such that $\text{Pot}(B_w) \geq f$, and, for a coalition K , let $\Lambda_K^w = B_w \cap \ln(K)$. If there exists an active coalition K such that $\text{Pot}(\Lambda_K^w) > \frac{\text{Pot}(B_w)}{2}$, then $d(\hat{w}, c_K^*) < 8 \max\{D(K), r(\ell_K)\}$.

Proof. We first consider the case that $d(A, w) \geq f$ and $\hat{w}$ coincides with w . If there exists an active coalition K such that $w \in \ln(K)$, the conclusion of the lemma follows from Ineq. (2). For any active coalition K' such that $w \notin \ln(K')$, Lemma 3 implies that $\text{Pot}(\Lambda_{K'}^w) < \frac{\text{Pot}(B_w)}{2}$, because $\text{Pot}(B_w \setminus \Lambda_{K'}^w) \geq d(A, w) \geq f$.

We have also to consider the case that $d(A, w) < f$. We observe that any subset of B_w including a potential greater than $\frac{\text{Pot}(B_w)}{2}$ must have a non-empty intersection with Λ_K^w . If $\frac{r_w}{2^\nu} < 6 \max\{D(K), r(\ell_K)\}$, let u be any demand in $\Lambda_K^w \cap B(\hat{w}, \frac{r_w}{2^\nu})$. Since u is an inner demand of K , using Ineq. (2), we show that

$$d(\hat{w}, c_K^*) \leq d(\hat{w}, u) + d(u, c_K^*) < 6 \max\{D(K), r(\ell_K)\} + 2 \max\{D(K), r(\ell_K)\} .$$

Otherwise, it must be $\frac{r_w}{2^{\nu+1}} \geq 3 \max\{D(K), r(\ell_K)\} > D(\ln(K))$. Therefore, for any $u \in \Lambda_K^w$, $B_w \cap B(u, \frac{r_w}{2^{\nu+1}})$ includes the entire set Λ_K^w and hence, a potential greater than $\frac{\text{Pot}(B_w)}{2}$. Consequently, there must be a single demand $u \in B_w$ such that $\text{Pot}(B_w \cap B(u, \frac{r_w}{2^\nu})) > \frac{\text{Pot}(B_w)}{2}$. Since the previous inequality is satisfied by any demand $u \in \Lambda_K^w$, there must be only one demand in Λ_K^w , and $\hat{w}$ must coincide with it. The lemma follows from Ineq. (2), because $\hat{w}$ is an inner demand of K . $\square$

Potential Function Argument. We use the potential function Φ to bound the total algorithm's cost. Let T_K be the hierarchical cover of C^* implied by Lemma 2.

$$\Phi = \sum_{K \in T_K} \Phi_K , \text{ where } \Phi_K = \frac{(2x+1)(\lambda+1)}{x(\lambda-2)} \Xi_K - \frac{\lambda+1}{\lambda} \Upsilon_K .$$

The function Ξ_K is the sum of three components, $\Xi_K = \Xi_K^{(1)} + \Xi_K^{(2)} + \Xi_K^{(3)}$, where

$$\begin{aligned}\Xi_K^{(1)} &= \sum_{j=0}^{\psi} \xi^{(1)}(K, j), \quad \xi^{(1)}(K, j) = \begin{cases} f & \text{if } d(A, c_K^*) > R(j). \\ 0 & \text{if } d(A, c_K^*) \leq R(j). \end{cases} \\ \Xi_K^{(2)} &= \begin{cases} 0 & \text{if } K \text{ is the root of } T_K. \\ f \max \left\{ \ln \left(\frac{\min\{d(A, c_K^*), (\lambda+1)D(P_K)\}}{\lambda \frac{D(P_K)}{\gamma^2}} \right), 0 \right\} & \text{otherwise.} \end{cases} \\ \Xi_K^{(3)} &= \begin{cases} 2f & \text{if } K \text{ is a typical coalition.} \\ f & \text{if } K \text{ is an isolated coalition.} \\ 0 & \text{if } K \text{ has become broken.} \end{cases}\end{aligned}$$

The function Υ_K is defined as $\Upsilon_K = \begin{cases} \sum_{u \in \Lambda_K} d(A, c_K^*) & \text{if } K \in \text{Act}(A). \\ 0 & \text{otherwise.} \end{cases}$

Let K be an active coalition. The function $\Xi_K^{(1)}$ compensates for the cost of opening the facility concluding the current phase of K . $\Xi_K^{(2)}$ compensates for the additional cost charged to the algorithm while K is typical active coalition (Lemma 7). $\Xi_K^{(3)}$ compensates for the cost of opening a facility which changes the status of K either from typical to isolated or from isolated to broken. The function Ξ_K never increases and can decrease only if a new facility closer to c_K^* is opened. The function Υ_K is equal to the potential accumulated by c_K^* . Υ_K increases when an inner demand of K is added to L and decreases when a new facility closer to c_K^* is opened.

In the following, $\Delta\Phi$ denotes the change in the potential function because of a demand w . More specifically, let Φ be the value of the potential function just before the arrival of w , and let Φ' be the value of the potential function just after the algorithm has finished processing w . Then, $\Delta\Phi = \Phi' - \Phi$. The same notation is used with any of the potential function components above.

We first prove that Φ_K remains non-negative (Lemma 5). If a demand w is added to L (i.e., no new facilities are opened), the algorithm incurs an assignment cost of $d(A, w)$, while if w is not added to L (i.e., a new facility at $\hat{w}$ is opened), the algorithm incurs a facility cost of f and an assignment cost of $d(\hat{w}, w) < \frac{f}{x}$. In the former case, we show that $d(A, w) + \Delta\Phi \leq (\lambda+1)\sigma^* + ((\lambda+1)\rho+1)d_w^*$ (Lemma 6). In the latter case, we show that $f + d(\hat{w}, w) + \Delta\Phi \leq \frac{4(\lambda+1)}{\lambda-2} [(\lambda+1)\sigma^*|B_w| + ((\lambda+1)\rho+1) \sum_{u \in B_w} d_u^*]$ (Lemma 8).

Lemma 5. *For any coalition K , if $\ell_K \geq 0$, then $\Upsilon_K < \frac{\lambda}{\lambda-2}f$, while if $\ell_K = -1$, then $\Upsilon_K = 0$.*

Proof. In the last phase ($\ell_K = -1$), $\Upsilon_K = 0$ because there are no inner demands ($\text{In}(K) = \emptyset$). Otherwise, DFL maintains the invariant that $\text{Pot}(\Lambda_K) < f$ (Lemma 3). In addition, for any $u \in \Lambda_K$, $d(A, u) > \frac{\lambda-2}{\lambda}d(A, c_K^*)$, because of Ineq. (2). Therefore, $\Upsilon_K < \frac{\lambda}{\lambda-2}\text{Pot}(\Lambda_K) < \frac{\lambda}{\lambda-2}f$. $\square$

Lemma 5 implies that Φ_K is non-negative, because if K is an active coalition and $\ell_K \geq 0$, then $\frac{\lambda+1}{\lambda} \Upsilon_K < \frac{\lambda+1}{\lambda-2} f \leq \frac{(2x+1)(\lambda+1)}{x(\lambda-2)} \Xi_K^{(1)}$. On the other hand, if either K is not an active coalition or $\ell_K = -1$, then $\Upsilon_K = 0$.

Lemma 6. *If the demand w is added to L , then $d(A, w) + \Delta\Phi \leq (\lambda + 1)\sigma^* + ((\lambda + 1)\rho + 1)d_w^*$.*

Proof. Let K be the unique active coalition such that $w \in C_K$. If w is an inner demand of K , w is added to A_K , and $\Delta\Phi = -\frac{\lambda+1}{\lambda} \Delta\Upsilon_K = -\frac{\lambda+1}{\lambda} d(A, c_K^*)$. Using Ineq. (1), we conclude that $d(A, w) + \Delta\Phi \leq d_w^*$. If w is an outer demand of K , then $\Delta\Phi = 0$. Using Ineq. (3), we conclude that $d(A, w) + \Delta\Phi \leq (\lambda + 1)\sigma^* + ((\lambda + 1)\rho + 1)d_w^*$. $\square$

We have also to consider demands w which are not added to L (i.e., a new facility at $\hat{w}$ is opened). Let A be the facility configuration just before the arrival of w , and let $A' = A \cup \{\hat{w}\}$. We observe that if either K is not an active coalition or $\ell_K = -1$, $\Upsilon_K = 0$ and Φ_K cannot increase due to the new facility at $\hat{w}$. Therefore, we focus on active coalitions K such that $\ell_K \geq 0$.

Lemma 7. *Let $\hat{w}$ be the facility opened when the demand w arrives. Then, for any typical active coalition K , the quantity $\frac{(2x+1)(\lambda+1)}{x(\lambda-2)} \Xi_K - \frac{(2x+1)(\lambda+1)}{x\lambda} \Upsilon_K$ cannot increase due to $\hat{w}$.*

Proof. If either the current phase ends ($d(\hat{w}, c_K^*) \leq R(\ell_K)$) or K stops being a typical active coalition due to $\hat{w}$, then $\Delta\Xi_K \leq -f$, and the lemma follows from $-\Delta\Upsilon_K < \frac{\lambda}{\lambda-2} f$.

If K remains a typical active coalition with respect to A' and the current phase does not end ($d(\hat{w}, c_K^*) > R(\ell_K)$), let $\tau_K^w = \frac{d(A, c_K^*)}{d(A', c_K^*)} \geq 1$ be factor by which $d(A, c_K^*)$ decreases because of the new facility at $\hat{w}$. K cannot be the root of T_K , which is an isolated coalition by definition. Moreover, since K is a typical active coalition with respect to both A and A' , Lemma 2 implies that $(\lambda + 1)D(P_K) > d(A, c_K^*) \geq d(A', c_K^*) > \lambda \frac{D(P_K)}{\gamma^2}$. Therefore,

$$\Delta\Xi_K^{(2)} = \left[\ln \left(\frac{d(A', c_K^*)}{\lambda \frac{D(P_K)}{\gamma^2}} \right) - \ln \left(\frac{d(A, c_K^*)}{\lambda \frac{D(P_K)}{\gamma^2}} \right) \right] f = \ln \left(\frac{d(A', c_K^*)}{d(A, c_K^*)} \right) f = -\ln(\tau_K^w) f$$

If $B_w \cap \ln(K) = \emptyset$, no demands are removed from A_K , and $-\Delta\Upsilon_K \leq (1 - \frac{1}{\tau_K^w}) \Upsilon_K \leq \ln(\tau_K^w) \Upsilon_K$. Otherwise, we can show that $\tau_K^w > \frac{x}{3} > 3$, and $-\Delta\Upsilon_K \leq \Upsilon_K < \ln(\tau_K^w) \Upsilon_K$. In both cases, the lemma follows from $\Upsilon_K < \frac{\lambda}{\lambda-2} f$. $\square$

Lemma 8. *Let $\hat{w}$ be the facility opened when the demand w arrives. Then,*

$$f + d(\hat{w}, w) + \Delta\Phi \leq \frac{4(\lambda+1)}{\lambda-2} [(\lambda + 1)\sigma^* |B_w| + ((\lambda + 1)\rho + 1) \sum_{u \in B_w} d_u^*] .$$

Proof Sketch. Let A_w be the set of inner demands in B_w , and let $M_w = B_w \setminus A_w$ be the set of outer demands in B_w . We recall that $f + d(\hat{w}, w) \leq \frac{x+1}{x} f$.

Case Isolated. There exists an isolated active coalition K such that $d(\hat{w}, c_K^*) < d(A, c_K^*)$. Lemma 7 implies that for any typical active coalition K' , $\Delta\Phi_{K'} \leq 0$. In addition, for $x \geq 10$, we can prove that (i) for any isolated active coalition K' different from K , $d(\hat{w}, c_{K'}^*) \geq d(A, c_{K'}^*)$, and (ii) for any active coalition K' different from K , $B_w \cap \ln(K') = \emptyset$. As a result, for any isolated active coalition K' different from K , $\Delta\Phi_{K'} = 0$. In addition, only inner demands of K are included in B_w ($\Lambda_w \subseteq \ln(K)$).

We have also to bound $\frac{x+1}{x}f + \Delta\Phi_K$. Since $-\Delta\Upsilon_K < \frac{\lambda}{\lambda-2}f$ and $\lambda = 3x + 2$, $\frac{x+1}{x}f + \Delta\Phi_K < \frac{2(\lambda+1)}{\lambda-2}f + \Delta\Xi_K$. We distinguish between two cases depending on the potential contributed by Λ_w .

Case Isolated.A. $\text{Pot}(\Lambda_w) \leq \frac{\text{Pot}(B_w)}{2}$. Then, $\frac{2(\lambda+1)}{\lambda-2}f$ cannot exceed $\frac{4(\lambda+1)}{\lambda-2} \text{Pot}(M_w)$. We also recall that $\Delta\Xi_K \leq 0$. Hence, both the algorithm's cost and the increase in the potential function can be charged to the potential of the outer demands in B_w . Using Ineq. (3), we conclude that

$$\frac{x+1}{x}f + \Delta\Phi_K < \frac{4(\lambda+1)}{\lambda-2} \text{Pot}(M_w) \leq \frac{4(\lambda+1)}{\lambda-2} [(\lambda+1)\sigma^*|B_w| + ((\lambda+1)\rho + 1) \sum_{u \in B_w} d_u^*].$$

Case Isolated.B. $\text{Pot}(\Lambda_w) > \frac{\text{Pot}(B_w)}{2}$. Since $\Lambda_w \subseteq \ln(K)$, Lemma 4 implies that $d(\hat{w}, c_K^*) < 8 \max\{D(K), r(\ell_K)\}$. Hence, either the current phase ends or the coalition K becomes broken. In both cases, $\Delta\Xi_K \leq -f$ and the decrease in Ξ_K compensates for both the algorithm's cost and the decrease in Υ_K .

Case Typical. For any isolated active coalition K , $d(\hat{w}, c_K^*) \geq d(A, c_K^*)$. Therefore, no inner demands of K are included in B_w , because it would be $d(\hat{w}, c_K^*) < \frac{x}{3}d(A, c_K^*)$ otherwise. As a result, $\Delta\Phi_K = \Delta\Upsilon_K = 0$.

If w is an inner demand, let K_w be the unique typical active coalition such that $w \in \ln(K_w)$. Similarly to the proof of Lemma 7, we can show that $\frac{x+1}{x}f + \Delta\Phi_{K_w} \leq 0$. In addition, for any typical active coalition K' different from K_w , Lemma 7 implies that $\Delta\Phi_{K'} \leq 0$.

If w is an outer demand, using the following upper bound on $\text{Pot}(B_w)$, we can charge the algorithm's cost to the potential of B_w .

$$\frac{x+1}{x}f \leq \text{Pot}(B_w) \leq \frac{x+1}{x} \left[(\lambda+1)\sigma^*|B_w| + ((\lambda+1)\rho + 1) \sum_{u \in B_w} d_u^* \right] - \frac{x+1}{x} \frac{\lambda+1}{\lambda} \sum_{K \in \text{Act}(K)} \Delta\Upsilon_K$$

We conclude the proof by applying Lemma 7 for each typical active coalition. $\square$

In addition to the initial credit provided by the potential function Φ , a demand's optimal assignment cost is considered at most once by Lemma 6 (i.e., when the demand is added to L) and at most once by Lemma 8 (i.e., when the demand is removed from L). Therefore, the algorithm's total cost cannot exceed $\frac{2(2x+1)(\lambda+1)}{x(\lambda-2)} [\psi + 3 + \ln(\frac{\lambda+1}{\lambda}\gamma^2)] F^* + \frac{5\lambda+2}{\lambda-2} [(\lambda+1)\rho + \lambda + 2] S^*$. Setting $\gamma = 8\lambda$ and $\psi = \rho = O(\frac{\log n}{\log \log n})$ yields the claimed competitive ratio. $\square$

4 The Algorithm for Non-uniform Facility Costs

In this section, we outline the algorithm Non-Uniform Deterministic Facility Location – NDFL, which is a generalization of DFL and can handle non-uniform facility costs.

The algorithm first rounds down the facility costs to the nearest integral power of two. For each demand w , the algorithm computes r_w , B_w , $\text{Pot}(B_w)$, and $\hat{w}$ as in Fig. 1. If $|B_w| > 1$, NDFL opens the cheapest facility in $B(w, r_w) \cup B(\hat{w}, r_w)$ if its cost does not exceed $\text{Pot}(B_w)$. Ties are always broken in favour of $\hat{w}$. Namely, if there are many facilities of the same (cheapest) cost, the one nearest to $\hat{w}$ is opened. If a new facility is opened, the demands of B_w are removed from L . Otherwise, w is added to L . If $|B_w| = 1$, NDFL keeps opening the cheapest facility in $B(w, r_w)$ while there is a facility of cost no greater than $\text{Pot}(B_w)$. In this case, $\hat{w}$ coincides with w and ties are broken in favour of w . After opening a new facility, the algorithm updates r_w and $\text{Pot}(B_w)$ according to the new facility configuration and iterates. After the last iteration, w is added to L . As in Fig. 1, the algorithm finally assigns w to the nearest facility.

The following theorem can be proven by generalizing the techniques described in Section 3.

Theorem 3. *For any constant $x \geq 12$, the competitive ratio of NDFL is $O(\frac{\log n}{\log \log n})$.*

5 An Open Problem

In the framework of incremental clustering (e.g., [6,7]), an algorithm is also allowed to merge some of the existing clusters. On the other hand, the lower bound of Theorem 1 on the competitive ratio for Online Facility Location crucially depends on the restriction that facilities cannot be closed. A natural open question is how much the competitive ratio can be improved if the algorithm is also allowed to close a facility by re-assigning the demands to another facility (i.e., merge some of the existing clusters). This research direction is related to an open problem of [7] concerning the existence of an incremental algorithm for k -Median which achieves a constant performance ratio using $O(k)$ medians.

References

1. S. Albers and H. Koga. New online algorithms for the page replication problem. *J. of Algorithms*, 27(1):75–96, 1998.
2. B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. *Proc. of STOC '93*, pp. 164–173, 1993.
3. Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. *Proc. of FOCS '96*, pp. 184–193, 1996.
4. Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. *J. of Computer and System Sciences*, 51(3):341–358, 1995.

5. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
6. M. Charicar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *Proc. of STOC '97*, pages 626–635, 1997.
7. M. Charicar and R. Panigrahy. Clustering to minimize the sum of cluster diameters. *Proc. of STOC '01*, pages 1–10, 2001.
8. R. Fleischer and S. Seiden. New results for online page replication. *Proc. of APPROX '00, LNCS 1913*, pp. 144–154, 2000.
9. S. Guha. *Approximation Algorithms for Facility Location Problems*. PhD Thesis, Stanford University, 2000.
10. S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Proc. of SODA '98*, pp. 649–657, 1998.
11. M. Imase and B.M. Waxman. Dynamic Steiner tree problem. *SIAM J. on Discrete Mathematics*, 4(3):369–384, 1991.
12. K. Jain and V. Vazirani. Approximation algorithms for metric facility location and k -median problems using the primal-dual schema and Lagrangian relaxation. *J. of the ACM*, 48(2):274–296, 2001.
13. C. Lund, N. Reingold, J. Westbrook, and D.C.K. Yan. Competitive online algorithms for distributed data management. *SIAM J. on Computing*, 28(3):1086–1111, 1999.
14. M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. *Proc. of APPROX '02, LNCS 2462*, pp. 229–242, 2002.
15. R.R. Mettu and C.G. Plaxton. The online median problem. *Proc. of FOCS '00*, pp. 339–348, 2000.
16. A. Meyerson. Online facility location. *Proc. of FOCS '01*, pp. 426–431, 2001.
17. D. Shmoys. Approximation algorithms for facility location problems. *Proc. of APPROX '00, LNCS 1913*, pp. 27–33, 2000.
18. D. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. *Proc. of STOC '97*, pp. 265–274, 1997.

A Study of Integrated Document and Connection Caching^{*}

Susanne Albers¹ and Rob van Stee²

¹ Institut für Informatik, Albert-Ludwigs-Universität, Georges-Köhler-Allee, 79110 Freiburg, Germany. salbers@informatik.uni-freiburg.de.

² Centre for Mathematics and Computer Science (CWI), Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands. Rob.van.Stee@cw.nl.

Abstract. Document caching and connection caching are extensively studied problems. In document caching, one has to maintain caches containing documents accessible in a network. In connection caching, one has to maintain a set of open network connections that handle data transfer. Previous work investigated these two problems separately while in practice the problems occur together: In order to load a document, one has to establish a connection between network nodes if the required connection is not already open.

In this paper we present the first study that integrates document and connection caching. We first consider a very basic model in which all documents have the same size and the cost of loading a document or establishing a connection is equal to 1. We present deterministic and randomized online algorithms that achieve nearly optimal competitive ratios unless the size of the connection cache is extremely small. We then consider general settings where documents have varying sizes. We investigate a FAULT model in which the loading cost of a document is 1 as well as a BIT model in which the loading cost is equal to the size of the document.

1 Introduction

Recently there has been considerable research interest in document caching [5, 7, 8, 9, 10, 11, 12] and connection caching [2, 3, 4] in networks. In document caching, one has to maintain local caches containing documents available in the network. In connection caching, one has to maintain a set of open network connections that handle data transfer. However, previous work investigated these two problems separately, while in practice they are very closely related.

Consider a computer that is connected to a network. A user working at that computer wishes to access and download documents from other network sites. A downloaded document can be stored in local cache, so that it does not

^{*} Work supported by the Deutsche Forschungsgemeinschaft, Project AL 464/3-1, and by the European Community, Projects APPOL and APPOL II. Work done while the second author was at the Institut für Informatik, Albert-Ludwigs-Universität, Freiburg, Germany.

have to be retransmitted when the user wishes to access that document again. Serving requests to documents that are stored locally is much less expensive than transmitting requested documents over the network. Therefore, the local cache, which is of bounded capacity, should be maintained in a careful manner. The transmission of documents in a network is performed using protocols such as TCP (Transmission Control Protocol). If a network node v has to download a document available at node v' , then there has to exist an open (TCP) connection between v and v' . If the connection is not already open, it has to be established at a cost. Most networks, such as the Web, today work with persistent connections, i.e. an established connection can be kept open and reused later. However, each network node can only maintain a limited number of open connections and the collection of open connections can be viewed as a *connection cache*. The goal is to maintain this cache so that the connection establishment cost is as small as possible.

Clearly, caching decisions made on the document and connection levels heavily affect each other. Evicting a document d from the document cache at node v has a very negative effect if the connection between node v and node v' , where d is originally stored, is already closed. When d is requested again, one has to pay the connection establishment cost in addition to the necessary document transmission cost. A similar overhead occurs if a connection is closed that is frequently needed for data transfers. Therefore document and connection caching algorithms should coordinate their decisions. This can considerably improve the system's performance, i.e. the user perceived latency as well as the network congestion are reduced.

In this paper we present the first study of integrated document and connection caching. Formally, we consider a network node v . The node has two caches: one for the documents, also called *pages*, and one for the open connections currently maintained to other nodes. A sequence of *requests* must be served. Each request specifies a document d that the user at our network node wishes to access. If d resides in the document cache, then the request can be served at 0 cost. Otherwise a fault occurs and the request must be served by downloading d into the document cache at a cost of $\text{cost}(d) > 0$. Suppose that d is originally stored at network node v' . To load d into the document cache, an open connection must exist between v and v' . If the connection is already open, no cost is incurred. Otherwise the connection has to be established at a cost of $\text{cost}(v, v')$. The goal is to serve the request sequence so that the total cost is as small as possible.

The integrated document and connection caching problem is inherently online in that each request must be served without knowledge of future requests. We use competitive analysis to analyze the performance of online algorithms. We denote the cost of an algorithm $\mathcal{A}$ on a request sequence σ by $\mathcal{A}(\sigma)$. The optimal cost to serve this sequence is denoted by $\text{OPT}(\sigma)$. The goal of an online algorithm $\mathcal{A}$ is to minimize the competitive ratio $\mathcal{R}(\mathcal{A})$, which is defined as the smallest value $\mathcal{R}$ that satisfies $\mathcal{A}(\sigma) \leq \mathcal{R} \cdot \text{OPT}(\sigma) + a$, for any request sequence σ and some constant a independent of σ .

We remark here that a problem similar to that defined above arises in distributed databases. There, a user may have a file/page cache as well as a cache with pointers to files allowing fast access.

Previous work: As mentioned above document and connection caching have separately been the subjects of extensive research. There is a considerable body of work on document caching problems, see e.g [5,7,8,9,10,11,12]. However, the papers ignore that in a network setting, one may have to open a connection to load a document. If all documents have the same size and a loading cost of 1, which is the classical paging problem, the best competitive ratio of deterministic online algorithms is equal to k , where k is the number of documents that can be stored simultaneously in cache [11]. This competitiveness is achieved by the popular LRU (Least Recently Used) and FIFO (First-In First-Out) replacement strategies. On a fault, LRU evicts the page that was requested least recently and FIFO evicts the page that has been in cache longest. Fiat et al. [7] presented an elegant randomized paging algorithm called MARK that is $2H_k$ -competitive against oblivious adversaries, where H_k is the k -th Harmonic number. More complicated algorithms that achieve an optimal competitiveness of H_k were given in [1,10]. Irani [9] initiated the algorithmic study of the document caching problem when documents have different sizes. She considered a FAULT model where the loading cost of each document is equal to 1 as well as a BIT model, where the loading cost is equal to the size of the document. She presented randomized $O(\log^2 k)$ -competitive online algorithms for both settings. Young [12] gave a deterministic k -competitive online algorithm for a general cost model where the loading cost is an arbitrary non-negative value. Recently Feder et al. [5] studied a document caching problem where requests can be reordered. They concentrate on the case that the cache can hold one document. Gopalan et al. [8] study document caching in the Web when documents have expiration times. They assume all documents have the same size and a loading cost of 1.

Cohen et al. [3,4] introduced the connection caching problem. The input of the problem is a sequence of requests for TCP connections that must be established if not already open. Cohen et al. considered a distributed setting where requests occur at different network nodes. They gave deterministic k -competitive and randomized $O(H_k)$ -competitive online algorithms if all connections incur the same establishment cost. Here k is the maximum number of connections that a network node can keep open simultaneously. The case that connections can have varying establishment costs was considered in [2].

Our contribution: We investigate document and connection caching in an integrated manner. In the following let k be the number of documents that can be stored in the document cache and k' be the number of connections that can be kept open. We start by studying a basic setting in which all documents have the same size and a loading cost of 1; the connections have an establishment cost of 1. We present a deterministic online algorithm that achieves a competitive ratio of $k+4$ if $k' \geq k$ and a ratio of $\min\{2k - k' + 4, 2k\}$ if $k' < k$. Our algorithm uses LRU for the document cache and a phase based replacement strategy that tries

to keep connections of documents that may be evicted soon. We develop a lower bound on the performance of any deterministic online algorithm which implies that our algorithm is nearly optimal if k' is not extremely small. We also consider randomized online algorithms and prove that by replacing LRU by a randomized Marking strategy we obtain a competitive ratio of $2H_k + \min\{2H_k, 2(k - k') + 4\}$.

Additionally we investigate the problem that pages have varying sizes. If all documents have a loading cost of 1, which corresponds to Irani's FAULT model, we achieve a competitive ratio of $(4k + 14)/3$ if $k' \geq k$ and of $2k - 2k'/3 + 14/3$ if $k' < k$. Finally we consider a BIT model where the loading cost of a document is equal to the size of the document and the connection establishment cost is c , for some constant c . Here we prove a competitiveness of $(k + 5)(c' + 1)/2$ if $k' \geq k$, where $c' = c/s$ and s is the size of the smallest document ever requested. If $k' < k$, the competitiveness is $(2k - k' + 5)(c' + 1)/2$.

Finally we consider a distributed scenario, where requests can occur at different network nodes. We show that no deterministic online algorithm can in general be better than $2k$ -competitive, where k is the maximum number of documents that can be stored at any network node. A competitive ratio of $2k$ is easily achieved by an online algorithm that uses a k -competitive paging algorithm for the document cache and any replacement strategy for the connection cache.

2 Algorithms for the Basic Model

In this section we study a very basic scenario where all documents have the same size. Loading a missing document costs 1 and establishing a connection also costs 1.

2.1 Deterministic Algorithms

We present a deterministic online algorithm ALG for our basic setting. ALG works in phases. Each phase is defined as a maximal subsequence of requests to k distinct pages, which starts after the previous phase finishes (the first phase starts with the first request). Within each phase ALG works as follows.

At the beginning of each phase, evict all connections that were not used in the previous phase.
 On a page fault, use LRU to determine which page to evict from the page cache.
 On a connection fault, if there is a free slot in the cache, use it;
 otherwise, use MRU (Most Recently Used) to determine which connection to evict.

For ease of exposition, we first consider the case where the size of the connection cache is at least the same size as the page cache, i.e. $k' \geq k$. We then extend our analysis to the case $k' < k$.

Theorem 1. *If $k' \geq k$, then $\mathcal{R}(\text{ALG}) \leq k + 4$.*

Proof. Consider a request sequence σ . We first study the case that $k' = k$. Suppose there are $N + 1$ phases, numbered $0, 1, \dots, N$. For phase i , denote the number of page requests that cause a page fault by f_i ; the number of page requests that do not cause a page fault by p_i (these pages were requested in the previous phase by definition of LRU); the number of MRU faults m_i , and the number of holes created by h_i (i. e. the number of connections evicted at the start of phase i). Define $F = \sum_{i=1}^N f_i$, $M = \sum_{i=1}^N m_i$, $H = \sum_{i=2}^N h_i$ and $P = \sum_{i=1}^N p_i$. (We ignore phase 0.) Note $h_1 = 0$ and $f_i + p_i = k$ for each phase i .

Each hole that is created, is filled at most once, and this happens on a connection fault. (It is possible that some holes are never filled.) Thus the number of connection faults that cause holes to be filled is at most H . Furthermore, the remaining connection faults are exactly the connection faults where MRU is applied; this happens M times. Thus

$$\text{ALG}(\sigma) \leq F + M + H = kN + M + H - P. \quad (1)$$

Note that our algorithm is defined in such a way that the number of page faults is independent of the number of connection faults or the decisions as to which connections are evicted. The page cache is simply maintained by LRU. By definition of LRU, there must be one OPT page fault in each phase. Thus

$$\text{OPT}(\sigma) \geq N. \quad (2)$$

Each phase can be visualized as follows. The connection cache is at all times divided into two sets, PREVIOUS and CURRENT. Here PREVIOUS contains the connection slots that were not (yet) used in this phase, while CURRENT contains the connection slots that were used in the current phase. At the start of each phase, CURRENT is empty and PREVIOUS contains all k slots. Note that some of these slots may contain holes, in case a connection was evicted that was not used in the previous phase.

For each page fault in a phase, there are two possibilities:

1. No connection fault:
 - a) A not yet used connection slot is used for the first time in this phase (this connection was also used in the previous phase);
 - b) A connection slot already used in the current phase is used again (two or more pages are at the same node).
2. Connection fault occurs:
 - a) A hole is filled: a not yet used connection slot is used for the first time in this phase;
 - b) A connection slot already used in the current phase is used again by MRU;
 - c) (special case) A connection slot not yet used in the current phase is used by MRU.

Case 2.(c) can only occur if the very first page fault in a phase causes a connection fault; for a later page fault that also causes a connection fault, MRU always uses a slot that was already used in the current phase. From this list we have that only in cases 1.(a), 2.(a) and 2.(c), a connection slot moves from the set PREVIOUS to the set CURRENT.

Consider a phase $i > 0$. Suppose Case 2.(c) does not occur, and there are $m_i > 0$ MRU faults in phase i . Then at least m_i times, a connection slot already in CURRENT is used again. Hence at most $f_i - m_i$ times a connection slot moves from PREVIOUS to CURRENT. Therefore, at the end of phase i , there are at least $k - f_i + m_i$ connection slots still in PREVIOUS.

The pages requested in phase i can be divided into four groups:

1. pages that did not cause a page fault (p_i);
2. pages that caused a page fault, but no connection fault;
3. pages that caused a hole in the connection cache to be filled;
4. pages that caused a connection slot to be used again by MRU (m_i).

Every connection slot that at some point in phase i contains a connection to a page in group 2 or 3 (note that this may change later in the phase due to the use of MRU), is in CURRENT at the end of the phase. The other connection slots contain connections to pages that were either not requested in phase i (but were requested in phase $i - 1$, or they would have been evicted before), or that did not cause a fault. This last possibility occurs p_i times, so there are at least $k - f_i + m_i - p_i = m_i$ pages that are not requested again. This implies there are at least $k + m_i$ distinct pages requested in phase i and phase $i - 1$. Therefore OPT has at least m_i faults in phases $i - 1$ and i .

If Case 2.(c) does occur, then there were no holes at the start of phase i . Then the connections to the pages requested in phase $i - 1$ must all be distinct, $m_{i-1} = 0$ and $h_i = 0$. At the start of phase i , a connection slot moves from PREVIOUS to CURRENT using MRU. Case 2.(c) does not occur in the rest of the phase. Thus at the end of phase i , we have that there are at least $k - f_i + m_i - 1$ connection slots still in PREVIOUS. These slots correspond to connections that were used in the previous phase but not in this one, implying $k - f_i + m_i - p_i - 1 = m_i - 1$ pages that were requested in phase $i - 1$ but not in i . Then OPT has at least $m_i - 1$ faults in phases $i - 1$ and i . Moreover, it has at least one fault in phases $i - 2$ and $i - 1$, and $1 = m_{i-1} + 1$. By amortizing the cost, we find that OPT has at least m_i faults for every pair of phases $i - 1$ and i .

Thus $\text{OPT}(\sigma) \geq \sum_{i \text{ odd}} m_i$, and $\text{OPT}(\sigma) \geq \sum_{i \text{ even}} m_i$. This implies that

$$\text{OPT}(\sigma) \geq \frac{1}{2} \sum_{i>0} m_i = \frac{M}{2}. \quad (3)$$

The connections still in PREVIOUS at the end of phase i are evicted and become h_{i+1} holes. At most p_i of them lead to pages that were requested without a fault. Thus there are at least $k + h_{i+1} - p_i$ distinct pages requested in phases i and $i - 1$. This gives another bound for the cost of OPT:

$$\text{OPT}(\sigma) \geq \frac{1}{2} \sum_{i>0} (h_{i+1} - p_i) \geq \frac{H - P}{2} \quad (4)$$

Combining (1), (2), (3) and (4) gives

$$\text{ALG}(\sigma) \leq kN + M + H - P \leq k \cdot \text{OPT}(\sigma) + 2\text{OPT}(\sigma) + 2\text{OPT}(\sigma) = (k+4)\text{OPT}(\sigma).$$

This proves the ratio. It can be seen that the proof also holds for $k' > k$. $\square$

Theorem 2. *If $k' < k$, then $\mathcal{R}(\text{ALG}) \leq \min(k+4+(k-k'), 2k)$.*

Proof. Clearly, $\mathcal{R}(\text{ALG}) \leq 2k$ since ALG has at most $2k$ faults per phase (k connection faults and k page faults). We still have (2) and (4) by the exact same reasoning as in the proof of Theorem 1.

For m_i , we have again that each time that MRU is applied, no connection moves from PREVIOUS to CURRENT (unless Case 2.(c) occurs). So at most $f_i - m_i$ times a connection moves from PREVIOUS to CURRENT. Therefore, at the end of the phase, at least $k' - f_i + m_i$ connections are still in PREVIOUS. At most p_i of them refer to pages requested without a fault in phase i , so at least $k' - f_i + m_i - p_i = k' - k + m_i$ pages are requested in phase $i-1$ but not in phase i . Therefore there are at least $m_i + k'$ distinct pages requested in these two phases, and OPT has at least $m_i - (k - k')$ faults.

If Case 2.(c) occurs, there are only at least $k' - (f_i - (m_i - 1)) = m_i - (k - k') - 1$ connections still in PREVIOUS at the end. However, in that case we have $m_{i-1} \leq k - k'$ since there were no holes. Therefore $m_{i-1} - (k - k') \leq 0$ and we can amortize as before.

We therefore find

$$\text{OPT}(\sigma) \geq \frac{M - (k - k')N}{2}. \quad (5)$$

Using (2), this implies $M \leq 2\text{OPT}(\sigma) + (k - k')N \leq (k - k' + 2)\text{OPT}(\sigma)$. Therefore in this case

$$\text{ALG}(\sigma) \leq ((k+2) + (k - k' + 2))\text{OPT}(\sigma) \leq (2k - k' + 4)\text{OPT}(\sigma).$$

This proves the lemma. $\square$

2.2 Randomized Algorithms

For the standard paging problem, the randomized algorithm MARK is $2H_k$ -competitive, where H_k is the k -th Harmonic number [7]. Moreover, no randomized algorithm can have a competitive ratio less than H_k . The MARK algorithm processes a request sequence in phases. At the beginning of each phase, all pages in the memory system are unmarked. Whenever a page is requested, it is *marked*. On a fault, a page is chosen uniformly at random from among the unmarked pages in cache, and that page is evicted. A phase ends when all pages in cache are marked and a page fault occurs. Then, all marks are erased and a new phase is started.

In our algorithm ALG we substitute MARK for LRU to get a randomized algorithm. However, in this case it is also necessary to evict connections less

greedily to get a good performance. In particular, at the start of a phase we will not evict any connections that are associated with pages requested in the previous phase. Note that some of these connections may not have been used in that phase, because the relevant page might not have caused a page fault.

Theorem 3. *For the randomized version of ALG and $k' \geq k$, we have $\mathcal{R}(\text{ALG}) \leq 2H_k + 4$. For $k' < k$, we have. $\mathcal{R}(\text{ALG}) \leq 2H_k + \min(2H_k, 4 + 2(k - k'))$.*

Proof. We analyze this algorithm very similarly to the original analysis of MARK [7] and to the analysis in Section 2.1. We define q_i as the number of new pages requested in phase i . A page is new if it is not in the cache at the start of the phase. We define h_i , m_i , H and M as before and write $Q = \sum q_i$. Then by [7],

$$\text{ALG}(\sigma) \leq H_k Q + H + M.$$

Moreover, $\text{OPT}(\sigma) \geq Q/2$.

Following the proof of the deterministic case, we now have that every connection slot that at some point in phase i contains a connection to a page in group 1, 2 or 3 (note that this may change later in the phase due to the use of MRU), is in CURRENT at the end of the phase. Therefore any connections that are still in PREVIOUS at that time (which get evicted and form holes) must be to pages not requested in the phase. Therefore $\text{OPT}(\sigma) \geq H/2$.

Suppose $k' \geq k$. Due to the randomization, we do not know whether or not Case 2.(c) occurs in a phase. However, as observed in the proof of the deterministic algorithm, we can amortize the offline faults if 2.(c) occurs to get the bound $\text{OPT}(\sigma) \geq M/2$. Therefore analogously to in the proof of Theorem 1, we have

$$\mathcal{R}(\text{ALG}) \leq 2H_k + 4.$$

We now consider the case $k' < k$. The only change is that the bound $\text{OPT}(\sigma) \geq M/2$ is replaced by

$$\text{OPT}(\sigma) \geq \frac{M - (k - k')N}{2} \geq \frac{M - (k - k')Q}{2},$$

where we have used $Q \geq N$, which follows from the fact that there must be at least one new page in every new phase by definition of the phases. This gives us

$$\mathcal{R}(\text{ALG}) \leq \frac{H_k Q + H + M}{\text{OPT}(\sigma)} \leq 2H_k + 4 + 2(k - k').$$

However, since the number of connection faults, $H + M$, is also upper bounded by the number of page faults $H_k Q$, we find

$$\mathcal{R}(\text{ALG}) \leq 2H_k + \min(2H_k, 4 + 2(k - k')).$$

□

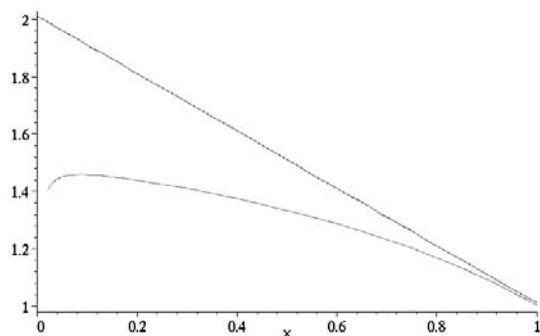


Fig. 1. The upper and lower bound: x -axis is k'/k , y -axis is $\mathcal{R}/k$

3 Lower Bounds

We present a lower bound on the performance of any deterministic online algorithm. The lower bound of Theorem 4 implies that if k' is not too small, our deterministic algorithm given in the last section is nearly optimal. Figure 1 depicts the lower as well as the upper bound.

Theorem 4. *Suppose $k' \geq 2$ and let $\alpha = k'/k$. Then for any online algorithm $\mathcal{A}$, we have*

$$\mathcal{R}(\mathcal{A}) \geq (k+1) \left(\frac{\alpha k - 1}{\alpha k} + \frac{1 - \alpha}{2 - \alpha + 3/k} \right).$$

Proof. We construct a lower bound as follows. We make use of $k+1$ pages that are stored at $k+1$ distinct nodes. Consider an online algorithm $\mathcal{A}$. Each page request in the sequence is to the (unique) page that $\mathcal{A}$ does not have in its cache. The sequence is divided into phases. In each phase, we count the number of distinct pages that have been requested in that phase; the first request to the $k+1$ st distinct page is defined to be the start of the next phase. Since the connection cache has size k' , $\mathcal{A}$ must have at least $k-k'$ connection faults in each phase. We define $\alpha = k'/k$, so that $k' = \alpha k$. We will write the average length of a phase as pk , where $p \geq 1$. The offline algorithm uses one of the following strategies depending on p .

Strategy 1. (For large p .) The first strategy is to always use LFD for the requested pages. We then count the number of offline page faults for each of the $k+1$ pages, and put $k'-1$ connections to pages on which the most offline faults occur, in the connection cache. This part of the connection cache is fixed during the entire processing of the request sequence. The last slot is used for connection faults on the remaining $k+1 - (k'-1) = k - k' + 2$ pages.

Consider $k+1$ phases. There are at most $k+1$ offline faults, and on average at most $k-k'+2$ of them are on pages of which the connections are not in the connection cache at all times. Thus there are on average at most $2k - k' + 3$ offline faults on $k+1$ phases.

Strategy 2. (For small p .) The second strategy begins by counting the number of requests to each page over the entire request sequence. Then, the $k - k' + 1$ pages that are requested the most often, are put in the page cache at the beginning, and the k' connections to the remaining pages are put in the connection cache. The entire connection cache is fixed throughout the sequence. The offline algorithm now uses LFD on the k' pages for which the connections are in the connection cache, and only uses the $k' - 1$ slots in the page cache that do not contain the $k - k' + 1$ most often requested pages. It has no connection faults at all.

Consider $(k + 1)(k' - 1)$ phases. These contain on average $(k + 1)(k' - 1)pk$ requests by definition of p . Thus, each page is requested on average $(k' - 1)pk$ times. The k' pages that are requested the least overall, must then be requested at most $k'(k' - 1)pk$ times on average at most. Since the offline algorithm has at most one fault every $k' - 1$ requests to this subset of pages, there are $k'pk$ offline faults.

Solving for p , we find that these two strategies have the same number of faults if

$$p = \frac{\alpha k - 1}{\alpha k} \left(2 - \alpha + \frac{3}{k} \right). \quad (6)$$

As long as this value is at least 1, we can use the first offline strategy if p is greater than the threshold, and the second strategy otherwise. The number of on-line faults in one phase must be at least $pk + (k - k')$ on average. This implies a competitive ratio of at least

$$\frac{(pk + k - k')(k + 1)(\alpha k - 1)}{k'pk} = (k + 1) \left(\frac{\alpha k - 1}{\alpha k} + \frac{1 - \alpha}{2 - \alpha + 3/k} \right).$$

Note that the threshold in (6) is greater than 1 for $k \geq k' \geq 2$. $\square$

We can show that the analysis of our algorithm ALG is asymptotically tight for $k' = 1$. Note that ALG behaves exactly like LRU in this case. This implies that even for $k' = 1$ it is nontrivial to find an algorithm with competitive ratio close to k .

Lemma 1. *For $k' = 1$, we have $\mathcal{R}(\text{ALG}) \geq 2k - 2$.*

Proof. We use a set of pages numbered $1, 2, \dots, k + 1$ and request them cyclically. All the odd pages are at some node v_1 while the even pages are at another node v_2 . It can be seen that our algorithm has a connection fault on every request, thus it has $2k$ faults per phase.

We now describe an off-line algorithm to serve this sequence. This algorithm only faults on pages in v_1 , and each time evicts the page from that node that will be requested the furthest in the future. All pages in v_2 are in the cache at all times. Suppose k is even, then there are $k/2$ slots available in the cache for $k/2 + 1$ pages. Thus this off-line algorithm has a fault once every $k/2$ requests to pages in v_1 .

Consider $k + 1$ phases. It contains $k(k + 1)$ requests, exactly k per page. Thus there are $2(k/2 + 1) = k + 2$ offline faults in total, giving a competitive ratio of

$$\frac{2k(k + 1)}{k + 2} = 2k - \frac{2k}{k + 2} \geq 2k - 2.$$

For odd k , there is one off-line fault per $(k - 1)/2$ requests to pages in v_1 . In $k - 1$ phases there are $k(k - 1)$ requests, thus $k(k - 1)/2$ requests to pages in v_1 and in total k offline faults. This gives a ratio of exactly $2k - 2$. $\square$

4 Generalized Models

In this section we study generalized problem settings in which the documents can have different sizes. For the standard multi-sized paging problem, the algorithm LRU is $(k + 1)$ -competitive in both the BIT and the FAULT model [6]. Here k is defined as the maximum number of pages that can fit in the cache, i.e. $k = K/s$ where K is the size of the cache (in bits) and s is the size of the smallest possible page. It is nontrivial to extend the analysis of our algorithm to these models.

In both models, a *phase* is now defined as a maximal subsequence of requests to a minimal volume of distinct pages that is larger than K . Thus there are at most $k + 1$ page faults in a phase.

4.1 The Fault Model

For the FAULT model, we need to consider the number of pages requested in each phase, which can be less than k .

Theorem 5. *In the FAULT model, $\mathcal{R}(\text{ALG}) \leq (4k + 14)/3$ for $k' \geq k$ and $\mathcal{R}(\text{ALG}) \leq 2k - \frac{2}{3}k' + \frac{14}{3}$ for $k' < k$.*

Proof. Suppose $k' = k$. Denote the number of pages requested in phase i by Φ_i . Write $\Delta_i = \Phi_i - \Phi_{i-1}$.

If there are m_i connection faults where MRU is applied, then m_i times a connection slot remains in CURRENT. Thus at most $k + 1 - m_i$ times a connection slot moves from PREVIOUS to CURRENT, and at least $m_i - 1$ connection slots are still in PREVIOUS at the end of the phase. These connections lead to at least $m_i - 1$ pages that were requested in phase $i - 1$ but not in phase i .

Denote the set of pages requested in phase $i - 1$ but not in phase i by F . Denote the set of pages requested in phase i by S . We partition F in two sets: F_1 contains the pages that OPT faults on, F_2 contains the rest. Consider the set F_2 . OPT does not fault on these pages and thus has them in its cache at the start of phase $i - 1$. This means that some pages in S are not yet in its cache and need to be loaded later.

Write the number of OPT faults in these two phases as $m_i - 1 - x$. If $x \leq 0$, we are done. Otherwise, F_2 contains $z \geq x > 0$ pages. OPT has exactly $z - x$ faults on the set S , that is, $z - x$ pages are loaded to come “in the place of”

the z pages in F_2 (OPT does not necessarily replace exactly these pages in the cache). Since at most $k + 1$ pages were requested in phase $i - 1$, the set S then contains at most $k + 1 - x$ pages, i.e. our algorithm has at most $k + 1 - x$ page faults in phase $i + 1$.

That is, if OPT has x faults less than $m_i - 1$ in phases $i - 1$ and i , then our algorithm has (at least) x faults less than $k + 1$ in phase i . Writing the number of OPT faults as $m_i - 1 - x_i$ in all cases where it is less than $m_i - 1$, this gives

$$\text{OPT}(\sigma) \geq \frac{M - N - X}{2},$$

where $X = \sum x_i$. (That is, all values x_i are positive.)

We can treat the holes that are created in the same way to find

$$\text{OPT}(\sigma) \geq \frac{H - P - X}{2}.$$

Finally we also still have $\text{OPT} \geq N$. We have

$$\text{ALG}(\sigma) \leq (k + 1)N - X + M + H - P \quad \text{and} \quad \text{ALG} \leq 2((k + 1)N - X),$$

where the second inequality follows since ALG has at most one connection fault for each page fault.

Thus if $X \geq \frac{kN-4}{3}$, we find that the competitive ratio is at most $4k/3 + 14/3$. On the other hand, if $X < \frac{kN-4}{3}$, then

$$\text{ALG}(\sigma) \leq (k + 1)\text{OPT}(\sigma) + 4\text{OPT}(\sigma) + X \leq (k + 5 + k/3 - 4/3)\text{OPT}(\sigma) = \frac{4k + 14}{3}\text{OPT}(\sigma).$$

This analysis can easily be extended to the case $k' < k$ as before, giving $\mathcal{R}(\text{ALG}) \leq 2k - \frac{2}{3}k' + \frac{14}{3}$. Details are omitted in this extended abstract. $\square$

4.2 The Bit Model

In this section we investigate a BIT model in which the cost of loading a document is equal to the size of the document. We also assume that the cost of establishing a connection is equal to c , for some constant $c > 0$.

Theorem 6. *In the BIT model, $\mathcal{R}(\text{ALG}) \leq \frac{k+5}{2}(c'+1)$ for $k' \geq k$, where $c' = c/s$ is the cost of a connection fault divided by the size of the smallest possible page. For $k' < k$, $\mathcal{R}(\text{ALG}) \leq \frac{2k+5-k'}{2}(c' + 1)$.*

Proof. Denote the average phase length by $K + \delta$ for some $\delta > 0$. Denote the average number of MRU faults in a phase by m and the average number of bits worth of old pages that are implied by m' , then $m' \geq ms$. Denote the average number of pages on which there is no fault in a phase by p and the average number of bits that are requested without fault by p' , then $p' \geq ps$. Finally, denote the average number of holes created in a phase by h . Denote the cost of a single connection fault by c and write $c' = c/s$. Similarly to in the previous

section, it can be seen that for the average cost in a phase we have $\text{ALG}/s \leq k + \delta/s + (m + h)c/s - p'/s$ and $\text{OPT}/s \geq \max(\max(1, \delta/s), m/2, h - p/2)$. Here the first maximum in the second equation follows from $\sum_i \max(\delta_i, s)/Ns \geq \max(Ns, N\delta)/Ns = \max(1, \delta/s)$, where $K + \delta_i$ is the amount of bits from distinct requests requested in phase i .

Since the number of connection faults in a phase is bounded from above by the number of page faults, we have

$$m + h \leq \frac{K + \delta - p'}{s} \Rightarrow h \leq k + \frac{\delta}{s} - p - m \leq (k + 1) \frac{\text{OPT}}{s} - p - m. \quad (7)$$

We also have $h \leq 2 \frac{\text{OPT}}{s} + p$. Note that $2 \frac{\text{OPT}}{s} + p = (k + 1) \frac{\text{OPT}}{s} - p - m \Rightarrow 2p = (k - 1) \text{OPT}/s - m$. Suppose $p \leq ((k - 1) \text{OPT}/s - m)/2$. (The other case is handled similarly.) Then

$$\begin{aligned} \frac{\text{ALG}}{s} &\leq (k + 1) \frac{\text{OPT}}{s} + mc' + hc' - p \leq (k + 1) \frac{\text{OPT}}{s} + mc' + 2 \frac{\text{OPT}}{s} c' + p(c' - 1) \\ &\leq (k + 1) \frac{\text{OPT}}{s} + mc' + 2 \frac{\text{OPT}}{s} c' + (c' - 1)((k - 1) \frac{\text{OPT}}{s} - m)/2 \\ &\leq (k + 1) \frac{\text{OPT}}{s} + (c' + 1)m/2 + 2 \frac{\text{OPT}}{s} c' + (c' - 1)(k - 1) \frac{\text{OPT}}{2s} \\ &\leq (k + c' + 2 + 2c' + \frac{(c' - 1)(k - 1)}{2}) \frac{\text{OPT}}{s} = \frac{(k + 5)(c' + 1)}{2} \cdot \frac{\text{OPT}}{s}. \end{aligned}$$

For $k' < k$, we have $\text{OPT}(\sigma)/s \geq (m - (k - k'))/2$ and $\mathcal{R}(\text{ALG}) \leq \frac{(2k - k' + 5)(c' + 1)}{2}$; details are omitted in this extended abstract. $\square$

Hence the competitive ratio grows linearly with k and with c (c'). The reason for this is that we cannot identify connection faults by OPT; it is conceivable that OPT never has a connection fault.

5 The Distributed Setting

We finally study the distributed problem setting where requests can occur at various network nodes. Again, each node has a document cache and a connection cache. Here, a request is specified by a pair (v, d) , indicating that document d is requested by the user at node v . The cost of serving requests is the same as before. The crucial difference is in the usage of connections. An open connection between nodes v and v' can be used for downloading documents from v to v' as well as from v' to v . However, if one of the nodes of the connection decides to close the connection, then the connection cannot be used by the other node either. Hence, the connection cache configurations affect each other.

Theorem 7. *In the distributed problem setting, no deterministic online algorithm can achieve a competitive ratio smaller than $2k/(1 + 1/k')$, where k is the size of the largest document cache and k' is the maximum number of connections that a network node can keep open.*

Proof. Consider a node v at which $k + 1$ documents are stored. Additionally we have $k' + 1$ nodes $v_i, 1 \leq i \leq k' + 1$. Each node in the network has a document cache of size k and a connection cache of size k' . Requests are generated as follows. At any time one of the connections (v, v_i) is closed in the configuration of an online algorithm A because v can only maintain k' open connections and a connection is open only if it is cached by both of its endpoints. An adversary generates a request at this node v_i for the document that is currently not stored in A 's document cache at v_i . Suppose that a request sequence consists of m requests and that m_i requests were generated at $v_i, 1 \leq i \leq k' + 1$. The online cost is equal to $2m$. An optimal offline algorithm has at most $\lceil \frac{m_i}{k} \rceil$ document faults at v_i and hence no more than $\frac{m}{k} + k' + 1$ document faults in total. Furthermore an optimal algorithm can maintain the connection cache at v in such a way that at most $\lceil (\frac{m}{k} + k' + 1)/k' \rceil$ connection faults occur. Thus as $m \rightarrow \infty$, the ratio of the online to offline cost tends to $2/(\frac{1}{k} + \frac{1}{kk'}) = 2k(1 + 1/k')$. $\square$

Note that a competitive ratio of $2k$ is achieved by any caching algorithm that uses a k -competitive paging strategy for the document cache any replacement rule for the connection cache.

6 Conclusions

In this paper we studied integrated document and connection caching in a variety of problem settings. An open question left by our work is to find a better algorithm for the case where the connection cache is very small (relative to k). We conjecture that the true competitive ratio for this problem should be close to k .

References

1. D. Achlioptas, M. Chrobak, and J. Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234:203–218, 2000.
2. S. Albers. Generalized connection caching. In *Proceedings of the Twelfth ACM Symposium on Parallel Algorithms and Architectures*, pages 70–78. ACM, 2000.
3. E. Cohen, H. Kaplan, and U. Zwick. Connection caching. In *Proceedings of the 31st ACM Symposium on the Theory of Computing*, pages 612–621. ACM, 1999.
4. E. Cohen, H. Kaplan, and U. Zwick. Connection caching under various models of communication. In *Proceedings of the Twelfth ACM Symposium on Parallel Algorithms and Architectures*, pages 54–63. ACM, 2000.
5. T. Feder, R. Motwani, R. Panigrahy, and A. Zhu. Web caching with request reordering. In *Proceedings 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 104–105, 2002.
6. A. Feldman, R. Karp, M. Luby, and L. A. McGeoch. Personal communication cited in [9].
7. A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, and N.E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, Dec 1991.

8. P. Gopalan, H. Karloff, A. Mehta, M. Mihail, and N. Vishnoi. Caching with expiration times. In *Proceedings 13th ACM-SIAM Symposium on Discrete Algorithms*, pages 540–547, 2002.
9. S. Irani. Page replacement with multi-size pages and applications to web caching. In *Proceedings 29th ACM Symposium on Theory of Computing*, pages 701–710, 1997.
10. L. McGeoch and D. Sleator. A strongly competitive randomized paging algorithm. *J. Algorithms*, 6:816–825, 1991.
11. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.
12. N. Young. On-line file caching. In *Proceedings 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 82–86, 1998.

A Solvable Class of Quadratic Diophantine Equations with Applications to Verification of Infinite-State Systems

Gaoyan Xie¹, Zhe Dang^{1*}, and Oscar H. Ibarra^{2**}

¹ School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA

² Department of Computer Science
University of California
Santa Barbara, CA 93106, USA

Abstract. A *k*-system consists of *k* quadratic Diophantine equations over nonnegative integer variables $s_1, \dots, s_m, t_1, \dots, t_n$ of the form:

$$\begin{aligned} \sum_{1 \leq j \leq l} B_{1j}(t_1, \dots, t_n) A_{1j}(s_1, \dots, s_m) &= C_1(s_1, \dots, s_m) \\ &\vdots \\ \sum_{1 \leq j \leq l} B_{kj}(t_1, \dots, t_n) A_{kj}(s_1, \dots, s_m) &= C_k(s_1, \dots, s_m) \end{aligned}$$

where l, n, m are positive integers, the B 's are nonnegative linear polynomials over $t_1, \dots, t_n$ (i.e., they are of the form $b_0 + b_1 t_1 + \dots + b_n t_n$, where each b_i is a nonnegative integer), and the A 's and C 's are nonnegative linear polynomials over $s_1, \dots, s_m$. We show that it is decidable to determine, given any 2-system, whether it has a solution in $s_1, \dots, s_m, t_1, \dots, t_n$, and give applications of this result to some interesting problems in verification of infinite-state systems. The general problem is undecidable; in fact, there is a fixed $k > 2$ for which the *k*-system problem is undecidable. However, certain special cases are decidable and these, too, have applications to verification.

1 Introduction

During the past decade, there has been significant progress in automated verification techniques for finite-state systems. One such technique is model-checking [5,19] that explores the state space of a finite-state system and checks that a desired temporal property is satisfied. Model-checkers like SMV [13] and SPIN [10] have been successful in many industrial-level applications. The successes have greatly inspired researchers to develop automatic techniques for analyzing infinite-state systems (such as systems that contain integer variables and parameters). However, in general, it is not possible to develop such techniques,

* Corresponding author (zdang@eecs.wsu.edu).

** The research of Oscar H. Ibarra has been supported in part by NSF Grants IIS-0101134 and CCR02-08595.

e.g., it is not possible to (automatically) verify whether an arithmetic program with two integer variables is going to halt [17]. Therefore, an important aspect of the research on infinite-state system verification is to identify what kinds of practically useful infinite-state models are decidable with respect to a particular form of properties (e.g., reachability).

In this paper, we look at a class of infinite-state systems that contain parameterized or unspecified constants. For instance, consider a nondeterministic finite state machine M . Each transition in M is assigned a label. On firing the transition $s \rightarrow^a s'$ from state s to state s' with label a , an *activity* a is performed. There are finitely many labels $a_1, \dots, a_l$ in M . M can be used to model, among others, a finite state process where an execution of the process corresponds to an execution path (e.g., $s_0 \rightarrow^{a^0} s_1 \rightarrow^{a^1} \dots \rightarrow^{a^r} s_{r+1}$, for some r) in M . On the path, a sequence of activities $a^0 \dots a^r$ are performed. Let $\Sigma_1, \dots, \Sigma_k$ be any k sets (not necessarily disjoint) of labels. An activity a is of type i if $a \in \Sigma_i$. An activity could have multiple types. Additionally, activities $a_1, \dots, a_l$ are associated with *weights* $w_1, \dots, w_l$ that are unspecified (or parameterized) constants in $\mathbf{N}$, respectively. Depending on the various application domains, the weight of an activity can be interpreted as, e.g., the time in seconds, the bytes of memory, or the budget in dollars, etc., needed to complete the activity. A type of activities is useful to model a “cluster” of activities. When executing M , we use nonnegative integer variables W_i to denote the accumulated weight of all the activities of type i performed so far, $1 \leq i \leq k$. One verification question concerns reachability:

- (*) whether, for some values of the parameterized constants $w_1, \dots, w_l$, there is an execution path from a given state to another on which $w_1, \dots, w_l, W_1, \dots, W_k$ satisfy a given Presburger formula P (a Boolean combination of linear constraints and congruences).

One can easily find applications for the verification problem. For instance, consider a packet-based network switch that uses a scheduling discipline to decide the order in which the packets from different incoming connections $c_1, \dots, c_l$ are serviced (visited). Suppose that each connection c_i is assigned a weight w_i , $1 \leq i \leq l$, and each time when a connection is serviced (visited), the number of packets serviced from that connection is in proportion to its weight. But in this switch we have two outgoing connections (two servers in the “queue theory” jargon) o_1 and o_2 each of which serves a set of incoming connections C_1 and C_2 respectively ($C_1 \cup C_2 = \{c_1, \dots, c_l\}$). The scheduling discipline for this switch can be modeled as a finite state system. If we take the event that an incoming connection is serviced by a specific server as an activity, then the weight of the activity could be the number of packets served that is in proportion to the weight of the incoming connection. Thus W_1 and W_2 could be used to denote the total amount (accumulated weights) of packets served by the two servers respectively. Later in the paper, we shall see how to model a fairness property using (*).

In this paper, we study the verification problem in (*) and its variants. First, we show that the problem is undecidable, in general. Then, we consider various restricted as well as modified cases in which the problem becomes decidable.

For instance, if P in (*) has only one linear constraint that contains some of $W_1, \dots, W_k$, then the problem is decidable. Also, rather surprisingly, if in the problem in (*) we assume that the weight of each activity a_i can be nondeterministically chosen as any value between a concrete constant (such as 5) and a parameterized constant w_i , then it becomes decidable. We also consider cases when the transition system is augmented with other unbounded data structures, such as a pushdown stack, dense clocks, and other restricted counters.

In the heart of our decidability proofs, we first show that some special classes of systems of quadratic Diophantine equations/inequalities are decidable (though in general, these systems are undecidable [16]). This nonlinear Diophantine approach towards verification problems is significantly different from many existing techniques for analyzing infinite-state systems (e.g., automata-theoretic techniques in [14,3,7], computing closures for Presburger transition systems [6, 4], etc.). Then, we study a more general version of the verification problem by considering weighted semilinear languages in which a symbol is associated with a weight. Using the decidability results on the restricted classes of quadratic Diophantine systems, we show that various verification problems concerning weighted semilinear languages are decidable. Finally, as applications, we “re-interpret” the decidability results for weighted semilinear languages into the results for some classes of machine models, whose behaviors (e.g., languages accepted, reachability sets, etc) are known to be semilinear, augmented with weighted activities.

Adding weighted activities to a transition system can be found, for instance, in [15]. In that paper, a “price” is associated with a control state in a timed automaton [2]. The price may be very complex; e.g., linear in other clock values etc. In general, the reachability problem for priced timed automata is undecidable [15]. Here, we are mainly interested in the decidable cases of the problem: what kind of “prices” (i.e., weights) can be placed such that some verification queries are still decidable, for transition systems like pushdown automata, restricted counter machines, etc., in addition to timed automata.

The paper is organized as follows. In the next section, we present the decidability results for the satisfiability problem of two special classes of quadratic Diophantine systems (Lemma 2 and Theorem 1). Then in Section 3, we generalize the verification problem in (*) in terms of weighted semilinear languages, and reduce the problem and its restricted versions to the classes of quadratic Diophantine systems studied in Section 2. In Section 4, we discuss the application aspects and extensions of the decidability results to other machine models. Due to space limitation, some of the proofs are omitted in the paper. The full version of the paper is accessible at www.eecs.wsu.edu/~zdang.

2 Preliminaries

Let $\mathbf{N}$ be the set of nonnegative integers and let $x_1, \dots, x_n$ be n variables over $\mathbf{N}$. A *linear constraint* is defined as $a_1x_1 + \dots + a_nx_n > b$, where $a_1, \dots, a_n$ and b are integers. A *congruence* is $x_i \equiv_b c$, where $1 \leq i \leq n$, and $b \neq 0, 0 \leq$

$c < b$. A Presburger formula is a Boolean combination of linear constraints and congruences using $\vee$ and $\neg$. Notice that, here, Presburger formulas are defined over nonnegative integer variables (instead of integer variables). It is well known that Presburger formulas are closed under quantifications ($\forall$ and $\exists$).

A subset S of $\mathbf{N}^n$ is a *linear set* if there exist vectors $v_0, v_1, \dots, v_t$ in $\mathbf{N}^n$ such that $S = \{v | v = v_0 + b_1 v_1 + \dots + b_t v_t, b_i \in \mathbf{N}\}$. The set S is a *semilinear set* if it is a finite union of linear sets. It is well known that S is semilinear iff S is Presburger definable (i.e., there is a Presburger formula P such that $P(v)$ iff $v \in S$).

A *linear polynomial* is a polynomial of the form $a_0 + a_1 x_1 + \dots + a_n x_n$ where each coefficient a_i , $0 \leq i \leq n$, is an integer. The polynomial is *constant* if each $a_i = 0$, $1 \leq i \leq n$. The polynomial is *nonnegative* if each a_i , $0 \leq i \leq n$, is in $\mathbf{N}$. The polynomial is *positive* if it is nonnegative and $a_0 > 0$. A variable *appears* in a linear polynomial iff its coefficient in that polynomial is nonzero. The following result is needed in the paper.

Lemma 1. *It is decidable whether an equation of the following form has a solution in nonnegative integer variables $s_1, \dots, s_m, t_1, \dots, t_n$:*

$$L_0 + L_1 t_1 + \dots + L_n t_n = 0 \quad (1)$$

where $L_0, L_1, \dots, L_n$ are linear polynomials over $s_1, \dots, s_m$. The decidability remains even when the solution is restricted to satisfy a given Presburger formula P over $s_1, \dots, s_m$.

Proof. The first part of the lemma has already been proved in [8], while the second part is shown below using a “semilinear transform”. As we mentioned earlier, the set of all $(s_1, \dots, s_m) \in \mathbf{N}^m$ satisfying P is a semilinear set (i.e., a finite union of linear sets). For each linear set of P , one can find nonnegative integer variables $u_1, \dots, u_k$ for some k and a nonnegative linear polynomial $p_i(u_1, \dots, u_k)$ for each $1 \leq i \leq m$ such that $(s_1, \dots, s_m)$ is in the linear set iff each $s_i = p_i(u_1, \dots, u_k)$, for some $u_1, \dots, u_k$. The second part follows from the first part by substituting $p_i(u_1, \dots, u_k)$ for s_i in $L_0, L_1, \dots, L_n$. ■

Let I, J and K be three pairwise disjoint subsets of $\{1, \dots, n\}$. An *n-inequality* is an inequality over n nonnegative integer variables $t_1, \dots, t_n$ and m (for some m) nonnegative integer variables $s_1, \dots, s_m$ of the following form:

$$\begin{aligned} D_1 + a \left(\sum_{i \in I} L_{1i} t_i + \sum_{j \in J} L_{1j} t_j \right) &\leq D_2 + \sum_{i \in I} L_{2i} t_i + \sum_{k \in K} L_{2k} t_k \\ &\leq D'_1 + a' \left(\sum_{i \in I} L_{1i} t_i + \sum_{j \in J} L_{1j} t_j \right), \end{aligned} \quad (2)$$

where $a < a' \in \mathbf{N}$, the D 's (resp. the L 's) are nonnegative (resp. positive) linear polynomials over $s_1, \dots, s_m$, and $D_1 \leq D'_1$ is always true (i.e., true for all $s_1, \dots, s_m \in \mathbf{N}$).

Lemma 2. *For any n , it is decidable whether an n -inequality in (2) has a solution in nonnegative integer variables $s_1, \dots, s_m, t_1, \dots, t_n$. The decidability remains even when the solution is restricted to satisfy a given Presburger formula P over $s_1, \dots, s_m$.*

Theorem 1. *It is decidable whether a system in the following form has a solution in nonnegative integer variables $s_1, \dots, s_m, t_1, \dots, t_n$: $P(D_1 + \sum_{1 \leq i \leq n} L_{1i}t_i, D_2 + \sum_{1 \leq i \leq n} L_{2i}t_i)$, where P is a Presburger formula over two nonnegative integer variables and the D 's and the L 's are nonnegative linear polynomials over $s_1, \dots, s_m$.*

3 Semilinear Languages with Weights

We first recall the definition of semilinear languages. Let $\Sigma = \{a_1, \dots, a_l\}$ be an alphabet. For each word α in Σ^* , the *Parikh* map of α is defined to be $\phi(\alpha) = (\phi_{a_1}(\alpha), \dots, \phi_{a_l}(\alpha))$ where $\phi_{a_i}(\alpha)$ denotes the number of occurrences of symbol a_i in word α , $1 \leq i \leq l$. For a language $L \in \Sigma^*$, the Parikh map of L is $\phi(L) = \{\phi(\alpha) | \alpha \in L\}$. The language L is semilinear iff $\phi(L)$ is a semilinear set. L is effectively semilinear if the semilinear set $\phi(L)$ can be computed from the description of L .

Now, we add “weights” to a language L . A *weight measure* is a mapping that maps a symbol in Σ to a weight in $\mathbf{N}$. We shall use $w_1, \dots, w_l$ to denote the weights for $a_1, \dots, a_l$, respectively, under the measure. Let $\Sigma_1, \dots, \Sigma_k$ be any k fixed subsets of Σ . For each $1 \leq i \leq k$, we use $W_i(\alpha)$ to denote the total weight of all the occurrences for symbols $a \in \Sigma_i$ in word α ; i.e.,

$$W_i(\alpha) = \sum_{a_j \in \Sigma_i} w_j \cdot \phi_{a_j}(\alpha). \quad (3)$$

$W_i(\alpha)$ is called the *accumulated weight* of α wrt Σ_i . We are interested in the following *k-accumulated weight problem*:

- **Given:** An effectively semilinear language L , k subsets $\Sigma_1, \dots, \Sigma_k$ of Σ , and a Presburger formula P over $l + k$ variables.
- **Question:** Is there a word α in L such that, for some $w_1, \dots, w_l \in \mathbf{N}$,

$$P(w_1, \dots, w_l, W_1(\alpha), \dots, W_k(\alpha)) \quad (4)$$

holds?

In a later section, we shall look at the application side of the problem. The rest of this section investigates the decidability issues of the problem by transforming the problem and its restricted versions to a class of Diophantine equations.

A *k-system* is a quadratic Diophantine equation system that consists of k equations over nonnegative integer variables $s_1, \dots, s_m, t_1, \dots, t_n$ (for some m, n) in the following form:

$$\begin{cases} \sum_{1 \leq j \leq l} B_{1j}(t_1, \dots, t_n) A_{1j}(s_1, \dots, s_m) = C_1(s_1, \dots, s_m) \\ \vdots \\ \sum_{1 \leq j \leq l} B_{kj}(t_1, \dots, t_n) A_{kj}(s_1, \dots, s_m) = C_k(s_1, \dots, s_m) \end{cases} \quad (5)$$

where the A 's, B 's and C 's are nonnegative linear polynomials, and l, n, m are positive integers.

Theorem 2. *For each k , the k -accumulated weight problem is decidable iff it is decidable whether a k -system has a solution.*

It is known [12] that there is a fixed k such that there is no algorithm to solve Diophantine systems in the following form: $t_1 F_1 = G_1$, $t_1 H_1 = I_1$, $\dots$, $t_k F_k = G_k$, $t_k H_k = I_k$, where the F 's, G 's, H 's, I 's are nonnegative linear polynomials over nonnegative integer variables $s_1, \dots, s_m$, for some m . Observe that the above systems are $2k$ -systems. Therefore, from Theorem 2,

Theorem 3. *There is a fixed k such that the k -accumulated weight problem is undecidable.*

Currently, it is an open problem to find the maximal k such that the k -accumulated weight problem is decidable. Clearly, when $k = 1$, the problem is decidable. This is because 1-systems are decidable (Lemma 1). Below, using Theorem 1, we show that the problem is decidable when $k = 2$. Interestingly, it is still open whether the decidability remains for $k = 3$.

Theorem 4. *The 2-accumulated weight problem is decidable.*

In some restricted cases, the accumulated weight problem is decidable for a general k . We are now going to elaborate these cases. Consider a k -accumulated weight problem such that (4) is a disjunction of formulas in the following special form:

$$Q(w_1, \dots, w_l) \wedge a_1 W_1(\alpha) + \dots + a_k W_k(\alpha) + b_1 w_1 + \dots + b_l w_l \sim a_0 \quad (6)$$

where Q is a Presburger formula over l variables, the a 's and b 's are integers, and $\sim \in \{=, \neq, >, <, \geq, \leq\}$. Under this restriction, the k -accumulated weight problem is decidable.

Theorem 5. *For each k , the k -accumulated weight problem, in which (4) is a disjunction of formulas in the form of (6), is decidable.*

Currently we do not know whether Theorem 5 still holds if (6) is conjuncted with one additional inequality: $a'_1 W_1(\alpha) + \dots + a'_k W_k(\alpha) + b'_1 w_1 + \dots + b'_l w_l \sim a'_0$.

As in the statement of the problem at the beginning of this section, a weight measure assigns numbers $w_1, \dots, w_l$ to symbols $a_1, \dots, a_l$ respectively. Instead of a fixed one, suppose that the weight of a symbol a_i can take any value between a given number q_i and w_i . That is, the weight measure defines a possible weight

range that a symbol can have, with the given number q_i being the lowest possible weight. Thus, in contrast to (3), $W_i(\alpha)$, $1 \leq i \leq l$, will be a set:

$$\{\hat{W}_i : \sum_{a_j \in \Sigma_i} q_j \cdot \phi_{a_j}(\alpha) \leq \hat{W}_i \leq \sum_{a_j \in \Sigma_i} w_j \cdot \phi_{a_j}(\alpha)\}. \quad (7)$$

For instance, suppose $\Sigma_1 = \{a_1\}$, $q_1 = 2$, $w_1 = 7$, and a word $\alpha = a_1 a_1 a_1$. Clearly, 12 is a weight in $W_1(\alpha)$ according to (7).

With the new definition of $W_i(\alpha)$, the following *loose k -accumulated weight problem* can be formulated:

- **Given:** An effectively semilinear language L , numbers $q_1, \dots, q_l \in \mathbf{N}$, k subsets $\Sigma_1, \dots, \Sigma_k$ of Σ , and a Presburger formula P over $l + k$ variables.
- **Question:** Is there a word α in L such that, for some $w_1, \dots, w_l \in \mathbf{N}$, and for some $\hat{W}_1, \dots, \hat{W}_k$,

$$\hat{W}_1 \in W_1(\alpha) \wedge \dots \wedge \hat{W}_k \in W_k(\alpha) \wedge P(w_1, \dots, w_l, \hat{W}_1, \dots, \hat{W}_k) \quad (8)$$

holds?

Notice that the lower weight bounds $q_1, \dots, q_l$ are in the **Given**-part, hence they are constants; while the upper bounds $w_1, \dots, w_l$ in the **Question**-part, are essentially unspecified parameters. (Otherwise, if the lower bounds $q_1, \dots, q_l$ are moved into the **Question**-part; i.e., both the lower and the upper bounds are parameterized constants, then the k -accumulated weight problem is a special case of the loose k -accumulated weight problem under this definition, by letting the lower bound and the upper bound be the same parameterized constant for each activity.)

The following result shows that the loose k -accumulated weight problem is decidable for each k . It is in contrast to Theorem 3 that the k -accumulated weight problem is undecidable for some large k .

Theorem 6. *For each k , the loose k -accumulated weight problem is decidable.*

4 Applications

In this section, we will apply the results presented in the previous section to some verification problems concerning infinite systems containing parameterized constants. We start with a general definition.

A transition system M can be described as a relation $T \subseteq S \times \Gamma^* \times \Sigma \times S \times \Gamma^*$, where S is a finite set of states, Γ is the configuration alphabet, and Σ is the activity alphabet. Obviously, we always assume that M can be effectively described; i.e., T is recursive. A configuration $\langle s, \beta \rangle$ of M is a pair of a state s in S and a word β in Γ^* . In the description of M , an initial configuration is also designated. According to the definition of T , an activity in Σ transforms one configuration to another. More precisely, we write $\langle s, \beta \rangle \xrightarrow{a} \langle s', \beta' \rangle$ if $T(s, \beta, a, s', \beta')$.

Let $\alpha \in \Sigma^*$ with $\alpha = a^1 \dots a^m$ for some m . We say that $\langle s, \beta, \alpha \rangle$ is *reachable* if, for some configurations $\langle s_0, \beta_0 \rangle, \dots, \langle s_m, \beta_m \rangle$, the following is satisfied

$$\langle s_0, \beta_0 \rangle \xrightarrow{a^1} \dots \xrightarrow{a^m} \langle s_m, \beta_m \rangle, \quad (9)$$

where $\langle s_0, \beta_0 \rangle$ is the initial configuration, $s_m = s$ and $\beta_m = \beta$. We use L_s to denote the set $\{(\beta, \alpha) : \langle s, \beta, \alpha \rangle \text{ is reachable}\}$. M is a *semilinear system* if L_s is an effectively semilinear language for each $s \in S$ (i.e., the semilinear set of L_s is computable from the description of M). As before, we use $w_1, \dots, w_l$ to denote a weight measure of $\Sigma = \{a_1, \dots, a_l\}$, and use $\Sigma_1, \dots, \Sigma_k$ to denote k subsets of Σ . We may introduce *weight counters* $W_1, \dots, W_k$ into M to indicate that the accumulated weight on each Σ_i is incremented by w_i whenever an activity $a_j \in \Sigma_i$ is performed. That is, on a transition $\langle s, \beta \rangle \xrightarrow{a_j} \langle s', \beta' \rangle$ in M , the counters are updated as follows, for each $1 \leq i \leq k$, if $a_j \in \Sigma_i$ then $W_i := W_i + w_j$ else $W_i := W_i$. Similarly, for a loose weight measure $(q_1, w_1), \dots, (q_l, w_l)$, the counters are updated on the transition as follows: for each $1 \leq i \leq k$, if $a_j \in \Sigma_i$ then $W_i := W_i + p_j$ else $W_i := W_i$, for some $q_j \leq p_j \leq w_j$ (i.e., p_j is non-deterministically chosen between q_j and w_j). Starting with 0, the weight counters are updated along an execution path in (9). We say that $\langle s, \beta, \alpha, W_1, \dots, W_k \rangle$ is *reachable* (under the weight measure $w_1, \dots, w_l$) if the weight counters have values $W_1, \dots, W_k$ at the end of an execution path in (9) witnessing that $\langle s, \beta, \alpha \rangle$ is reachable.

Let $y_1, \dots, y_u$ and $z_1, \dots, z_v$ be distinct variables. A (u, v) -formula, denoted by $P([y_1, \dots, y_u]; [z_1, \dots, z_v])$, is a Presburger formula that is a Boolean combination (using $\wedge$ and $\neg$) of Presburger formulas over $y_1, \dots, y_u$ and Presburger formulas over $z_1, \dots, z_v$. For the M specified in above, we let $u = |\Gamma| + l$ and $v = l + k$. Now, we consider the *k-reachability problem* for M : given a state s and a (u, v) -formula P , are there $w_1, \dots, w_l \in \mathbf{N}$ such that

$$P([\phi(\alpha), \phi(\beta)]; [w_1, \dots, w_l, W_1, \dots, W_k]) \quad (10)$$

holds for some reachable $\langle s, \beta, \alpha, W_1, \dots, W_k \rangle$ (under the weight measure $w_1, \dots, w_l$)? The *loose k-reachability problem* for M can be defined similarly where the lower weights $q_1, \dots, q_l$ are given. Directly from Theorems 4, 5 and 6, one can show the following results.

Theorem 7. *The 2-reachability problem is decidable for semilinear systems.*

Theorem 8. *For each k , the k -reachability problem is decidable for semilinear systems, when P in (10) is a disjunction of formulas in the following form:*

$$Q([\phi(\alpha), \phi(\beta)]; [w_1, \dots, w_l]) \wedge c_1 W_1 + \dots + c_k W_k + d_1 w_1 + \dots + d_l w_l \sim c_0,$$

where Q is a (u, l) -formula, the c 's and d 's are integers, and $\sim \in \{=, \neq, >, <, \geq, \leq\}$.

Theorem 9. *For each k , the loose k -reachability problem for semilinear systems is decidable.*

Many machine models are semilinear systems. We start with a simple model. Consider a nondeterministic finite state machine M , which is specified in Section 1 with a designated initial state. Notice that, in this case, $\Gamma = \emptyset$. Let s be a state. Clearly, L_s , the set of all the activity sequences when M moves from the initial state to s is a regular (and hence semilinear) language. Therefore, Theorems 7 and 8 hold for such M . Conversely, for any semilinear language L , one can construct, from the semilinear set of L , a regular language whose semilinear set is the same as the semilinear set of L [18]. From the regular language, one can easily construct a M and a state s such that the regular language is exactly L_s . It is routine to establish the fact that the k -reachability problem is decidable (for the M) iff the k -accumulated weight problem is decidable (for the L). From Theorem 3, one can show

Theorem 10. *There is a fixed k such that the k -reachability problem is undecidable for finite state machines M .*

In the definition of the k -reachability problem, the Presburger formula P in (10) is to specify the undesired values for the w 's and the W 's. When M is understood as a design of some system, a positive answer to the instance of the k -reachability problem indicates a design bug. In software engineering, it is highly desirable that a design bug is found as early as possible, since it is very costly to fix a bug once a system has already been implemented. It is noticed that in a specific implementation of the design, the parameterized constants are concrete, though the values differ from one implementation to another. Of course, one may test the specification by plugging in a particular choice for the concrete values. However, it is important to guarantee that for *any* concrete values for the parameterized constants, the design M is bug-free.

For instance, consider again the packet-based network switch example, where as we mentioned in Section 1, the switch is modeled as a finite state machine. Suppose the scheduling discipline is required to achieve such fairness property that no matter how the weights are assigned, the total packets serviced by o_1 must be greater than that of o_2 only if the summation of weights of connections in C_1 is greater than that of C_2 (we assume that all connections are nonempty at any time); i.e.,

$$\sum_{c_i \in C_1} w_i - \sum_{c_i \in C_2} w_i \geq 0 \rightarrow W_1 - W_2 \geq 0.$$

From Theorem 7, we know this fairness property can be automatically verified. When there are k servers involved in the example switch, a fairness property can be similarly formulated as a conjunction of the fairness between any two servers. In this case, the fairness property over k -servers is hard to be automatically verified, because of Theorem 10.

One may consider other variations on the model of M . For instance, an activity a_i is associated with, instead of one parameterized weight w_i , but two

(or any fixed number of) parameterized weights w_i^1 and w_i^2 , from which an instance of the activity can nondeterministically choose during execution. But this variation does not increase the expressive power of M , since “performing activity a_i ” can be simulated by “performing activity a_i^1 ” or “performing activity a_i^2 ” (nondeterministically chosen) where activity a_i^1 (resp. a_i^2) has weight w_i^1 (resp. w_i^2). One may consider another variation on the model of M where an instance of activity a_i has a weight nondeterministically chosen in between some given number (such as 2) in $\mathbf{N}$ and a parameterized constant w_i . Clearly, from Theorem 9, the loose k -reachability problem is decidable for this model of M .

M can be further generalized; e.g., M is augmented with a pushdown stack. Each transition in M now is in the following form: $s \rightarrow^{a,b,\gamma} s'$, indicating that M moves from state s to state s' while performing an activity a and also updating the stack (replacing the top symbol b in the stack by a stack word γ). There are only finitely many transitions in the description of M . Initially, the stack contains a designated initial symbol (i.e., an *initialized stack*) and the machines stays at an initial state. Notice that, for this model of M , L_s is a permutation of a context-free (hence semilinear) language. Therefore, M is still a semilinear system. The results of Theorems 7, 8 and 9 apply for pushdown systems.

M can be further augmented with a finite number of reversal-bounded counters. A nonnegative integer counter is reversal-bounded [11] if it alternates between a nondecreasing mode and a nonincreasing mode (and vice versa) for a given finite number of times, independent of the computations. Hence, a transition in M , in addition to the stack operation, can increment/decrement a counter by one and test a counter against zero. When the counter values are encoded as unary strings, it is not hard to show that the language of L_s is a semilinear language [11]. Hence, this model of M is still a semilinear system, and hence, Theorems 7, 8 and 9 can be applied.

M can be further generalized by adding a number of dense clocks. A clock is a nonnegative real variable. Clock behavior in M includes progresses and resets. A clock progress makes all the clocks advance with the same rate for a nondeterministically chosen amount in positive reals. A clock reset brings a clock value to 0 while keeping all the other clocks unchanged. In M , a transition is either a *stay transition* or a *reset transition*. A stay transition makes M stay in the current state and not perform any stack and counter operations, but the clocks progress. A reset transition makes M move from a state to another while performing an activity, a stack operation, and/or a counter operation. In addition, the transition resets some clocks. A *clock constraint* is a Boolean combination of formulas $x \sim c$ and $x - y \sim c$ where x, y are clocks, and c is an integer, $\sim \in \{>, <, =, \geq, \leq\}$. A (stay/reset) transition in M is also associated with a clock constraint that must be satisfied in order for the transition to fire. The reader may have already noticed that, when M does not have reversal-bounded counters and the pushdown stack, and when each activity is understood as an “input symbol”, M is simply equivalent to a timed automaton [2] that has been well studied in recent years for modeling and verifying real-time systems (see [1] for a survey). Here in this paper, an activity on a transition in M is associated

with a weight. This weight can be understood as a special form of “prices” in the sense of [15] that tries to model some (e.g., linearly) time-dependent variables in a complex real-time systems. Though, in general, priced timed automata are undecidable for reachability [15], some restricted forms of prices should be decidable, as shown in below, when one understands a weight as a special form of prices.

Consider an execution of M that starts from the initial state and ends with state s . Initially, all the clocks and counters are 0 and the stack is initialized. At the end of the execution, we require that the clock values $(x_1, \dots, x_t)$, the counter values $(y_1, \dots, y_u)$, and the stack content (γ) satisfy a given formula $Q(x_1, \dots, x_t, y_1, \dots, y_u, z_1, \dots, z_m)$ where z_i is the number of occurrences of stack symbol b_i in stack word γ . The form of the formula Q is a Boolean combination of $l(x_1, \dots, x_t, y_1, \dots, y_u, z_1, \dots, z_m) \sim 0$ where l is a linear polynomial and $\sim \in \{>, <, =, \geq, \leq\}$. Notice that Q contains both dense variables and discrete variables. Here, we use L to denote the set of all activity sequences on all such executions. If M does not have counters and the stack, L is a regular language and Q is a clock constraint (i.e., as we defined earlier, comparing one clock or the difference of two clocks against a constant). The regularity can be shown using the classic region technique in [2]. In general, however, L is not regular. Using the main theorem in [9], one can show that L can be accepted by a nondeterministic pushdown automaton with reversal-bounded counters. Hence, L is still a semi-linear language according to [11]. Associating an activity with a parameterized constant, one can formulate a k -reachability problem for M (similar to (10)): Is there an execution of M from the initial state to state s such that, at the end of the execution, the parameterized constants $w_1, \dots, w_l$, the accumulated weights $W_1, \dots, W_k$, the clocks values $x_1, \dots, x_t$, the counter values $y_1, \dots, y_u$, and the stack word counts $z_1, \dots, z_m$, satisfy

$$P(w_1, \dots, w_l, W_1, \dots, W_k) \wedge Q(x_1, \dots, x_t, y_1, \dots, y_u, z_1, \dots, z_m)?$$

Following the same proof ideas, one can show that the results of Theorems 7, 8 and 9 still hold for the M augmented with dense clocks, a pushdown stack and reversal-bounded counters.

As a final example, we use the decidability of 2-systems to strengthen recent results in [12]. Consider the model of a two-way deterministic finite automaton augmented with monotonic (i.e., nondecreasing) counters $C_1, \dots, C_k$ operating on an input of the form $a_1^{i_1} \dots a_n^{i_n}$ (for some fixed n), with left and right endmarkers. M starts in its initial state on the left end of the input with all counters initially zero. At each step, a counter can be incremented by 0 or 1, but the counters do not participate in the dynamics of the machine. An m -equality relation E over the counter values is a conjunction of m atomic relations of the form $c_i = c_j$. The m -equality relation problem is that of deciding, given a machine M , a state q , and an m -equality relation E , whether there is $(i_1, \dots, i_n)$ such that M , on input $a_1^{i_1} \dots a_n^{i_n}$, reaches some configuration where the state is q and the counter values satisfy E . Note that in dealing with the m -equality relation problem, we need only consider machines with at most $2m$ monotonic counters. It is open

whether the m -equality relation problem is decidable. However, when $m = 1$, it was recently shown in [12] that the 1-equality relation problem is decidable. The proof of the decidability for $m = 1$ in [12] does not generalize to the case when the two counter values must satisfy an arbitrary Presburger formula E . We give a proof of this generalization below.

First we generalize the m -equality relation problem by allowing E to be an arbitrary Presburger relation $E(c_1, \dots, c_k)$ over the counter values $c_1, \dots, c_k$. Call this the *Presburger relation problem*. Note that the m -equality relation problem is a very special case of the Presburger relation problem. We can use the decidability of 2-systems to show that the Presburger relation problem for machines with only 2 monotonic counters is decidable. The idea is as follows. In [12], it was shown that the values c_1 and c_2 of the two counters at any time can effectively be represented by equations of the form:

$$c_1 = A_1 + yB_1 + C_1, c_2 = A_2 + yB_2 + C_2,$$

where y is a nonnegative integer variable, and $A_1, B_1, C_1, A_2, B_2, C_2$ are nonnegative linear polynomials in some nonnegative integer variables $x_1, \dots, x_m$. (Even though C_1 and C_2 can be absorbed by A_1 and A_2 , we use the formulation above to be consistent with the formulation in [12].) Since E (subset of $\mathbb{N}^2$) is Presburger, it is semilinear. First assume that E is a linear set. Then the two components of E can be represented by nonnegative linear polynomials $p_1(z_1, \dots, z_r)$ and $p_2(z_1, \dots, z_r)$ for some nonnegative integer variables $z_1, \dots, z_r$. Thus, using the two equations above, we get: $A_1 + yB_1 + C_1 = p_1(z_1, \dots, z_r)$, $A_2 + yB_2 + C_2 = p_2(z_1, \dots, z_r)$. Rearranging terms, these two equations can be written as: $yB_1 = p_1 - A_1 - C_1$ and $yB_2 = p_2 - A_2 - C_2$. By semilinear transformation, we can reduce these equations to $yB_1 = D_1$ and $yB_2 = D_2$, where B_1, B_2, D_1, D_2 are nonnegative linear polynomials in some nonnegative integer variables $w_1, \dots, w_t$. Since the above equations constitute a 2-system, it is solvable in $y, w_1, \dots, w_t$. When E is a semilinear set, we just need to check if at least one of a finite number of equations of the form above has a solution.

It is open whether the Presburger relation problem is decidable when there are more than 2 monotonic counters (since the m -equality relation problem, which is a special case, is open). But suppose the Presburger relation E takes the following special form: $p_1(c_1, \dots, c_k) \sim d_1 \wedge p_2(c_1, \dots, c_k) \sim d_2 \wedge \dots \wedge p_m(c_1, \dots, c_k) \sim d_m$, where $d_1, \dots, d_m$ are integers (positive, negative, zero) and each $p_i(c_1, \dots, c_k)$ is a linear polynomial (not necessarily nonnegative), and each $\sim$ in $\{>, <, =, \geq, \leq\}$. It is easy to see that when $m = 2$, i.e., there are only two linear polynomials p_1 and p_2 involved in the conjunction above, then by adding “slack” variables and doing semilinear transformation, we can again reduce the problem to solving a system of the form: $yB_1 = D_1, yB_2 = D_2$, and, therefore, solvable. However, the case when $m > 2$ is open.

Acknowledgement. The authors would like to thank the anonymous referees for many valuable comments and suggestions.

References

1. R. Alur. Timed automata. In *CAV'99*, volume 1633 of *LNCS*, pages 8–22. Springer, 1999.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.
3. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model-checking. In *CONCUR'97*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.
4. T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results. *ACM Transactions on Programming Languages and Systems*, 21(4):747–789, July 1999.
5. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.
6. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV'98*, volume 1427 of *LNCS*, pages 268–279. Springer, 1998.
7. Z. Dang. Verifying and debugging real-time infinite state systems (PhD. Dissertation). *Department of Computer Science, University of California at Santa Barbara*, 2000.
8. Z. Dang, O. Ibarra, and Z. Sun. On the emptiness problems for two-way nondeterministic finite automata with one reversal-bounded counter. In *ISAAC'02*, volume 2518 of *LNCS*, pages 103–114. Springer, 2002.
9. Zhe Dang. Binary reachability analysis of pushdown timed automata with dense clocks. In *CAV'01*, volume 2102 of *LNCS*, pages 506–517. Springer, 2001.
10. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997. Special Issue: Formal Methods in Software Practice.
11. O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, January 1978.
12. O. H. Ibarra and Z. Dang. Deterministic two-way finite automata augmented with monotonic counters. 2002 (submitted).
13. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.
14. O. Kupferman and M.Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *CAV'00*, volume 1855 of *LNCS*, pages 36–52. Springer, 2000.
15. K. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *CAV'01*, volume 2102 of *LNCS*, pages 493–505. Springer, 2001.
16. Y. V. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.
17. M. Minsky. Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines. *Ann. of Math.*, 74:437–455, 1961.
18. R. Parikh. On context-free languages. *Journal of the ACM*, 13:570–581, 1966.
19. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *LICS'86*, pages 332–344. IEEE Computer Society Press, 1986.

Monadic Second-Order Logics with Cardinalities^{*}

Felix Klaedtke¹ and Harald Rueß²

¹ Albert-Ludwigs-Universität Freiburg, Germany

² SRI International, CA, USA

Abstract. We delimit the boundary between decidability versus undecidability of the weak monadic second-order logic of one successor (WS1S) extended with linear cardinality constraints of the form $|X_1| + \dots + |X_r| < |Y_1| + \dots + |Y_s|$, where the X_i s and Y_j s range over finite subsets of natural numbers. Our decidability and undecidability results are based on an extension of the classic logic-automata connection using a novel automaton model based on Parikh maps.

1 Introduction

In the automata-theoretic approach for solving the satisfiability problem of a logic one develops an appropriate notion of automata and establishes a translation from formulas to automata. The satisfiability problem for the logic then reduces to the automata emptiness problem. Most prominently, decidability of the (weak) monadic second-order logic of one successor (W)S1S is proved by a translation of formulas to word automata, see e.g. [27]. Despite the nonelementary worst-case complexity [19, 26], the automata-based decision procedure for WS1S, implemented in the Mona tool [10, 16], has been found to be effective for reasoning about a multitude of computation systems ranging from circuits [3, 2] to protocols [17, 25]. Furthermore, it has been integrated in theorem provers to decide well-defined fragments of higher-order logic [1, 21].

Many interesting verification problems, however, fall outside the scope of WS1S. For example, the verifications in WS1S for the sequential circuits considered in [3] are only with respect to concrete values of parameters such as setup time and minimum clock period since some linear arithmetic is used on these parameters. Also, certain distributed algorithms such as the Byzantine generals problem [18] of reaching distributed consensus in the presence of unreliable messengers and treacherous generals cannot be modeled in WS1S, since reasoning about the combination of (finite) sets and cardinality constraints on these sets is required here.

In order to support this kind of reasoning and to significantly extend the range of automated verification procedures we extend WS1S with atomic formulas of the form $|X_1| + \dots + |X_r| < |Y_1| + \dots + |Y_s|$, where the X_i s and Y_j s are

^{*} This work was supported by SRI International internal research and development, and NASA through contract NAS1-00079.

monadic second-order (MSO) variables, and $|X|$ denotes the cardinality of the MSO variable X . The extension of WS1S with cardinality constraints is denoted by $\text{WS1S}^{\text{card}}$. Our main results are

- (i) $\text{WS1S}^{\text{card}}$ is undecidable. More precisely, (a) the fragment of $\text{WS1S}^{\text{card}}$ consisting of the sentences of the form $\forall X \exists \bar{Y} \varphi$ is undecidable, where X is an MSO variable, $\bar{Y}$ is a vector of MSO variables ranging over finite sets of natural numbers, and all quantifiers in φ are first-order, that is, ranging over natural numbers. And, (b) the fragment of $\text{WS1S}^{\text{card}}$ consisting of the sentences of the form $\exists X \forall y \exists \bar{Y} \varphi$, where y is a first-order variable and all quantifiers in φ are first-order.
- (ii) The fragment consisting of the sentences of the form $Q_1 x_1 \dots Q_\ell x_\ell Q \bar{Y} \varphi$ is decidable, where $Q_1, \dots, Q_\ell \in \{\exists, \forall\}$ are first-order quantifiers and an MSO variable occurring in a cardinality constraint in φ is bound by $Q \in \{\exists, \forall\}$.

Together the results (i) and (ii) delimit the boundary between decidability versus undecidability of MSO logics with cardinality constraints.

We use an automata-theoretic approach for obtaining these results by defining a suitable extension of finite word automata. These extensions work over an extended alphabet in which a vector of natural numbers is attached to each letter of the input alphabet. An input is accepted if an input word is accepted in the traditional sense *and* if a projection of the word via a monoid homomorphism to a vector of natural numbers satisfies given arithmetic constraints. Since this monoid homomorphism generalizes Parikh's *commutative image* [23] on words, we call such an extended automaton a *Parikh finite word automaton* (PFWA). PFWAs characterize the expressiveness of the existential fragment of $\text{WS1S}^{\text{card}}$.

The undecidability results (i) follow from the undecidability of the universality problem for PFWAs and the undecidability of the halting problem for 2-register machines, whereas the decidability result (ii) is based on a two-step construction. First, we build a PFWA for the formula $\exists \bar{Y} \varphi$, and, second, we transform the PFWA into a corresponding Presburger arithmetic formula. This latter construction takes care of the quantification of the first-order variables $x_1, \dots, x_\ell$. Compared to simply checking the emptiness problem for the PFWA associated with a formula, this two-stage translation yields a decision procedure for a much more expressive fragment of $\text{WS1S}^{\text{card}}$. These constructions can readily be extended to obtain corresponding results for cardinality constraints in second-order monadic logics over trees [15].

The paper is structured as follows. In §2 we introduce PFWAs. Then, in §3 we define $\text{WS1S}^{\text{card}}$ and compare the expressiveness of the existential fragment of $\text{WS1S}^{\text{card}}$ with PFWAs. In §4 we prove the results (i) and (ii), and illustrate applications of the decidability result (ii). Finally, in §5 we draw conclusions.

2 Parikh Automata

We introduce a framework that extends the acceptance condition of machines operating on words. In addition to the traditional acceptance condition of a

machine, we require that an input satisfies arithmetic properties, where the input is associated with a vector of natural numbers. Parikh finite word automata are an instance of this framework.

Let $\Sigma = \{b_1, \dots, b_n\}$ be a linearly ordered alphabet. Parikh's [23] *commutative image* $\Phi : \Sigma^* \rightarrow \mathbb{N}^{|\Sigma|}$ maps the elements of the free monoid Σ^* , so-called words, to vectors of natural numbers. The commutative image is defined by $\Phi(b_i) := \bar{e}_i$ and $\Phi(uv) := \Phi(u) + \Phi(v)$, where $\bar{e}_i \in \mathbb{N}^{|\Sigma|}$ is the unit vector with the i th coordinate equal to 1 and all other coordinates are 0. Intuitively, the i th position of $\Phi(w)$ counts how often b_i occurs in $w \in \Sigma^*$. We extend Parikh's commutative image by considering the Cartesian product of Σ and a nonempty set D of vectors of natural numbers.

Definition 1. Let Γ be an alphabet of the form $\Sigma \times D$, where D is a nonempty subset of $\mathbb{N}^N$, for some $N \geq 1$. We define the **projection** $\Psi : \Gamma^* \rightarrow \Sigma^*$ and the **extended Parikh map** $\Phi : \Gamma^* \rightarrow \mathbb{N}^N$ as monoid homomorphisms.

- (i) $\Psi(b, \bar{d}) := b$, for $(b, \bar{d}) \in \Sigma \times D$, and $\Psi(uv) := \Psi(u)\Psi(v)$.
- (ii) $\Phi(b, \bar{d}) := \bar{d}$, for $(b, \bar{d}) \in \Sigma \times D$, and $\Phi(uv) := \Phi(u) + \Phi(v)$.

Note that if we attach to each letter $b_i \in \Sigma$ the unit vector $\bar{e}_i \in \mathbb{N}^{|\Sigma|}$ in a word $w \in \Sigma^*$ then the extended Parikh map yields the commutative image of w .

We constrain a language by an arithmetic property given by a set of vectors of natural numbers.

Definition 2. For a language $L \subseteq (\Sigma \times D)^*$ and $C \subseteq \mathbb{N}^N$, let

$$L \upharpoonright_C := \{\Psi(w) \mid w \in L \text{ and } \Phi(w) \in C\}$$

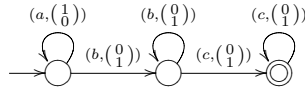
be the **restriction** of L with respect to C .

The acceptance condition of a machine operating on words can be extended in the following way. A word w over Σ is accepted if the machine accepts a word over $\Sigma \times D$ both in the traditional sense and if the sum of the attached vectors to the symbols in w is in a given subset of $\mathbb{N}^N$. Here, we are mainly concerned with finite word automata and arithmetic constraints restricted to *semilinear sets* $U \subseteq \mathbb{N}^s$. This means that there are linear polynomials $p_1, \dots, p_m : \mathbb{N}^r \rightarrow \mathbb{N}^s$ such that U is the union of the images of these polynomials, that is, $U = \bigcup_{1 \leq i \leq m} \{p_i(x_1, \dots, x_r) \mid x_1, \dots, x_r \in \mathbb{N}\}$.

Definition 3. A **Parikh finite word automaton (PFWA)** of dimension $N \geq 1$ is a pair $(\mathcal{A}, C)$, where $\mathcal{A}$ is a finite word automaton with an alphabet of the form $\Sigma \times D$, D is a finite, nonempty subset of $\mathbb{N}^N$, and C is a semilinear subset of $\mathbb{N}^N$. The PFWA $(\mathcal{A}, C)$ **recognizes** the language $L(\mathcal{A}, C) := L(\mathcal{A}) \upharpoonright_C$, where $L(\mathcal{A})$ is the language recognized by $\mathcal{A}$.

The PFWA $(\mathcal{A}, C)$ is **deterministic** if for the transition function δ of $\mathcal{A}$ it holds that for every state q and for every $(b, \bar{d}) \in \Sigma \times D$, $|\delta(q, (b, \bar{d}))| \leq 1$, and if $|\delta(q, (b, \bar{d}))| = 1$ then $|\delta(q, (b, \bar{d}'))| = 0$, for every $\bar{d} \neq \bar{d}'$.

For example, the deterministic PFWA $(\mathcal{A}, \{(\bar{z}) \mid z \in \mathbb{N}\})$, where $\mathcal{A}$ is given by the picture



recognizes $\{a^{i+j}b^ic^j \mid i, j > 0\}$, which is context-sensitive but not context-free.

PFWAs are strictly more expressive than finite word automata, where the accepted words are constrained by their commutative images and semilinear sets. It is easy to define a deterministic PFWA automaton that recognizes the language $L := \{a^ib^ja^ib^j \mid i, j \geq 1\}$. But there does not exist a finite word automaton $\mathcal{A}$ with the alphabet $\{a, b\}$ and a set $C \subseteq \mathbb{N}^2$ such that $w \in L$ iff $w \in L(\mathcal{A})$ and $(k, \ell) \in C$, where (k, ℓ) is the commutative image of w .

A PFWA can be seen as a finite word automaton extended with counters, where a vector of natural numbers attached to a symbol is interpreted as an increment of the counters. In contrast to other counter automaton models in the literature, for example [4, 7, 11], we do not restrict the applicability of transitions in a run by additional guards on the values of the counters. Instead, a PFWA constrains the language of a finite word automaton over the extended alphabet by a semilinear set. It turns out (a) that PFWAs are equivalent to reversal-bounded multicounter machines [11, 12] and (b) that PFWAs are equivalent to weighted finite automata over the groups $(\mathbb{Z}^k, +, 0)$ [6, 20] with $k \geq 1$ in the sense that all these three different kinds of machines describe the same class of languages. We refer the reader to [15], for definitions and a detailed comparison of these automaton models, proofs of the equivalences, and a comparison of PFWAs to other automaton models.

We state some properties of PFWAs. The details can be found in [15].

- Property 4.** (1) *Deterministic PFWAs are closed under union, intersection, complement, and inverse homomorphisms, but not under homomorphisms.*
 (2) *PFWAs are closed under union, intersection, homomorphisms, inverse homomorphisms, concatenations, and left and right quotients, but not under complement.*

The decidability of the emptiness problem relies on Parikh’s result [23], which states that the commutative image of a context-free language is semilinear.

Lemma 5. *Let Γ be an alphabet of the form $\Sigma \times D$ with $D \subseteq \mathbb{N}^N$, for some $N \geq 1$. For every context-free language $L \subseteq \Gamma^*$, there are linear polynomials $q_1, \dots, q_m : \mathbb{N}^r \rightarrow \mathbb{N}^N$, for some $r \geq 1$, such that*

$$\Phi(L) = \bigcup_{1 \leq i \leq m} \{q_i(x_1, \dots, x_r) \mid x_1, \dots, x_r \in \mathbb{N}\},$$

where Φ is the extended Parikh map of Γ . Moreover, the polynomials $q_1, \dots, q_m$ are effectively constructible if L is given by a pushdown automaton.

For a PFWA $(\mathcal{A}, C)$, we know by Lemma 5 that the set $\Phi(L(\mathcal{A}))$ is semilinear and effectively constructible. The decidability of the emptiness problem follows from the facts that semilinear sets are effectively closed under intersection, and $L(\mathcal{A}, C) \neq \emptyset$ iff $\Phi(L(\mathcal{A})) \cap C \neq \emptyset$.

Property 6. *The emptiness problem for PFWAs is decidable.*

The undecidability of the universality problem for PFWAs can be shown by reduction from the word problem for Turing machines.

Property 7. *The universality problem for PFWAs is undecidable.*

Note that the universality problem for deterministic PFWAs is decidable since they are closed under complement and the emptiness problem is decidable.

3 WS1S with Cardinality Constraints

We extend WS1S in order to compare cardinalities of sets. We call this extension $\text{WS1S}^{\text{card}}$. The classic logic-automata connection of finite word automata and WS1S extends to PFWAs and the existential fragment of $\text{WS1S}^{\text{card}}$.

The Weak Monadic Second-Order Logic of One Successor. The atomic formulas of WS1S are membership Xx , and the successor relation $\text{succ}(x, y)$, where x, y are first-order (FO) variables, and X is a monadic second-order (MSO) variable. We adopt the following notation: lowercase letters $x, y, \dots$ denote FO variables and uppercase letters $X, Y, \dots$ denote MSO variables. Moreover, $\alpha, \beta, \dots$ range over FO and MSO variables. Formulas are built from the atomic formulas and the connectives $\neg$ and $\vee$, and the existential quantifier $\exists$ for FO and MSO variables. We also use the connectives $\wedge$, $\rightarrow$ and $\leftrightarrow$, and the universal quantifiers $\forall$ for FO and MSO variables, and we use the standard conventions for omitting parentheses. A formula is **existential** if it is of the form $\exists \bar{X} \varphi$ where all bound variables in φ are FO.

Formulas are interpreted over the natural numbers with the successor relation, that is, the structure $(\mathbb{N}, \text{succ})$. An interpretation I maps FO variables to natural numbers and MSO variables are mapped to *finite* subsets of $\mathbb{N}$. The truth value of a formula φ in $(\mathbb{N}, \text{succ})$ with respect to an interpretation I , in symbols $(\mathbb{N}, \text{succ}), I \models \varphi$, is defined in the obvious way. Note that existential quantification for MSO variables only ranges over *finite* subsets of $\mathbb{N}$. We write $(\mathbb{N}, \text{succ}) \models \varphi$ if φ is a sentence, that is, φ does not have free variables.

Equality $x = y$ can be expressed by $\exists z (\text{succ}(x, z) \wedge \text{succ}(y, z))$. For a natural number $t \in \mathbb{N}$, we write $x = t$ for $\exists z_0 \dots \exists z_t (x = z_t \wedge \bigwedge_{0 \leq i < t} \text{succ}(z_i, z_{i+1}) \wedge \forall y \neg \text{succ}(y, z_0))$, and we write $Yx + 1$ for $\exists z (\text{succ}(x, z) \wedge Yz)$. The formula $\text{part}(U, X_1, \dots, X_k)$ expresses that $X_1, \dots, X_k$ is a partition of U , that is $\forall y (Uy \leftrightarrow \bigvee_{1 \leq i \leq k} X_i y) \wedge \neg \exists y (\bigvee_{1 \leq i < j \leq k} (X_i y \wedge X_j y))$. Note that in these formulas only FO variables are quantified.

A word $w = b_1 \dots b_n \in (\{0, 1\}^k)^*$ determines the interpretation I_w for variables $\alpha_1, \dots, \alpha_k$, where for all $1 \leq j \leq k$,

- $i \in I_w(\alpha_j)$ iff $\chi_j(b_i) = 1$, if α_j is an MSO variable, and
- $i = I_w(\alpha_j)$ iff $\chi_j(b_i) = 1$ and $\chi_j(b_{i'}) = 0$ for all $i' \neq i$, if α_i is an FO variable,

where χ_j projects a vector in $\{0, 1\}^k$ to its j th coordinate. We extend χ_j homomorphically to words. Note that we have implicitly assumed that the variables

$\alpha_1, \dots, \alpha_k$ are ordered in the sense that the interpretation I_w of the variable α_j is determined by the j th projection of $b_1 \dots b_n$, that is, $\chi_j(b_1 \dots b_n)$. In the following, we write χ_{α_j} for χ_j . For a formula $\varphi(\alpha_1, \dots, \alpha_k)$, we define

$$L(\varphi) := \{w \in (\{0, 1\}^k)^* \mid (\mathbb{N}, \text{succ}), I_w \models \varphi\}.$$

Later, we shall need the following facts that are due to Büchi, Elgot, and Trakhtenbrot. For more details, see, for example, [27].

Fact 8. $L(\varphi)$ is regular for every WS1S formula φ . Moreover, we can effectively build a finite word automaton recognizing $L(\varphi)$.

For the other direction, that is, describing regular languages by WS1S formulas, there is a subtlety that we want to point out. Note that natural numbers and finite subsets of $\mathbb{N}$ have several encodings, e.g., all the words in $\{0\}^*$ encode the empty set. It is easy to see that languages definable by WS1S formulas, are closed under $\bar{0}$ -padding and $\bar{0}$ -cutting, that is, $w \in L(\varphi)$ iff $w\bar{0} \in L(\varphi)$, where $\bar{0}$ is the letter $(0, \dots, 0)$. We call a $\bar{0}$ -padding and $\bar{0}$ -cutting closed language **$\bar{0}$ -closed**.

Fact 9. For every regular $\bar{0}$ -closed language $L \subseteq (\{0, 1\}^k)^*$ there is an existential WS1S formula $\varphi(X_1, \dots, X_k)$ such that $L(\varphi) = L$.

To obtain an equivalence of the logic and the regular languages, one has to look at finite *word models* [27]. The main difference is that the universe of a finite word model is not $\mathbb{N}$, but $\{0, \dots, n-1\}$ where n is given by the length of the word. The distinction between the different semantics is emphasized by using the name M2L(str) or MSO[+1] instead of WS1S. The results below carry over to finite word models. We use the WS1S semantics since it simplifies matters.

Cardinality Constraints. $\text{WS1S}^{\text{card}}$ has in addition to the atomic formulas of WS1S the atomic formulas of the form $|X_1| + \dots + |X_r| < |Y_1| + \dots + |Y_s|$, where the truth value with respect to an interpretation I is defined as

$$(\mathbb{N}, \text{succ}), I \models |X_1| + \dots + |X_r| < |Y_1| + \dots + |Y_s| \quad \text{iff} \quad \sum_{1 \leq i \leq r} |I(X_i)| < \sum_{1 \leq i \leq s} |I(Y_i)|.$$

Let $\mathbf{C}$ be the set of formulas of the form $|X_1| + \dots + |X_r| < |Y_1| + \dots + |Y_s|$ and their negations. We write formulas like $-2|X| = 3|Y| + |Z|$ which can be transformed to an equivalent Boolean combination of formulas in $\mathbf{C}$ by standard arithmetic. Moreover, we also use the summation symbol $\sum$ for a shorter representation.

Parikh Automata and $\text{WS1S}^{\text{card}}$. We carry over the Facts 8 and 9 to the existential fragment of $\text{WS1S}^{\text{card}}$ and PFWAs. We start with the direction in Fact 9.

Theorem 10. *For every PFWA $(\mathcal{A}, C)$ where $L(\mathcal{A}, C) \subseteq (\{0, 1\}^s)^*$ is $\bar{0}$ -closed, there is an existential $\text{WS1S}^{\text{card}}$ formula $\psi_{\mathcal{A}, C}(U_1, \dots, U_s)$ with $L(\psi) = L(\mathcal{A}, C)$.*

Proof. Let $N \geq 1$ be the dimension of $(\mathcal{A}, C)$, and let $\mathcal{A} = (Q, \{0, 1\}^s \times D, \delta, q_1, F)$. Without loss of generality, we assume that $Q = \{1, \dots, r\}$, for some $r \geq 1$. Let K be the maximal natural number occurring in a vector in D , that is $K := \max(\bigcup_{(d_1, \dots, d_N) \in D} \{d_1, \dots, d_N\})$.

Let $b_0 \dots b_{n-1} \in L(\mathcal{A}, C)$ with $n \geq 0$ and $b_{n-1} \neq \bar{0}$. The formula $\psi_{\mathcal{A}, C}$ describes the existence of an accepting run $\varrho = q_0 \dots q_{n+m} \in Q^*$ on a word $(b_0, \bar{d}_0) \dots (b_{n-1}, \bar{d}_{n-1})(\bar{0}, \bar{d}_n) \dots (\bar{0}, \bar{d}_{n+m-1}) \in (\{0, 1\}^s \times D)^*$, for some $m \geq 0$. Note that $L(\mathcal{A}, C)$ is $\bar{0}$ -closed. It holds, $q_0 = q_1$, $q_{i+1} \in \delta(q_i, (b_i, \bar{d}_i))$, for $0 \leq i < n + m$, and $q_{n+m} \in F$. We encode ϱ by pairwise disjoint sets $Y_1, \dots, Y_r \subseteq \{0, \dots, n + m\}$ such that Y_q contains those positions i with $q = q_i$. Moreover, we keep track of the numbers at the k th position of the vectors $\bar{d}_i$ with the sets $Z_k^0, \dots, Z_k^K \subseteq \{0, \dots, n + m\}$: it holds that $0 \in Z_k^0$ and $i \in Z_k^d$ iff the k th position of $\bar{d}_{i-1}$ is d , for $1 \leq i \leq n + m$. Therefore, the k th position of the vector $\bar{d}_0 + \dots + \bar{d}_{n+m-1}$ is $\sum_{0 \leq d \leq K} d |Z_k^d|$. We have to check $\bar{d}_0 + \dots + \bar{d}_{n+m-1} \in C$. Formally,

$$\begin{aligned} & \exists Y_1 \dots \exists Y_r \exists Z_1^0 \dots \exists Z_1^K \dots \exists Z_N^0 \dots \exists Z_N^K \exists U \left(\text{domain}(U, U_1, \dots, U_s) \wedge \right. \\ & \quad \text{part}(U, Y_1, \dots, Y_r) \wedge \bigwedge_{1 \leq i \leq N} \text{part}(U, Z_i^0, \dots, Z_i^K) \wedge \\ & \quad \forall x \left((x = 0 \rightarrow Y_{q_1} x \wedge \bigwedge_{1 \leq i \leq N} Z_i^0 x) \wedge \right. \\ & \quad \quad (Ux \rightarrow \bigvee_{q \in \delta(p, (b, (d_1, \dots, d_N)))} (Y_p x \wedge \text{letter}_b(x, U_1, \dots, U_s) \wedge \\ & \quad \quad \quad Y_q x + 1 \wedge \bigwedge_{1 \leq i \leq N} Z_i^{d_i} x + 1)) \wedge \\ & \quad \quad \left. (Ux \wedge \neg Ux + 1 \rightarrow (\bigvee_{q \in F} Y_q x)) \right) \wedge \\ & \quad \left. \psi_C(Z_1^0, \dots, Z_1^K, \dots, Z_N^0, \dots, Z_N^K) \right), \end{aligned}$$

where $\text{domain}(U, U_1, \dots, U_s)$ is the formula $\forall x (U_1 x \vee \dots \vee U_s x \rightarrow Ux + 1) \wedge \forall x (Ux + 1 \rightarrow Ux)$ and $\text{letter}_b(x, U_1, \dots, U_s) := (\bigwedge_{b_i=0} \neg U_i x) \wedge (\bigwedge_{b_i=1} U_i x)$, for $b = (b_1, \dots, b_s) \in \{0, 1\}^s$. It remains to define the formula ψ_C . Since C is semilinear we can assume that C is the union of the images of linear polynomials $p_1, \dots, p_\ell : \mathbb{N}^k \rightarrow \mathbb{N}^N$, for some $k \geq 1$. For $1 \leq i \leq \ell$, let

$$\begin{aligned} \psi_{p_i} := & \exists X_1 \dots \exists X_k \left(\bigwedge_{1 \leq j \leq N} \exists X (\forall y (Xy \leftrightarrow \bigvee_{0 \leq t < d_j^0} y = t) \wedge \right. \\ & \quad \left. \sum_{1 \leq d \leq K} d |Z_j^d| = |X| + d_1^j |X_1| + \dots + d_k^j |X_k|) \right), \end{aligned}$$

where we assume that $p_i(x_1, \dots, x_k)$ has the form $\bar{d}_0 + \bar{d}_1 x_1 + \dots + \bar{d}_k x_k$ with $\bar{d}_j = (d_j^1, \dots, d_j^N)$, for $0 \leq j \leq k$. Let $\psi_C := \psi_{p_1} \vee \dots \vee \psi_{p_\ell}$. The quantification of the MSO variables in ψ_C can be pulled out to obtain an existential formula. $\dashv$

We give a set of formulas describing languages recognizable by PFWAs. This carries over Fact 8 to PFWAs and the existential fragment of $\text{WS1S}^{\text{card}}$. Let E be the set of $\text{WS1S}^{\text{card}}$ formulas of the form $\exists X_1 \dots \exists X_n \varphi$, where all MSO variables occurring in a subformula in C of φ are either free or bound by one of the existential quantifiers for $X_1, \dots, X_n$. Note that an existential formula

is in E , but E contains formulas that are not existential. A formula in E can contain MSO variables Y that are universally quantified if Y does not occur in subformulas in C and the quantification of Y happens below the existential quantification of $X_1, \dots, X_n$.

Theorem 11. *For every $\varphi \in E$, we can construct a PFWA recognizing $L(\varphi)$.*

Proof (Sketch). We can assume that $\varphi \in E$ is of the form $\exists X_1 \dots \exists X_n (\bigvee_i \bigwedge_j \psi_{ij})$, where ψ_{ij} is either a WS1S formula or $\psi_{ij} \in C$. By Fact 8 we can construct a finite word automaton $\mathcal{A}_{ij}$ with $L(\mathcal{A}_{ij}) = L(\psi_{ij})$ if ψ_{ij} is a WS1S formula, and for $\psi_{ij} \in C$, it is straightforward to give a PFWA $(\mathcal{A}_{ij}, C_{ij})$ with $L(\mathcal{A}_{ij}, C_{ij}) = L(\psi_{ij})$. From Property 4 we can construct a PFWA recognizing $L(\varphi)$. $\dashv$

Theorems 10 and 11 together reveal the following equivalence.

Corollary 12. *For a $\bar{0}$ -closed language $L \subseteq (\{0, 1\}^s)^*$, the following two conditions are equivalent:*

- (i) *L is recognizable by a PFWA, that is, there is a PFWA $(\mathcal{A}, C)$ with $L(\mathcal{A}, C) = L$.*
- (ii) *L is definable in the existential fragment of $\text{WS1S}^{\text{card}}$, that is, there is an existential $\text{WS1S}^{\text{card}}$ formula φ with $L(\varphi) = L$.*

Another extension of the classical logic-automata connection with a similar flavor is given in [22] relating Petri net languages with the existential fragment of the MSO logic on words extended with partial orders $\leq_g$ and $=_g$ on subsets of $\{0, \dots, n-1\}$ defined as $X \leq_g Y$ iff $|X \cap \{0, \dots, m-1\}| \leq |Y \cap \{0, \dots, m-1\}|$, for all $m \leq n$, and $X =_g Y$ iff $X \leq_g Y$ and $|X| = |Y|$. [22] does not investigate decidability problems about this logic as we will do in the next section for $\text{WS1S}^{\text{card}}$.

We want to point out that there is also a relationship between $\text{WS1S}^{\text{card}}$ and Petri nets. Petri net reachability is expressible in $\text{WS1S}^{\text{card}}$ [14]. From this it is not difficult to see that ($\bar{0}$ -closed) Petri net languages can be described in $\text{WS1S}^{\text{card}}$. But the formulas for expressing the reachability problem (or describing Petri net languages) require a top-level quantification of the form $\exists x \exists \bar{X} \forall y \forall \bar{Y}$. In the next section we are going to show that the fragment with such a top-level quantification is undecidable. Note that the reachability problem for Petri nets is decidable.

4 Undecidability and Decidability Results

Decidability and undecidability results about MSO logics with cardinality constraints summarized in Figure 1 using the notation introduced below. Together, these results delimit the boundary between decidability versus undecidability in $\text{WS1S}^{\text{card}}$. Furthermore, we illustrate applications in hardware and software verification of a decidable fragment of $\text{WS1S}^{\text{card}}$.

We introduce the following notation to uniformly describe fragments of $\text{WS1S}^{\text{card}}$.

undecidable	$[\forall_{\text{MSO}} \exists_{\text{MSO}}^* \text{FO}; \text{succ}]$	$[\exists_{\text{MSO}} \forall_{\text{FO}} \exists_{\text{MSO}}^5 \text{FO}; \text{succ}]$
decidable	$[\text{FO}(\exists_{\text{MSO}}^* \cup \forall_{\text{MSO}}^*) \text{FO}; \text{RWS1S}]$	

Fig. 1. Undecidable and decidable fragments of $\text{WS1S}^{\text{card}}$.

Definition 13. Let $Q \subseteq \{\exists_{\text{MSO}}, \forall_{\text{MSO}}, \exists_{\text{FO}}, \forall_{\text{FO}}\}^*$ and let R be a set of relations over the natural numbers and finite subsets of natural numbers. We write $[Q; R]$ for the set of sentences of the form $Q_1 \alpha_1 \dots Q_n \alpha_n \varphi$, where $Q_1 \dots Q_n \in Q$ and φ is a quantifier-free formula with relations in R .

We write $[Q; R_1, \dots, R_n]$ for $[Q; \{R_1, \dots, R_n\}]$. Let R_{WS1S} be the set of relations that are definable in WS1S. We will often give the set Q as a regular expression. For example, the set FO of arbitrary FO quantifier prefixes is $(\exists_{\text{FO}} \cup \forall_{\text{FO}})^*$, and we write, e.g., $\exists_{\text{MSO}}^2$ for $\exists_{\text{MSO}} \exists_{\text{MSO}}$.

Undecidability Results.

Theorem 14. The fragment $[\forall_{\text{MSO}} \exists_{\text{MSO}}^* \text{FO}; \text{succ}]$ is undecidable.

Proof. To prove this theorem we look at the universality problem for $\bar{0}$ -closed PFWAs which is undecidable. This can be shown by adopting the proof that shows the undecidability for universality problem for PFWAs.

Let $(\mathcal{A}, C)$ be a $\bar{0}$ -closed PFWA with $L(\mathcal{A}, C) \subseteq \{0, 1\}^*$. For the formula $\psi_{\mathcal{A}, C}(U_1)$ from Theorem 10, it holds, $(\mathbb{N}, \text{succ}) \models \forall U_1 \psi_{\mathcal{A}, C}$ iff $L(\mathcal{A}, C) = \{0, 1\}^*$. Since $\psi_{\mathcal{A}, C}$ is existential and the universality problem is undecidable, we have that the fragment $[\forall_{\text{MSO}} \exists_{\text{MSO}}^* \text{FO}; \text{succ}]$ is undecidable. $\dashv$

Theorem 15. The fragment $[\exists_{\text{MSO}} \forall_{\text{FO}} \exists_{\text{MSO}}^5 \text{FO}; \text{succ}]$ is undecidable.

Proof (Sketch). The undecidability is shown by encoding the halting problem for 2-register machines as a formula in $[\exists_{\text{MSO}} \forall_{\text{FO}} \exists_{\text{MSO}}^5 \text{FO}; \text{succ}]$.

Let $\mathcal{C}$ be a 2-register machine. A computation of $\mathcal{C}$ can be encoded as a word $w \in \{0, 1\}^*$ in the following way. The word w consists of segments of the form $110b_1 \dots b_s 0z_0z_1z'_0z'_1$. The sequence $b_1 \dots b_s$ encodes the state, namely $b_q = 1$ iff the state of the configuration is q . The sequence $z_0z_1z'_0z'_1$ encodes the increment or decrement of a register: $z_i = 1$ iff the i th register is incremented, and $z'_i = 1$ iff the i th register is decremented. With the letters $110 \dots 0 \dots$ we can check whether a subword of w represents an encoding of a configuration.

We define a sentence of the form $\exists X \forall y \exists U \exists Z_0 \exists Z_1 \exists Z'_0 \exists Z'_1 \psi$, where ψ is FO. The details on this sentence are in [15]. Intuitively, X represents a word $w \in \{0, 1\}^*$ that is an encoding of a computation of $\mathcal{C}$, where w is a concatenation of sequences of the form $110 \dots 0 \dots$ as explained above. The FO variable y intuitively ranges over all the configurations in X , and the MSO variables Z_i, Z'_i take care of the increments and decrements of the i th register up to the y th configuration. Therefore, $|Z_i| - |Z'_i|$ is the value of the i th counter in the y th configuration. The MSO variable U is used for some technical reasons; it represents the set $\{0, \dots, y\}$. $\dashv$

Decidability Result. Since the emptiness problem for PFWAs is decidable and the construction of the PFWA in Theorem 11 for a given formula in $\mathbf{E}$ is constructive, we get a decision procedure for $\mathbf{E}$: the formula is satisfiable iff the language of the constructed PFWA is nonempty. Here we show a stronger decidability result. Namely, we give a decision procedure for sentences that have an arbitrary prefix of FO quantifiers and the body of the sentence or its negation is in $\mathbf{E}$. This is done by two constructions. We first construct a PFWA using Theorem 11, where we drop the prefix of FO quantifiers of the given sentence. Second, we construct from this PFWA a formula in Presburger arithmetic taking care of the quantification of the FO variables.

Theorem 16. *The fragment $[\text{FO}(\exists_{\text{MSO}}^* \cup \forall_{\text{MSO}}^*)\text{FO}; \text{RWS1S}]$ is decidable.*

Proof. Case I: $\varphi \in [\text{FO}\exists_{\text{MSO}}^*\text{FO}; \text{RWS1S}]$. Note that every relation R occurring in φ is expressible by a WS1S formula ψ_R . Therefore, we can assume that φ is of the form $Q_1x_1 \dots Q_mx_m\varphi'$ with $Q_1, \dots, Q_m \in \{\exists, \forall\}$ and $\varphi' \in \mathbf{E}$ by substituting the relations R with ψ_R . By Theorem 11 we can construct a PFWA $(\mathcal{A}, C)$ with dimension $N \geq 1$ and $L(\varphi') = L(\mathcal{A}, C)$. Assume that $\mathcal{A} = (S, \Gamma, \delta, s_I, F)$ with $\Gamma \subseteq \{0, 1\}^m \times \mathbb{N}^N$. It holds, $(\mathbb{N}, \text{succ}) \models Q_1x_1 \dots Q_mx_m\varphi'$ iff

$$Q_1\tilde{x}_1 \in \mathbb{N}, \dots, Q_m\tilde{x}_m \in \mathbb{N} \text{ there is a word } w \in L(\mathcal{A}, C) \text{ such that} \quad (1)$$

$$I_w(x_1) = \tilde{x}_1, \dots, I_w(x_m) = \tilde{x}_m.$$

By definition, (1) is equivalent to

$$Q_1\tilde{x}_1 \in \mathbb{N}, \dots, Q_m\tilde{x}_m \in \mathbb{N} \text{ there is a word } w \in L(\mathcal{A}) \text{ such that} \quad (2)$$

$$I_{\Psi(w)}(x_1) = \tilde{x}_1, \dots, I_{\Psi(w)}(x_m) = \tilde{x}_m \text{ and } \Phi(w) \in C,$$

where $\Psi : \Gamma \rightarrow \{0, 1\}^m$ is the projection and Φ the extended Parikh map of Γ .

We extend the alphabet Γ to $\Gamma' := \{(b, \bar{v}, \bar{v}') \mid (b, \bar{v}) \in \Gamma \text{ and } \bar{v}' \in \{0, 1\}^m\}$, that is, we append the vectors in $\{0, 1\}^m$ to each symbol $(b, \bar{v}) \in \Gamma$. Let Φ' be the extended Parikh map of Γ' , and let $h : \Gamma'^* \rightarrow \Gamma^*$ be the homomorphism defined by $h(b, \bar{v}, \bar{v}') := (b, \bar{v})$. We construct an automaton $\mathcal{A}'$ accepting $w \in \Gamma'^*$ iff $h(w) \in L(\mathcal{A})$ and $\Phi'(w) = (\Phi(h(w)), I_{\Psi(h(w))}(x_1), \dots, I_{\Psi(h(w))}(x_m))$. Let $\mathcal{A}' := (S, \Gamma', \delta', s_I, F)$, where the transition function δ' contains the same transitions as δ except that $\mathcal{A}'$ marks the positions in a word that determine the values of the interpretations for the FO variables $x_1, \dots, x_m$. For each x_i , let $B_i \subseteq S$ be the set that contains all the states that are reachable *before* reading a symbol that determines the interpretation of x_i , that is $B_i := \bigcup_{j \in \mathbb{N}} B_i^j$, where $B_i^0 := \{s_I\}$ and for $j > 0$, $B_i^{j+1} := \{s \in \delta(B_i^j, (b, \bar{v})) \mid (b, \bar{v}) \in \Gamma \text{ with } \chi_{x_i}(b) = 0\}$. Note that if a state s is in B_i and from s we can still reach an accepting state then for every word $w \in \Gamma'^*$ with $\delta(s_I, w) = s$ it holds that $\chi_{x_i}(w)$ is of the form $0 \dots 0$. Otherwise, there would be a word in $L(\mathcal{A}, C)$ that is not an interpretation for the FO variable x_i . For $s \in S$, let $\bar{c}(s) \in \{0, 1\}^m$ be the characteristic vector of s , that is $\bar{c}(s) := (c_1, \dots, c_m)$, where $c_i = 1$ iff $s \in B_i$. Now, $\delta' : S \times \Gamma' \rightarrow \mathcal{P}(S)$ is defined by $\delta'(s, (b, \bar{v}, \bar{v}')) := \{s' \in \delta(s, (b, \bar{v})) \mid \bar{c}(s') = \bar{v}'\}$. By the construction of $\mathcal{A}'$, (2) is equivalent to

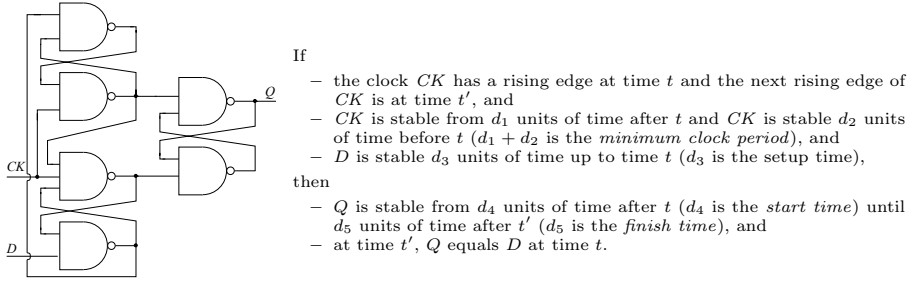


Fig. 2. Circuit of an edge-triggered D-type flip-flop and its specification.

$$Q_1 \tilde{x}_1 \in \mathbb{N}, \dots, Q_m \tilde{x}_m \in \mathbb{N} \text{ there is a word } w \in L(A') \text{ such that} \quad (3)$$

$$\Phi(h(w)) \in C \text{ and } \Phi'(w) = (\Phi(h(w)), \tilde{x}_1, \dots, \tilde{x}_m).$$

From Lemma 5, we know that $\Phi'(L(A'))$ is the union of the images of linear polynomials $q_1, \dots, q_\ell : \mathbb{N}^r \rightarrow \mathbb{N}^N$, for some $r \geq 1$. Moreover, these polynomials are constructible from A' . We conclude that (3) is equivalent to

$$Q_1 \tilde{x}_1 \in \mathbb{N}, \dots, Q_m \tilde{x}_m \in \mathbb{N} \text{ there are } y_1, \dots, y_r \in \mathbb{N} \text{ and } \bar{v} \in \mathbb{N}^N \text{ such that} \quad (4)$$

$$\bar{v} \in C \text{ and } q_i(y_1, \dots, y_r) = (\bar{v}, \tilde{x}_1, \dots, \tilde{x}_m), \text{ for some } 1 \leq i \leq \ell.$$

Note that (4) can be expressed as a sentence in Presburger arithmetic. The claim follows from the decidability of Presburger arithmetic.

Case II: $\varphi \in [\text{FO}\forall^*_{\text{MSO}}\text{FO}; \text{RWS1S}]$. Follows from Case I by the duality of quantifiers. $\dashv$

Applications. As an application, we sketch how this decidable fragment can be used to decide WS1S extended with some restricted linear arithmetic. Our example is the verification of an edge-triggered D-type flip-flop, taken from [3,8]. Although the circuit is built from only six nand-gates (left half of Figure 2), proving that the circuit meets its specification (right half of Figure 2) is “fairly complicated”, as Gordon noted in [8]. The proof in [8] was done by paper and pencil, and contained a flaw, as reported in [3,28]. The correctness proof in [3] was done automatically by naturally expressing the higher-order logic formalization from [8] in WS1S and using the implementation of the automata-based decision procedure for WS1S in the Mona tool [10]. This verification technique works only if the parameters $d_1, \dots, d_5$ are instantiated with concrete values because the specification contains some linear arithmetic, for example, “ Q is stable from d_4 units of time after t until d_5 units of time after t' ”. Reusing most of the WS1S formalization from [3] we can formalize in the decidable fragment of $\text{WS1S}^{\text{card}}$ whether the circuit meets its specification for all $d_1, \dots, d_5 \in \mathbb{N}$ satisfying, for instance, the constraints $d_1 \geq 2$, $d_2 \geq 2$, $d_1 + d_2 \geq 5$, $d_3 \geq 3$, $d_4 \geq 3$, and $d_5 \leq 2$. Together with Theorem 16 this demonstrates that such parameterized verification problems are actually decidable problems.

We briefly recall the formalization in [3].¹ To keep the formulas readable, we use some syntactic sugar for WS1S. It will always be straightforward to translate the used notation to WS1S. Note that $x \leq y$ can be defined by $\forall Z(Zy \wedge \forall z(Zz + 1 \rightarrow Zz) \rightarrow Zx)$.

The temporal behavior of a unit-delay nand-gate with inputs X and Y , and output Z up to time $\$$ is described by $nand(\$, X, Y, Z) := \forall t(t < \$ \rightarrow (Zt + 1 \leftrightarrow \neg(Xt \wedge Yt)))$, where $\$$ is an FO variable and X, Y , and Z are MSO variables. The temporal behavior of a nand-gate with three inputs can be described analogously. The circuit of the left half of Figure 2 implementing a D-type flip-flop can now be described by the following formula, where the internal wires are hidden by existential quantification.

$$\begin{aligned} imp(\$, D, CK, Q) := \exists W_1 \exists W_2 \exists W_3 \exists W_4 \exists W_5 (&nand(\$, W_2, D, W_1) \wedge \\ &nand3(\$, W_3, CK, W_1, W_2) \wedge nand(\$, W_4, CK, W_3) \wedge \\ &nand(\$, W_1, W_3, W_4) \wedge nand(\$, W_3, W_5, Q) \wedge \\ &nand(\$, Q, W_2, W_5)) \end{aligned}$$

We recall the definitions [3] of the temporal concepts needed to formalize the flip-flop's specification.

- X is stable in the interval $[t, t']$:

$$stable(t, t', X) := \forall u(t \leq u < t' \rightarrow (Xu \leftrightarrow Xt))$$

- X rises at t :

$$rise(t, X) := t > 0 \wedge \neg X(t-1) \wedge Xt$$

- t' is the next instance after t where X rises:

$$nextRise(t, t', X) := rise(t', X) \wedge \forall u(t < u < t' \rightarrow \neg rise(u, X))$$

The flip-flop's specification given in the right half of Figure 2 can be formalized as

$$\begin{aligned} spec(\$, t, t', D, CK, Q) := (&d_2 \leq t < t' \leq \$ - d_5 \wedge \\ &rise(t, CK) \wedge nextRise(t, t', CK) \wedge \\ &stable(t, t + d_1, CK) \wedge stable(t - d_2, t, CK) \wedge \\ &stable(t + 1 - d_3, t + 1, D)) \rightarrow \\ &stable(t + d_4, t' + d_5, Q) \wedge (Qt' \leftrightarrow Dt). \end{aligned}$$

Note that this formula is a WS1S formula if $d_1, \dots, d_5$ are not FO variables but natural numbers. For fixed values for $d_1, \dots, d_5$, Mona checks automatically if the circuit meets its specification by computing the truth value of the formula

$$verify(\$, t, t', D, CK, Q) := imp(\$, D, CK, Q) \rightarrow spec(\$, t, t', D, CK, Q).$$

¹ Actually, Basin and Klarlund did not use WS1S but M2L(str). There are some technical differences between these logics as explained in §3. We have adopted their formalization to WS1S.

In the following, we show how the decidability result from Theorem 16 can be used to check whether the circuit is correct, for instance, for all $d_1, \dots, d_5 \in \mathbb{N}$ with $d_1 \geq 2$, $d_2 \geq 2$, $d_1 + d_2 \geq 5$, $d_3 \geq 3$, $d_4 \geq 3$, and $d_5 \leq 2$. The constraints on the parameters can be expressed in WS1S by

$$\text{constr}(d_1, \dots, d_5) := (d_1 \geq 2 \wedge d_2 \geq 3 \vee d_1 \geq 3 \wedge d_2 \geq 2) \wedge d_3 \geq 3 \wedge d_4 \geq 3 \wedge d_5 \leq 2.$$

Unfortunately, $\forall d_1 \dots \forall d_5 (\text{constr}(d_1, \dots, d_5) \rightarrow \forall \$ \forall t \forall t' \forall D \forall CK \forall Q (\text{verify}(\$, t, t', D, CK, Q)))$ is not a WS1S^{card} formula, since *verify* contains in the subformula *spec* terms involving linear arithmetic, for example, $t + d_1$. But we can take a detour using MSO variables. For example, for the term $t + d_1$, we introduce an FO variable x_{t+d_1} and MSO variables T, D_1 with $x_{t+d_1} = |T| + |D_1|$, $T = \{0, \dots, t-1\}$, and $D_1 = \{0, \dots, d_1-1\}$. It holds $x_{t+d_1} = t + d_1$. Thus, the term $t + d_1$ can be substituted by x_{t+d_1} . Let *spec'* be the formula, where we replace in *spec* the terms τ involving linear arithmetic by fresh variables x_τ , that is,

$$\begin{aligned} \text{spec}'(\$, t, t', D, CK, Q, x_{\$-d_5}, x_{t+d_1}, x_{t-d_2}, x_{t-d_3}, x_{t+d_4}, x_{t'+d_5}) := \\ (d_2 \leq t < t' \leq x_{\$-d_5} \wedge \\ \text{rise}(t, CK) \wedge \text{nextRise}(t, t', CK) \wedge \\ \text{stable}(t, x_{t+d_1}, CK) \wedge \text{stable}(x_{t-d_2}, t, CK) \wedge \\ \text{stable}(x_{t-d_3} + 1, t + 1, D) \rightarrow \\ \text{stable}(x_{t+d_4}, x_{t'+d_5}, Q) \wedge (Qt' \leftrightarrow Dt)). \end{aligned}$$

We write $x = \pm |X_1| \pm \dots \pm |X_r|$ for $\exists Z (\forall z (Zz \leftrightarrow z < x) \wedge |Z| = \pm |X_1| \pm \dots \pm |X_r|)$, where x is an FO variable and the X_i s are MSO variables. The formula *aux* ensures that the new variables in *spec'* have the correct values, for example, x_{t+d_1} equals $t + d_1$.

$$\begin{aligned} \text{aux}(d_1, \dots, d_5, \$, t, t', x_{\$-d_5}, x_{t+d_1}, x_{t-d_2}, x_{t-d_3}, x_{t+d_4}, x_{t'+d_5}) := \\ \exists D_1 \dots \exists D_5 \exists \mathcal{L} \exists T \exists T' (d_1 = |D_1| \wedge \dots \wedge d_5 = |D_5| \wedge \\ \$ = |\mathcal{L}| \wedge t = |T| \wedge t' = |T'| \wedge \\ x_{\$-d_5} = |\mathcal{L}| - |D_5| \wedge x_{t+d_1} = |T| + |D_1| \wedge \\ x_{t-d_2} = |T| - |D_2| \wedge x_{t-d_3} = |T| - |D_3| \wedge \\ x_{t+d_4} = |T| + |D_4| \wedge x_{t'+d_5} = |T'| + |D_5|) \end{aligned}$$

For proving the circuit correct, we have to check whether the formula

$$\text{verify}' := \text{aux} \wedge \text{constr} \rightarrow (\text{imp} \rightarrow \text{spec}')$$

is valid. This can be done automatically by Theorem 16, since *verify'* can be transformed into a formula in $[\text{FO}\forall_{\text{MSO}}^* \text{FO}; \text{R}_{\text{WS1S}}]$ by universally quantifying over the FO variables and the MSO variables D, CK, Q , and by pulling out the existentially quantified MSO variables in *aux*. Note that the existential quantifiers become universal by this process. In addition to verification, our procedure may also be used for synthesizing sufficient parameter constraints if we do not

restrict the parameters $d_1, \dots, d_5$ by some constraints and do not universally quantify over them.

Although our decision procedure is built on top of a decision procedure for Presburger arithmetic and a translation from $\text{WS1S}^{\text{card}}$ formulas to PFWAs and the worst-case complexity is in both cases very high we are encouraged by the outcomes with a prototype implementation. We tested our implementation on various case studies, such as the D-type flip-flop above and lemmas in a PVS theory about cardinalities of finite sets that were used in [24] to verify oral message algorithms. Such kinds of proofs are cumbersome and rather involved. Our decidability result opens up the possibility to effectively automate such kinds of verification problems.

5 Conclusions

We have extended WS1S with linear cardinality constraints, proved the undecidability of this extension, and identified decidable fragments (see Figure 1). These results were obtained by extending the logic-automata connection to fragments of WS1S with cardinality constraints and an appropriate automaton model that we call *Parikh finite word automata*. The resulting decision procedure has applications in both hardware and protocol verification [14, 15], and initial experiments with an extension of the Mona tool with cardinality constraints are encouraging [13].

One advantage of our notion of Parikh word automata is that it easily generalizes to trees. A decidability result for a fragment of the weak monadic second-order logic of *two* successors with cardinality constraints using *Parikh finite tree automata* is included in [15]. Since monadic second-order logics on trees give a theoretical foundation of XML query languages [9], our results on trees may serve as a theoretical basis for extending current query languages as in [5].

The framework in §2 can also be generalized to infinite words and trees. A possible acceptance condition is in the spirit of the Büchi acceptance condition: one requires that the arithmetic constraints have to be satisfied for infinitely many prefixes in order to accept the input. Another extension that we want to look at is to generalize the framework to graphs with bounded tree-width.

Future work will include detailed complexity analyses, theoretically and practically, on Parikh automata and on the decision procedure for the decidable fragment of $\text{WS1S}^{\text{card}}$.

Acknowledgments. We thank J. Rushby for initiating and supporting this research, and the anonymous referees for their invaluable comments. The first author also thanks J. Meseguer.

References

1. D. BASIN AND S. FRIEDRICH, *Combining WS1S and HOL*, in FroCos'98, Applied Logic Series, 2000, pp. 39–56.

2. D. BASIN, S. FRIEDRICH, AND S. MÖDERSHEIM, *B2M: A semantic based tool for BLIF hardware descriptions*, in FMCAD'00, vol. 1954 of LNCS, 2000, pp. 91–107.
3. D. BASIN AND N. KLARLUND, *Automata based symbolic reasoning in hardware verification*, FMSD, 13 (1998), pp. 255–288.
4. H. COMON AND Y. JURSKI, *Multiple counters automata, safety analysis and Presburger arithmetic*, in CAV'98, vol. 1427 of LNCS, 1998, pp. 268–279.
5. S. DAL ZILIO AND D. LUGIEZ, *XML schema, tree logic and sheaves automata*, Research Report 4631, INRIA, 2002.
6. J. DASSOW AND V. MITRANA, *Finite automata over free groups*, International Journal of Algebra and Computation, 10 (2000), pp. 725–737.
7. A. FINKEL AND G. SUTRE, *Decidability of reachability problems for classes of two counter automata*, in STACS'00, vol. 1770 of LNCS, 2000, pp. 346–357.
8. M. GORDON, *Why higher-order logics is a good formalism for specifying and verifying hardware*, in Formal Aspects of VLSI Design, North-Holland, 1986, pp. 153–177.
9. G. GOTTLOB AND C. KOCH, *Monadic Datalog and the expressive power of languages for web information extraction*, in PODS'02, 2002, pp. 17–28.
10. J. HENRIKSEN, J. JENSEN, M. JORGENSEN, N. KLARLUND, B. PAIGE, T. RAUHE, AND A. SANDHOLM, *Mona: Monadic second-order logic in practice*, in TACAS'95, vol. 1019 of LNCS, 1995, pp. 89–110.
11. O. IBARRA, *Reversal-bounded multicounter machines and their decision problems*, JACM, 25 (1978), pp. 116–133.
12. O. IBARRA, J. SU, Z. DANG, T. BULTAN, AND R. KEMMERER, *Counter machines and verification problems*, TCS, 289 (2002), pp. 165–189.
13. F. KLAEDTKE, *CMona: Monadic second-order logics with linear cardinality constraints in practice*, in preparation, 2003.
14. F. KLAEDTKE AND H. RUESS, *WS1S with cardinality constraints*, Technical Report SRI-CSL-05-01, SRI International, 2001.
15. ———, *Parikh automata and monadic second-order logics with linear cardinality constraints*, Technical Report 177, Albert-Ludwigs-Universität Freiburg, 2002. (revised version).
16. N. KLARLUND, A. MØLLER, AND M. SCHWARTZBACH, *MONA implementation secrets*, in CIAA'00, vol. 2088 of LNCS, 2000, pp. 182–194.
17. N. KLARLUND, M. NIELSEN, AND K. SUNESEN, *Automated logical verification based on trace abstraction*, in PODC'96, 1996, pp. 101–110.
18. L. LAMPORT, R. SHOSTAK, AND M. PEASE, *The Byzantine Generals problem*, TOPLAS, 4 (1982), pp. 382–401.
19. A. MEYER, *Weak monadic second-order theory of successor is not elementary-recursive*, in Logic Colloquium, vol. 453 of LNM, 1975, pp. 132–154.
20. V. MITRANA AND R. STIEBE, *Extended finite automata over groups*, Discrete Applied Mathematics, 108 (2001), pp. 287–300.
21. S. OWRE AND H. RUESS, *Integrating WS1S with PVS*, in CAV'00, vol. 1855 of LNCS, 2000, pp. 548–551.
22. M. PARIGOT AND E. PELZ, *A logical approach of Petri net languages*, TCS, 39 (1985), pp. 155–169.
23. R. PARIKH, *On context-free languages*, JACM, 13 (1966), pp. 570–581.
24. J. RUSHBY, *Systematic formal verification for fault-tolerant time-triggered algorithms*, IEEE Trans. on Software Engineering, 2 (1999), pp. 651–660.
25. M. SMITH AND N. KLARLUND, *Verification of a sliding window protocol using IOA and MONA*, in FORTE/PSTV'00, vol. 183 of IFIP Conf. Proc., 2000, pp. 19–34.
26. L. STOCKMEYER, *The Complexity of Decision Problems in Automata Theory and Logic*, PhD thesis, Dept. of Electrical Engineering, MIT, Boston, Mass., 1974.

27. W. THOMAS, *Languages, automata, and logic*, in Handbook of Formal Languages, vol. 3, Springer-Verlag, 1997, pp. 389–455.
28. A. WILK AND A. PNUELI, *Specification and verification of VLSI systems*, in IC-CAD'89, 1989, pp. 460–463.

$\Pi_2 \cap \Sigma_2 \equiv AFMC$

Orna Kupferman^{1*} and Moshe Y. Vardi^{2**}

¹ Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel

orna@cs.huji.ac.il, <http://www.cs.huji.ac.il/~orna>

² Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
vardi@cs.rice.edu, <http://www.cs.rice.edu/~vardi>

Abstract. The μ -calculus is an expressive specification language in which modal logic is extended with fixpoint operators, subsuming many dynamic, temporal, and description logics. Formulas of μ -calculus are classified according to their *alternation depth*, which is the maximal length of a chain of nested alternating least and greatest fixpoint operators. Alternation depth is the major factor in the complexity of μ -calculus model-checking algorithms. A refined classification of μ -calculus formulas distinguishes between formulas in which the outermost fixpoint operator in the nested chain is a least fixpoint operator (Σ_i formulas, where i is the alternation depth) and formulas where it is a greatest fixpoint operator (Π_i formulas). The *alternation-free μ -calculus* (AFMC) consists of μ -calculus formulas with no alternation between least and greatest fixpoint operators. Thus, AFMC is a natural closure of $\Sigma_1 \cup \Pi_1$, which is contained in both Σ_2 and Π_2 . In this work we show that $\Sigma_2 \cap \Pi_2 \equiv AFMC$. In other words, if we can express a property ξ both as a least fixpoint nested inside a greatest fixpoint and as a greatest fixpoint nested inside a least fixpoint, then we can express ξ also with no alternation between greatest and least fixpoints. Our result refers to μ -calculus over arbitrary Kripke structures. A similar result, for directed μ -calculus formulas interpreted over trees with a fixed finite branching degree, follows from results by Arnold and Niwinski. Their proofs there cannot be easily extended to Kripke structures, and our extension involves *symmetric nondeterministic Büchi* tree automata, and new constructions for them.

1 Introduction

The μ -calculus is an expressive specification language in which formulas are built from Boolean operators, existential ($\Diamond$) and universal ($\Box$) next-time modalities, and least (μ) and greatest (ν) fixpoint operators [Koz83]. The discovery and use of *symbolic model-checking* methods [McM93] for verification of large systems

* Supported in part by NSF grant CCR-9988172 and by a research grant from the Center for Pure and Applied Mathematics at the University of California, Berkeley

** Supported in part by NSF grants CCR-9988322, CCR-0124077, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation.

has made the μ -calculus important also from a practical point of view: symbolic model-checking tools proceed by computing fixpoint expressions over the model's set of states. For example, to find the set of states from which a state satisfying some predicate p is reachable, the model checker starts with the set S of states in which p holds, and repeatedly add to S the set $\Diamond S$ of states that have a successor in S . Formally, the model checker calculates the set of states that satisfy the μ -calculus formula $\mu y.p \vee \Diamond y$.

Formulas of μ -calculus are classified according to their *alternation depth*, which is the maximal length of a chain of nested alternating least and greatest fixpoint operators. From a practical point of view, the classification is important, as the alternation depth is the major factor in the complexity of μ -calculus model-checking algorithms: the original algorithm for model checking a structure of size m with respect to a formula of length n and alternation depth d requires time $O(mn)^d$ [EL86], and more sophisticated algorithms can do the job in time roughly $O(mn)^{\lfloor \frac{d}{2} \rfloor + 1}$ [Jur00]. From a theoretical point of view, the classification naturally raises questions about the expressive power of the classes. In particular, the question whether the expressiveness hierarchy for the μ -calculus collapses (i.e., whether there is some $d \geq 1$ such that all μ -calculus formulas can be translated to formulas of alternation depth d) has been answered to the negative [Bra98]. The alternation-depth hierarchy of μ -calculus and the model-checking problem for the various classes in the hierarchy are strongly related to the index hierarchy in *parity games* and to the problem of deciding such games [Jur00].

A more refined classification of μ -calculus formulas distinguishes between formulas in which the outermost fixpoint operator in the nested chain is a least fixpoint operator (Σ_i formulas, where i is the alternation depth) and formulas where it is a greatest fixpoint operator (Π_i formulas). For example, the formula $\mu y.p \vee \Diamond y$ is a Σ_1 formula, as it has alternating depth 1 and its outermost fixpoint operator is μ . Similarly, the formula $\nu y.\mu z.\Box[(p \wedge y) \vee z]$ is a Π_2 formula¹. By duality of the least and greatest fixpoint operators, the classes Π_i and Σ_i are complementary, in the sense that a formula ψ is in Π_i iff the formula $\neg\psi$ (in positive normal form, where negation is applied to atomic propositions only) is in Σ_i .

Some fragments of μ -calculus are of special interest in computer science: *Modal Logic* (ML) consists of μ -calculus formulas with no fixpoint operators (that is, $\text{ML} = \Sigma_0 \cup \Pi_0$). It is actually more correct to say that μ -calculus is the extension of ML with fixpoint operators. Extending ML with fixpoint operators still retain some of its basic semantic properties, in particular the property of being invariant under bisimulation [Ben91]. The *alternation-free μ -calculus* (AFMC) consists of μ -calculus formulas with no alternation between least and greatest fixpoint operators. Thus, AFMC is a natural closure of $\Sigma_1 \cup \Pi_1$, which is contained in both Σ_2 and Π_2 . AFMC subsumes the branching temporal logic CTL and the dynamic logic PDL [FL79]. Formulas of AFMC

¹ An exact definition of the classes Σ_i and Π_i refers to the scope of the fixpoint operators. As we discuss in Section 4, several different definitions are studied in the literature, and we follow here the definition of [Niw86].

can be symbolically evaluated in time linear in the structure [CS91,KVW00]. While designers may prefer to use higher-level logics to specify properties, model-checking tools often proceed by evaluating the corresponding AFMC formulas [BRS99]. Finally, it is hard to produce an understandable formula with more than one alternation. Thus, $\Pi_2 \cup \Sigma_2$ subsumes almost all formulas one may wish to specify in practice. Formally, $\Pi_2 \cup \Sigma_2$ subsumes the branching temporal logic CTL^* , and in fact, until [Bra98], the strictness of the expressiveness hierarchy of μ -calculus was known only for Π_i and Σ_i with $i \leq 2$ [AN90]. Also, the symbolic evaluation of linear properties is reduced to calculating a Π_2 formula [VW86, EL85].

For several hierarchies in computer science, even strict ones, it is possible to show local *coalescence*, where membership in some class of the hierarchy and in its complementary class implies membership in a lower class. For example, $RE \cap co-RE = Rec$ describes coalescence at the bottom of the arithmetical hierarchy [Rog67]. On the other hand, the analogous coalescence for the polynomial hierarchy is not known; it is a major open question whether $NP \cap co-NP = P$ [GJ79]. In [KV01], we showed that the bottom levels of the μ -calculus expressiveness hierarchy coalesce: $\Sigma_1 \cap \Pi_1 \equiv ML$. In other words, if we can express a property ξ both as a least fixpoint and as a greatest fixpoint, then we can express ξ without fixpoints. The proof uses the fact that μ -calculus formulas in $\Sigma_1 \cap \Pi_1$ correspond to languages that are both safety and co-safety. Consequently, for every property $\xi \in \Sigma_1 \cap \Pi_1$, we can construct two nondeterministic *looping* tree automata $\mathcal{U}$ and $\mathcal{U}'$ such that $\mathcal{U}$ and $\mathcal{U}'$ accept exactly all the trees that satisfy ξ and its complement, respectively (the fact that $\mathcal{U}$ and $\mathcal{U}'$ are looping means that they have trivial acceptance conditions – every infinite run is accepting). We showed in [KV01] how $\mathcal{U}$ and $\mathcal{U}'$ can be combined to a *cycle-free* automaton and then translate to an ML formula expressing ξ .

In this paper we show coalescence in higher classes of the hierarchy, namely $\Sigma_2 \cap \Pi_2 \equiv AFMC$.² In other words, if we can specify a property ξ both as a least fixpoint nested inside a greatest fixpoint and as a greatest fixpoint nested inside a least fixpoint, then we can express ξ also with no alternation between greatest and least fixpoints. Unfortunately, the technique of [KV01] is too weak to be helpful here. Indeed, formulas in Π_2 cannot be expressed by looping automata. As we explain below, the known automata-theoretic characterizations of Σ_2 and Π_2 , and their relation to AFMC, cannot help us either.

One such known characterization [Niw86,AN92] refers to the expressive power of the μ -calculus over trees with fixed finite branching degrees. Over such trees, the existential next-time modality of the μ -calculus can be parameterized with *directions*. A modality parameterized with direction d means that the corresponding existential requirement should be satisfied in the d -th child of the current state. For example, for a binary tree in which each node has a left child and a right child, the formula $\Diamond_1 p$ means that the left child of the root satisfies

² The analogous complexity-theoretic result would be $\Sigma_2^P \cap \Pi_2^P = P^{NP}$, where Σ_2^P and Π_2^P form the second level of the polynomial hierarchy and P^{NP} is the polynomial closure of NP [GJ79].

p , and the formula $\mu y.p \vee \Diamond_{\mathbf{r}} y$ means that some node in the rightmost path of the tree satisfies p . The ability of *directed μ -calculus* to distinguish between the various children of a node makes it convenient to translate formulas to tree automata and vice versa. In particular, it is known that directed- Π_2 is as expressive as *nondeterministic Büchi tree automata* [AN90,Kai95]. Our interest in this paper is in the expressive power of the μ -calculus over arbitrary *Kripke structures*, possibly with an infinite branching degree, which means that we cannot restrict attention to trees of fixed branching degrees.

An automata-theoretic framework for μ -calculus without directions is suggested in [JW95], by means of *μ -automata*, which are essentially symmetric alternating tree automata in a certain normal form. A related approach, in which alternation is more explicit, is presented in [Wil99]. Alternation allows the automaton to send several requirements to the same child. Symmetry means that the automaton does not distinguish between the different children of a node, and it sends copies to child nodes only in either a universal or an existential manner. It also means that the automaton can handle trees with a variable and even infinite branching degree. Formulas of μ -calculus in Π_i and Σ_i can be linearly translated to symmetric alternating parity/co-parity automata of index i . While it is possible to translate μ -calculus formulas to symmetric alternating automata, it is not immediately clear how such a translation can help in a translating of $\Sigma_2 \cap \Pi_2$ into the AFMC. By [AN92,KV99], formulas that are members of both directed- Π_2 and directed- Σ_2 can be translated to directed-AFMC. The proofs in [AN92,KV99] shows that given a formula $\psi \in \Sigma_2 \cap \Pi_2$, we can construct two nondeterministic Büchi tree automata $\mathcal{U}$ and $\mathcal{U}'$, for ψ and $\neg\psi$, and then combine the automata to a weak alternating automaton equivalent to ψ . The combination of $\mathcal{U}$ and $\mathcal{U}'$, however, crucially depends on the fact that the automata are nondeterministic (rather than alternating) and the fact that the automata can refer to particular directions in the tree.

The key to the results in [KV01] and here is a development of a theory of *symmetric nondeterministic tree automata*. In [KV01], we defined symmetric nondeterministic *looping* automata, and showed how to construct such automata for formulas in Π_1 . In order to handle Σ_2 and Π_2 , we define here symmetric nondeterministic *Büchi* automata, and translate Π_2 formulas to such automata. From a technical point of view, symmetric nondeterministic tree automata are essentially symmetric alternating automata with transitions in disjunctive normal form. Our main contribution is the development of various constructions for symmetric nondeterministic tree automata and their application to the study of the expressive power of the μ -calculus. Since removal of alternation in Büchi automata should take into an account the acceptance condition of the automaton and keep track of the states visited in each path of the run tree, the symmetry of the automaton poses real technical challenges. We then extend the construction in [KV99] to symmetric automata and combine the symmetric nondeterministic Büchi tree automata for ψ and $\neg\psi$ to a symmetric weak alternating automaton for ψ . Again, symmetry poses real technical challenges. (In fact, while the construction in [KV99] for the directed case is quadratic, here we end up with

quadratically many states but exponentially many transitions.) Once we have a weak symmetric alternating automaton for ψ , it is possible to generate from it an equivalent AFMC formula [KV98].

2 Preliminaries

For a set $D \subseteq \mathbb{N}$ of directions, a *D-tree* is a nonempty set $T \subseteq D^*$, where for every $x \cdot d \in T$ with $x \in D^*$ and $d \in D$, we have $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T . For every $x \in T$, the nodes $x \cdot d$, for $d \in D$, are the *children* of x . A node with no children is a *leaf*. The *degree* of a node x is the number of children x has. Note that the degree of x is bounded by $|D|$. For technical convenience, we assume that the set D is finite³. A *D-tree* is *leafless* if it has no leaves. Note that a leafless tree is infinite. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or exactly one child of x is in π . For two nodes x_1 and x_2 of T , we say that $x_1 \leq x_2$ iff x_1 is a prefix of x_2 ; i.e., there exists $z \in D^*$ such that $x_2 = x_1 \cdot z$. We say that $x_1 < x_2$ iff $x_1 \leq x_2$ and $x_1 \neq x_2$. A *frontier* of a leafless tree is a set $E \subset T$ of nodes such that for every path $\pi \subseteq T$, we have $|\pi \cap E| = 1$. For example, the set $E = \{0, 100, 101, 11\}$ is a frontier of the $\{0, 1\}$ -tree $\{0, 1\}^*$. For two frontiers E_1 and E_2 , we say that $E_1 \leq E_2$ iff for every node $x_2 \in E_2$, there exists a node $x_1 \in E_1$ such that $x_1 \leq x_2$. We say that $E_1 < E_2$ iff for every node $x_2 \in E_2$, there exists a node $x_1 \in E_1$ such that $x_1 < x_2$. Note that while $E_1 < E_2$ implies that $E_1 \leq E_2$ and $E_1 \neq E_2$, the other direction does not necessarily hold. Given an alphabet Σ , a Σ -labeled *D-tree* is a pair $\langle T, V \rangle$ where T is a *D-tree* and $V : T \rightarrow \Sigma$ maps each node of T to a letter in Σ . We extend V to paths in a straightforward way. For a Σ -labeled *D-tree* $\langle T, V \rangle$ and a set $A \subseteq \Sigma$, we say that E is an *A-frontier* iff E is a frontier and for every node $x \in E$, we have $V(x) \in A$. We denote by $trees(D, \Sigma)$ the set of all Σ -labeled *D-trees*, and denote by $trees(\Sigma)$ the set of all Σ -labeled *D-trees*, for some D . For a set $\mathcal{T} \subseteq trees(\Sigma)$, we denote by $comp(\mathcal{T})$ the set of Σ -labeled trees that are not in $\mathcal{T}$; thus $comp(\mathcal{T}) = trees(\Sigma) \setminus \mathcal{T}$.

Automata on infinite trees (tree automata, for short) run on leafless Σ -labeled trees. *Alternating tree automata* generalize nondeterministic tree automata and were first introduced in [MS87]. *Symmetric alternating tree automata* [JW95, Wil99] are capable of reading trees with variable branching degrees. When a symmetric automaton reads a node of the input tree it sends copies to all successors of that node or to some successor. Formally, for a given set X , let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X . For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ satisfies θ . A symmetric alternating Büchi tree automaton (symmetric ABT, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$ where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(\{\square, \diamond\} \times Q)$ is

³ As we detail in the proof of Theorem 6, due to the bounded-tree-model property for μ -calculus, this technical assumption does not prevent us from proving our main result also for general structures with an infinite branching degree.

a transition function, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a Büchi acceptance condition. Intuitively, an atom $\langle \Box, q \rangle$ in $\delta(q, \sigma)$ denotes a universal requirement to send a copy of the automaton in state q to all the children of the current node. An atom $\langle \Diamond, q \rangle$ denotes an existential requirement to send a copy of the automaton in state q to some child of the current node. When, for instance, the automaton is in state q , reads a node x with k children $x \cdot 1, \dots, x \cdot k$, and $\delta(q, V(x)) = (\Box, q_1) \wedge (\Diamond, q_2) \vee (\Diamond, q_3) \wedge (\Diamond, q_4)$, it can either send k copies in state q_1 to the nodes $x \cdot 1, \dots, x \cdot k$ and send a copy in state q_2 to some node in $x \cdot 1, \dots, x \cdot k$ or send one copy in state q_3 to some node in $x \cdot 1, \dots, x \cdot k$ and send one copy in state q_4 to some node in $x \cdot 1, \dots, x \cdot k$. So, while nondeterministic tree automata send exactly one copy to each child, symmetric alternating automata can send several copies to the same child. On the other hand, symmetric alternating automata cannot distinguish between the different successors and can send copies to child nodes only in either a universal or an existential manner. Formally, a *run* of $\mathcal{A}$ on an input Σ -labeled D -tree $\langle T, V \rangle$, for some set D of directions, is an $(D^* \times Q)$ -labeled $\mathbb{N}$ -tree $\langle T_r, r \rangle$ such that $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$, and for all $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$, there is a (possibly empty) set $S \subseteq \{\Box, \Diamond\} \times Q$, such that S satisfies θ , and for all $(c, s) \in S$, the following hold: (1) If $c = \Box$, then for each $d \in D$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot d, s)$. (2) If $c = \Diamond$, then for some $d \in D$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot d, s)$. Note that if $\theta = \mathbf{true}$, then y need not have children. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta = \mathbf{false}$, we cannot have a run that takes a transition with $\theta = \mathbf{false}$. For a run $\langle T_r, r \rangle$ and an infinite path $\pi \subseteq T_r$, we define $\text{inf}(\pi)$ to be the set of states that are visited infinitely often in π , thus $q \in \text{inf}(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in T \times \{q\}$. A run $\langle T_r, r \rangle$ is accepting if all its infinite paths satisfy the Büchi acceptance condition; thus $\text{inf}(\pi) \cap F \neq \emptyset$. A tree $\langle T, V \rangle$ is accepted by $\mathcal{A}$ iff there exists an accepting run of $\mathcal{A}$ on $\langle T, V \rangle$, in which case $\langle T, V \rangle$ belongs to $\mathcal{L}(\mathcal{A})$. A tree $\langle T, V \rangle$ is accepted by $\mathcal{U}$ iff there exists an accepting run of $\mathcal{A}$ on $\langle T, V \rangle$, in which case $\langle T, V \rangle$ belongs to the language, $\mathcal{L}(\mathcal{A})$, of $\mathcal{A}$.

The transition function of an ABT $\mathcal{A}$ induces a graph $G_{\mathcal{A}} = \langle Q, E \rangle$ where $E(q, q')$ if there is $\sigma \in \Sigma$ such that $(\Box, q')$ or $(\Diamond, q')$ appears in $\delta(q, \sigma)$. An ABT is a *weak alternating tree automaton* (AWT, for short) if for each strongly connected component $C \subseteq Q$ of $G_{\mathcal{A}}$, either $C \subseteq F$ or $C \cap F = \emptyset$ [MSS86]. Note that every infinite path of a run of an AWT ultimately gets “trapped” within some strongly connected component C of $G_{\mathcal{A}}$. The path then satisfies the acceptance condition if and only if $C \subseteq F$.

The symmetry condition can also be applied to nondeterministic tree automata. In a *symmetric nondeterministic Büchi tree automaton* (symmetric NBT, for short) $\mathcal{U} = \langle \Sigma, Q, \delta, q_0, F \rangle$, the state space is $Q = 2^S$ for some set S of *micro-states*, and the transition function $\delta : Q \times \Sigma \rightarrow 2^{2^S \times 2^S}$ maps a state and a letter to sets of pairs $\langle U, E \rangle$ of subsets of S . The set $U \subseteq S$ is the *universal set* and it describes the micro-states that should be members in all the child states. The set $E \subseteq S$ is the *existential set* and it describes micro-states

each of which has to be a member in at least one child state. Formally, given $k \geq 1$, a k -tuple $\langle S_1, \dots, S_k \rangle$ is induced by $\delta(q, \sigma)$ if there is $\langle U, E \rangle$ in $\delta(q, \sigma)$ such that for all $1 \leq i \leq k$ we have $U \subseteq S_i$, and for all $s \in E$ there is $1 \leq i \leq k$ such that $s \in S_i$. Intuitively, when the automaton reads a node x labeled σ that has k children, and it proceeds from the state q , it has to take two choices. First, the automaton chooses a pair $\langle U, E \rangle \in \delta(q, \sigma)$. Then, it chooses a way to deliver E among the k children. Thus, we can describe the two choices of the automaton by a pair $\langle U, \langle E_1, \dots, E_k \rangle \rangle$, where $\langle U, \bigcup_{1 \leq i \leq k} E_i \rangle \in \delta(q, \sigma)$. Note that E_z may be empty. We denote by $\delta_k(q, \sigma)$ the set of such pairs. A *run* of $\mathcal{U}$ on an input tree $\langle T, V \rangle$ is a Q -labeled tree $\langle T, r \rangle$, such that $r(\varepsilon) = q_0$, and for every $x \in T$ with $r(x) = q$, there exists $\langle q_1, \dots, q_k \rangle \in \delta_k(q, V(x))$ such that for all $1 \leq i \leq k$, we have $r(x \cdot i) = q_i$. Note that each node of the input tree corresponds to exactly one node in the run tree. A run $\langle T, r \rangle$ is accepting if all its paths satisfy the Büchi acceptance condition. Thus, for all paths π , we have $\inf(\pi) \cap F = \emptyset$, where $q \in \inf(\pi)$ if and only if there are infinitely many $x \in \pi$ for which $r(x) = q$. Equivalently, $\langle T, r \rangle$ is accepting iff $\langle T, r \rangle$ contains infinitely many F -frontiers $G_0 < G_1 < \dots$. For a state $q \in Q$, let $\mathcal{U}^q$ be $\mathcal{U}$ with initial state q . We say that a symmetric NBT is *monotonic* if for every two states q and p such that $q \subseteq p$, we have that $\mathcal{L}(\mathcal{U}^p) \subseteq \mathcal{L}(\mathcal{U}^q)$, and $p \in F$ implies $q \in F$. In other words, the smaller the state is, the easier it is to accept from it. Note that symmetric nondeterministic tree automata are essentially symmetric alternating automata with transitions in disjunctive normal form (DNF); if we write the transition functions in DNF, then each disjunct is a conjunction of universal and existential requirements, corresponding to a pair $\langle U, E \rangle$.

3 From Symmetric NBT and Co-NBT to Symmetric AWT

Let $\mathcal{U} = \langle \Sigma, \mathcal{D}, Q, q_0, M, F \rangle$ and $\mathcal{U}' = \langle \Sigma, \mathcal{D}, Q', q'_0, M', F' \rangle$ be two NBT, and let $|Q| \cdot |Q'| = m$. In [Rab70], Rabin studies the joint behavior of a run of $\mathcal{U}$ with a run of $\mathcal{U}'$. Recall that an accepting run of $\mathcal{U}$ contains infinitely many F -frontiers $G_0 < G_1 < \dots$, and an accepting run of $\mathcal{U}'$ contains infinitely many F' -frontiers $G'_0 < G'_1 < \dots$. It follows that for every labeled tree $\langle T, V \rangle \in \mathcal{L}(\mathcal{U}) \cap \mathcal{L}(\mathcal{U}')$ and accepting runs $\langle T, r \rangle$ and $\langle T, r' \rangle$ of $\mathcal{U}$ and $\mathcal{U}'$ on $\langle T, V \rangle$, the joint behavior of $\langle T, r \rangle$ and $\langle T, r' \rangle$ contains infinitely many frontiers $E_i \subset T$, with $E_i < E_{i+1}$, such that $\langle T, r \rangle$ reaches an F -frontier and $\langle T, r' \rangle$ reaches an F' -frontier between E_i and E_{i+1} . Rabin shows that the existence of m such frontiers, in the joint behavior of some runs of $\mathcal{U}$ and $\mathcal{U}'$, is sufficient to imply that the intersection $\mathcal{L}(\mathcal{U}) \cap \mathcal{L}(\mathcal{U}')$ is not empty. We now extend Rabin's result to symmetric automata.

Assume that $\mathcal{U}$ and $\mathcal{U}'$ above are symmetric NBT. We say that a sequence $E_0, \dots, E_m$ of frontiers of T is a *trap* for $\mathcal{U}$ and $\mathcal{U}'$ iff $E_0 = \{\varepsilon\}$ and there exists a tree $\langle T, V \rangle$ and (not necessarily accepting) runs $\langle T, r \rangle$ and $\langle T, r' \rangle$ of $\mathcal{U}$ and $\mathcal{U}'$ on $\langle T, V \rangle$, such that for every $0 \leq i \leq m - 1$, we have that $\langle T, r \rangle$ contains an F -frontier G_i such that $E_i \leq G_i < E_{i+1}$, and $\langle T, r' \rangle$ contains an F' -frontier G'_i

such that $E_i \leq G'_i < E_{i+1}$. We say that $\langle T, r \rangle$ and $\langle T, r' \rangle$ *witness* the trap for $\mathcal{U}$ and $\mathcal{U}'$.

Theorem 1. *Consider two symmetric nondeterministic Büchi tree automata $\mathcal{U}$ and $\mathcal{U}'$. If there exists a trap for $\mathcal{U}$ and $\mathcal{U}'$, then $\mathcal{L}(\mathcal{U}) \cap \mathcal{L}(\mathcal{U}')$ is not empty.*

Proof. The proof follows the same line of reasoning as in [Rab70]. For a state $q \in Q$, let $\mathcal{U}^q$ be $\mathcal{U}$ with initial state q , and similarly for $q' \in Q'$ and $\mathcal{U}'^{q'}$. We define a sequence of relations over $Q \times Q'$. Let $H_0 = Q \times Q'$. Then, $\langle q, q' \rangle \in H_{i+1}$ iff $\langle q, q' \rangle \in H_i$ and there is a nonempty Σ -labeled D -tree $\langle T, V \rangle$, a frontier $E \subseteq T$, and runs $\langle T, r \rangle$ and $\langle T, r' \rangle$ of $\mathcal{U}^q$ and $\mathcal{U}'^{q'}$ on $\langle T, V \rangle$, such that there is an F -frontier $G < E$ and an F' -frontier $G' < E$, such that for all $x \in E$, we have $\langle r(x), r'(x) \rangle \in H_i$. It is easy to see that $H_0 \supseteq H_1 \supseteq H_2 \supseteq \dots$. Also, if $H_i = H_{i+1}$, then $H_i = H_{i+k}$ for all $k \geq 0$. In particular, since $|Q| \times |Q'| = m$, it must be that $H_m = H_{m+k}$ for all $k \geq 0$. As in [Rab70], it can now be shown that $\mathcal{L}(\mathcal{U}) \cap \mathcal{L}(\mathcal{U}') \neq \emptyset$ iff $H_m(q_0, q'_0)$, and the result follows.

Theorem 1 is the key to the construction described in Theorem 2 below.

Theorem 2. *Let $\mathcal{U}$ and $\mathcal{U}'$ be two symmetric monotonic NBT with $\mathcal{L}(\mathcal{U}') = \text{comp}(\mathcal{L}(\mathcal{U}))$. There exists a symmetric AWT $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{U})$.*

Proof. Let $\mathcal{U} = \langle \Sigma, Q, q_0, M, F \rangle$ and $\mathcal{U}' = \langle \Sigma, Q', q'_0, M', F' \rangle$, and let $|Q| \cdot |Q'| = m$. Also, let S and S' be the micro-states of $\mathcal{U}$ and $\mathcal{U}'$, respectively, thus $Q = 2^S$ and $Q' = 2^{S'}$. We define the symmetric AWT $\mathcal{A} = \langle \Sigma, P, p_0, \delta, \alpha \rangle$ as follows.

- $P = Q \times Q' \times \{0, \dots, 2m-1\}$ and $p_0 = \langle q_0, q'_0, 0 \rangle$. Intuitively, a copy of $\mathcal{A}$ that visits the state $\langle q, q', i \rangle$ as it reads the node x of the input tree corresponds to runs r and r' of $\mathcal{U}$ and $\mathcal{U}'$ that visit the states q and q' , respectively, as they read the node x of the input tree. Let $\rho = y_0, y_1, \dots, y_{|x|}$ be the path from ε to x . Consider the joint behavior of r and r' on ρ . We can represent this behavior by a sequence $\tau_\rho = \langle t_0, t'_0 \rangle, \langle t_1, t'_1 \rangle, \dots, \langle t_{|x|}, t'_{|x|} \rangle$ of pairs in $Q \times Q'$ where $t_j = r(y_j)$ and $t'_j = r'(y_j)$. We say that a pair $\langle t, t' \rangle \in Q \times Q'$ is an F -pair iff $t \in F$ and is an F' -pair iff $t' \in F'$. We can partition the sequence τ_ρ to blocks $\beta_0, \beta_1, \dots, \beta_i$ such that we close block β_b and open block β_{b+1} whenever we reach the first F' -pair that is preceded by an F -pair in β_b . In other words, whenever we open a block, we first look for an F -pair, ignoring F' -pairness. Once an F -pair is detected, we look for an F' -pair, ignoring F -pairness. Once an F' -pair is detected, we close the current block and we open a new block. Note that a block may contain a single pair that is both an F -pair and an F' -pair. The third element of a state keeps track of the visits to blocks. When we visit $\langle q, q', i \rangle$, the index of the last block in τ_ρ is $\lfloor \frac{i}{2} \rfloor$, and this block already contains an F -pair iff i is odd. We refer to i as the *status* of the state $\langle q, q', i \rangle$. For a status $i \in \{0, \dots, 2m-1\}$, let $P_i = Q \times Q \times \{i\}$ be the set of states with status i .

- In order to define the transition function δ , we first define a function $next : P \rightarrow \{0, \dots, 2m-1\}$ that updates the status of states. For that, we first define the function $next' : P \rightarrow \{0, \dots, 2m\}$ as follows.

$$next'(\langle q, q', i \rangle) = \begin{cases} i & \text{If } (i \text{ is even and } q \notin F) \text{ or } (i \text{ is odd and } q' \notin F') \\ i+1 & \text{If } (i \text{ is even and } q \in F \text{ and } q' \notin F') \text{ or } (i \text{ is odd and } q' \in F') \\ i+2 & \text{If } i \text{ is even and } q \in F \text{ and } q' \in F'. \end{cases}$$

Now, $next(\langle q, q', i \rangle) = \min\{next'(\langle q, q', i \rangle), 2m-1\}$.

Intuitively, $next$ updates the status of states by recording and tracking of blocks. Recall that the status i indicates in which block we are and whether an F -pair in the current block has already been detected. The conditions for not changing i or for increasing it to $i+1$ and $i+2$ follow directly from the definition of the status. For example, the new status stays i if the current i is even and $\langle q, q' \rangle$ is not an F -block, or if i is odd and $\langle q, q' \rangle$ is not an F' -block. When i reaches or exceeds $2m-1$, we no longer increase it, even if $q' \in F'$. The automaton $\mathcal{A}$ proceeds as follows. Essentially, for every run $\langle T, r' \rangle$ of $\mathcal{U}'$, the automaton $\mathcal{A}$ guesses a run $\langle T, r \rangle$ of $\mathcal{U}$ such that for every path ρ of T , the run $\langle T, r \rangle$ visits F along ρ at least as many times as $\langle T, r' \rangle$ visits F' along ρ . Thus, when we record blocks along ρ , we do not want to get stuck in an even status. Since $\mathcal{L}(\mathcal{U}) \cap \mathcal{L}(\mathcal{U}') = \emptyset$, then, by Theorem 1, no run $\langle T, r \rangle$ can witness with $\langle T, r' \rangle$ a trap for $\mathcal{U}$ and $\mathcal{U}'$. Consequently, recording of visits to F and F' along ρ can be completed once $\mathcal{A}$ detects that τ_ρ contains m blocks as above.

Recall that $Q = 2^S$ and $Q' = 2^{S'}$. For a set $E \subseteq S$, a *partition* of E is a set $\{E_1, \dots, E_l\}$ with $E_i \subseteq E$ such that $E = \bigcup_{1 \leq i \leq l} E_i$, and for all $1 \leq i \neq j \leq n$, we have $E_i \cap E_j = \emptyset$. Let $par(E)$ be the set of partitions of E . Consider a set $E' \subseteq S'$ and a partition $\gamma' \in par(E')$. For a set $E \subseteq S$, we say that a partition η of $E \cup E'$ *agrees with* γ' if for all s'_1 and s'_2 in E' , we have that s'_1 and s'_2 are in the same set in η iff they are in the same set in γ' . Let $agree(E, \gamma')$ be the set of partitions of $E \cup E'$ that agree with γ' . For example, if $E = \{s_1\}$ and $E' = \{s_2, s_3\}$, then the two possible partitions of E' are $\gamma'_1 = \{\{s_2, s_3\}\}$ and $\gamma'_2 = \{\{s_2\}, \{s_3\}\}$. Then, $agree(E, \gamma'_1)$ contains the two partitions $\{\{s_1, s_2, s_3\}\}$ and $\{\{s_1\}, \{s_2, s_3\}\}$, and $agree(E, \gamma'_2)$ contains the three partitions $\{\{s_1, s_2\}, \{s_3\}\}$, $\{\{s_1, s_3\}, \{s_2\}\}$, and $\{\{s_1\}, \{s_2\}, \{s_3\}\}$. Now, let $p = \langle q, q', i \rangle$ be a state in P such that $M(q, \sigma) = \{\langle U_1, E_1 \rangle, \dots, \langle U_n, E_n \rangle\}$ and $M'(q', \sigma) = \{\langle U'_1, E'_1 \rangle, \dots, \langle U'_{n'}, E'_{n'} \rangle\}$. We distinguish between two cases.

- If $i < 2m-1$ or $q \notin F$, then

$$\delta(p, \sigma) = \bigwedge_{1 \leq j' \leq n'} \bigwedge_{\gamma' \in par(E'_{j'})} \left(\bigvee_{1 \leq j \leq n} \bigvee_{\eta \in agree(E_j, \gamma')} go(j, j', \eta, next(p)) \right), \text{ where}$$

$$go(j, j', \eta, l) = \Box \langle U_j, U'_{j'}, l \rangle \wedge \bigwedge_{X \in \eta} \Diamond \langle U_j \cup (X \cap E_j), U'_{j'} \cup (X \cap E'_{j'}), l \rangle.$$

That is, for every choice of $\mathcal{U}'$ for a $1 \leq j' \leq n'$ and for the way the existential requirements in $E'_{j'}$ are partitioned, there is a choice of $\mathcal{U}$

for a $1 \leq j \leq n$ and for the way the existential requirements in E_j are partitioned and combined with these in E'_j , to a partition of $E_j \cup E'_j$, such that the universal requirements in U_j and U'_j are sent to all directions, and existential requirements that are in the same set in the joint partition of $E_j \cup E'_j$ are sent to the same direction. Note that the sets U_j and U'_j are sent along with the existential requirements. This guarantees that the states that are sent in the existential mode correspond to the states that $\mathcal{U}$ and $\mathcal{U}'$ visit, and not to subsets of such states.

- If $i = 2m - 1$ and $q \in F$, then $\delta(p, \sigma) = \mathbf{true}$.

Note that $\text{par}(E')$ is exponential in $|E'|$, and the number of possible $\eta \in \text{agree}(E, \gamma')$ is exponential in $E \cup E'$. Thus, the size of δ is exponential in the sizes of M and M' .

- $\alpha = Q \times Q' \times \{i : i \text{ is odd}\}$. Thus, α makes sure that infinite paths of the run visits infinitely many states in which the status is odd, thus states in which we are in the second phase of blocks. *moshe2*:

Each set P_i is a strongly connected component, thus the automaton $\mathcal{A}$ is indeed an AWT. Note that, by the definition of α , a run is accepting iff no path of it gets trapped in a set of the form P_i , for an even i , namely a set in which $\mathcal{A}$ is waiting for a visit of $\mathcal{U}$ in a state in F . The number of states of $\mathcal{A}$ is $O(m^2)$. We prove that $\mathcal{L}(\mathcal{U}) = \mathcal{L}(\mathcal{A})$. We first prove that $\mathcal{L}(\mathcal{U}) \subseteq \mathcal{L}(\mathcal{A})$. Consider a D -tree $\langle T, V \rangle$. With every run $\langle T, r \rangle$ of $\mathcal{U}$ on $\langle T, V \rangle$ we can associate a run $\langle T_R, R \rangle$ of $\mathcal{A}$ on $\langle T, V \rangle$. Intuitively, the run $\langle T, r \rangle$ directs $\langle T_R, R \rangle$ in the nondeterminism in δ (that is, the choices of $1 \leq j \leq n$ and $\eta \in \text{agree}(E_j, \gamma')$). Formally, recall that a run of $\mathcal{A}$ on a D -tree $\langle T, V \rangle$ is a $(T \times P)$ -labeled tree $\langle T_R, R \rangle$, where a node $y \in T_R$ with $R(y) = \langle x, p \rangle$ corresponds to a copy of $\mathcal{A}$ that reads the node $x \in T$ and visits the state p . We define $\langle T_R, R \rangle$ as follows.

- $\varepsilon \in T_R$ and $R(\varepsilon) = (\varepsilon, \langle q_0, q'_0, 0 \rangle)$.
- Consider a node $y \in T_R$ with $R(y) = (x, \langle q, q', i \rangle)$. By the definition of $\langle T_R, R \rangle$ so far, we have $r(x) = t$ for $q \subseteq t$. Consider first the case that $t = q$. Let $\{x \cdot 1, \dots, x \cdot k\}$ be the children of x in T , and let $\langle U, \langle E_1, \dots, E_k \rangle \rangle \in M_k(q, V(x))$ describe the choice $\mathcal{U}$ makes when it proceeds from the node x . Thus, for each $1 \leq z \leq k$, we have $r(x \cdot z) = U \cup E_z$. Let $j = \text{next}(\langle q, q', i \rangle)$. Consider the set

$$Y = \bigcup_{\substack{\langle U', \langle E'_1, \dots, E'_k \rangle \rangle \in \\ M'_k(q', V(x))}} \{(1, \langle U, U', j \rangle), (1, \langle U \cup E_1, U' \cup E'_1, j \rangle), \dots, (k, \langle U, U', j \rangle), (k, \langle U \cup E_k, U' \cup E'_k, j \rangle)\}.$$

By the definition of δ , the set Y satisfies $\delta(\langle q, q', i \rangle, V(x))$ ⁴. Let $l = |M'_k(q', V(x))|$, and let $\langle U'^w, E'^w_1, \dots, E'^w_k \rangle$, for $1 \leq w \leq l$, be the w -th pair in $M'_k(q', V(x))$. For all $1 \leq w \leq l$ and $1 \leq z \leq k$, we have $\{y \cdot (2k(w-1) + z - 1), y \cdot (2k(w-1) + z)\} \subseteq T_R$, with $R(y \cdot (2k(w-1) + z - 1)) = (x \cdot z, \langle U, U'^w, j \rangle)$ and $R(y \cdot (2k(w-1) + z)) = (x \cdot z, \langle U \cup E_z, U'^w \cup E'^w_z, j \rangle)$.

⁴ Note that $\delta(\langle q, q', i \rangle, V(x))$ is a formula in $\mathcal{B}^+(\{\square, \diamond\} \times P)$, whereas $Y \subseteq \{1, \dots, k\} \times P$, but the extension of the satisfaction relation to this setting is straightforward: an atom $(\diamond, p)$ is satisfied in Y if there is $1 \leq z \leq k$ with $(z, p) \in Y$, and an atom $(\square, p)$ is satisfied in Y if for all $1 \leq z \leq k$, we have $(z, p) \in Y$.

Note that the invariant that for all $y \in T_R$ with $R(y) = (x, \langle q, q', i \rangle)$, we have $r(x) = t$ for $q \subseteq t$, is maintained. In fact, we know that all the nodes $y \in T_R$ that correspond to copies of $\mathcal{A}$ that satisfy an existential requirement have $q = t$, and node $y \in T_R$ that correspond to copies of $\mathcal{A}$ that satisfy a universal requirement have $q = t$ iff the run r sends no existential requirement to the corresponding direction.

Consider now the case where $q \subset t$. Since $\mathcal{U}$ is monotonic, there is an accepting run $\langle T^x, r_q^x \rangle$ of $\mathcal{U}^q$ on the subtree of T with root x . We can proceed exactly as above, with $\langle T^x, r_q^x \rangle$ instead of $\langle T, r \rangle$.

Consider a tree $\langle T, V \rangle \in \mathcal{L}(\mathcal{U})$. Let $\langle T, r \rangle$ be an accepting run of $\mathcal{U}$ on $\langle T, V \rangle$, and let $\langle T_R, R \rangle$ be the run of $\mathcal{A}$ on $\langle T, V \rangle$ induced by $\langle T, r \rangle$ (and the “subtree runs”, like $\langle T^x, r_q^x \rangle$ above). It can be shown that $\langle T_R, R \rangle$ is a legal accepting run. Indeed, since $\langle T, r \rangle$ and the subtree runs contains infinitely many F -frontiers, and since (by the definition of monotonic automaton) we do not lose visits to F when we switch to subset runs), no infinite paths of $\langle T_R, R \rangle$ can get trapped in a set P_i for an even i .

It is left to prove that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{U})$. For that, we prove that $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{U}') = \emptyset$. Since $\mathcal{L}(\mathcal{U}) = \text{comp}(\mathcal{L}(\mathcal{U}'))$, it follows that every tree that is accepted by $\mathcal{A}$ is also accepted by $\mathcal{U}$. Consider a tree $\langle T, V \rangle$. With each run $\langle T_R, R \rangle$ of $\mathcal{A}$ on $\langle T, V \rangle$ and run $\langle T, r' \rangle$ of $\mathcal{U}'$ on $\langle T, V \rangle$, we associate a run $\langle T, r \rangle$ of $\mathcal{U}$ on $\langle T, V \rangle$. Intuitively, $\langle T, r \rangle$ makes the choices that $\langle T_R, R \rangle$ has made in its copies that correspond to the run $\langle T, r' \rangle$. Formally, $\langle T, r \rangle$ is such that $r(\varepsilon) = q_0$, and for all $x \in T$ with $r(x) = q$, we proceed as follows. Let $\{x \cdot 1, \dots, x \cdot k\}$ be the children of x in T , and let $r'(x) = q'$. The run $\langle T, r' \rangle$ selects a pair $\langle U', \langle E'_1, \dots, E'_k \rangle \rangle \in M'_k(q', V(x))$ that $\mathcal{U}'$ proceeds with when it reads the node x . Formally, for all $1 \leq z \leq k$, we have $r'(x \cdot z) = U' \cup E'_z$.⁵ By the definition of $r(x)$ so far, the run $\langle T_R, R \rangle$ contains a node $y \in T_R$ with $R(y) = \langle x, \langle q, q', i \rangle \rangle$ for some status i . If $\delta(\langle q, q', i \rangle, V(x)) = \text{true}$, we define the reminder of $\langle T, r \rangle$ arbitrarily. Otherwise, let $1 \leq j' \leq n'$ and $\gamma' \in \text{par}(E'_{j'})$ be such that $\langle U', \langle E'_1, \dots, E'_k \rangle \rangle$ corresponds to j' and γ' . By the definition of δ , there are $1 \leq j \leq n$ and $\eta \in \text{agree}(E_j, \gamma')$ such that $\text{go}(j, j', \eta, \text{next}(\langle q, q', i \rangle))$ is satisfied and R proceeds according to j and η . Thus, if $\{E_j^1, \dots, E_j^k\}$ is the partition of E_j that corresponds to η , then T_R contains at least k nodes $y \cdot c_z$, for $1 \leq z \leq k$, such that $R(y \cdot c_z) = \langle x \cdot z, \langle U_j \cup E_j^z, U' \cup E'_z, \text{next}(\langle q, q', i \rangle) \rangle \rangle$. For all $1 \leq z \leq k$, we define $r(x \cdot z) = U_j \cup E_j^z$. Note that the invariant about the runs $\langle T, r \rangle$ and $\langle T_R, R \rangle$ is maintained. Note also that if $E_j^z \cup E'_z = \emptyset$, then the existence of a node $y \cdot c_z$ as above is guaranteed from universal part of δ , and if $E_j^z \cup E'_z \neq \emptyset$, its existence is guaranteed from the existential part (in which case it is crucial that we sent the universal requirements along with the existential ones).

We can now prove that $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{U}') = \emptyset$. Assume, by way of contradiction, that there exists a tree $\langle T, V \rangle$ such that $\langle T, V \rangle$ is accepted by both $\mathcal{A}$ and $\mathcal{U}'$. Let

⁵ For a monotonic NBT, we assume that runs satisfy the requirements in transition function in an optimal way; thus when $\mathcal{A}$ chooses to proceed with $\langle U', \langle E'_1, \dots, E'_k \rangle \rangle \in M'_k(q', V(x))$, it is indeed the case that $r'(x \cdot z) = U' \cup E'_z$. If $r'(x \cdot z) \supset U' \cup E'_z$, we can replace r' with a run for which the equation holds.

$\langle T_R, R \rangle$ and $\langle T, r' \rangle$ be the accepting runs of $\mathcal{A}$ and $\mathcal{U}'$ on $\langle T, V \rangle$, respectively, and let $\langle T, r \rangle$ be the run of $\mathcal{U}$ on $\langle T, V \rangle$ induced by $\langle T_R, R \rangle$ and $\langle T, r' \rangle$. We claim that then, $\langle T, r \rangle$ and $\langle T, r' \rangle$ witness a trap for $\mathcal{U}$ and $\mathcal{U}'$. Since, however, $\mathcal{L}(\mathcal{U}) \cap \mathcal{L}(\mathcal{U}') = \emptyset$, it follows from Theorem 1, that no such trap exists, and we reach a contradiction. To see that $\langle T, r \rangle$ and $\langle T, r' \rangle$ indeed witness a trap, define $E_0 = \{\varepsilon\}$, and define, for $0 \leq i \leq m-1$, the set E_{i+1} to contain exactly all nodes x for which there exists $y \in T_R$ such that either $R(y) = \langle x, \langle r(x), r'(x), 2i+1 \rangle \rangle$ and $r'(x) \in F'$ or $R(y) = \langle x, \langle r(x), r'(x), 2i \rangle \rangle$ and $r(x) \in F$ and $r'(x) \in F'$. That is, for every path ρ of T , the set E_{i+1} consists of the nodes in which the i 'th block is closed in τ_ρ . By the definition of δ , for all $0 \leq i \leq m-1$, the run $\langle T, r \rangle$ contains an F -frontier G_i such that $E_i \leq G_i < E_{i+1}$ and the run $\langle T, r' \rangle$ contains an F' -frontier G'_i such that $E_i \leq G'_i < E_{i+1}$. Hence, $E_0, \dots, E_m$ is a trap for $\mathcal{U}$ and $\mathcal{U}'$.

4 From $\Pi_2 \cap \Sigma_2$ to the Alternation-Free μ -Calculus

The μ -calculus is a propositional modal logic augmented with least and greatest fixpoint operators [Koz83]. Specifically, we consider a μ -calculus where formulas are constructed from Boolean propositions with Boolean connectives, the temporal operators $\exists\bigcirc$ (“exists next”) and $\Box$ (“for all next”), as well as least (μ) and greatest (ν) fixpoint operators. We assume that μ -calculus formulas are written in positive normal form (negation only applied to atomic propositions constants and variables). Formally, given a set AP of atomic proposition constants and a set APV of atomic proposition variables, a μ -calculus formula is either:

- **true, false**, p or $\neg p$ for all $p \in AP$.
- y for all $y \in APV$;
- $\varphi \wedge \psi$, $\varphi \vee \psi$, $\Diamond\varphi$, or $\Box\varphi$, where φ and ψ are μ -calculus formulas;
- $\mu y.\varphi(y)$ or $\nu y.\varphi(y)$, where $y \in APV$ and $\varphi(y)$ is a μ -calculus formula containing y as a free variable.

We classify formulas to classes Σ_i and Π_i according to the nesting of fixpoint operators in them. Several versions to such a classification can be found in the literature [EL86, Niw86, Bra98]. We describe here the version defined in [Niw86]:

- A formula is in $\Sigma_0 = \Pi_0$ if it contains no fixpoint operators.
- A formula is in Σ_{i+1} if it is one of the following θ_i , $\theta_i \wedge \theta'_i$, $\theta_i \vee \theta'_i$, $\Diamond\theta_i$, $\Box\theta_i$, $\mu y.\varphi_{i+1}(y)$, $\varphi_{i+1}(Y)[y \leftarrow \varphi'_{i+1}]$, where θ_i and θ'_i are $\Sigma_i \cup \Pi_i$ formulas, φ_{i+1} and φ'_{i+1} are Σ_{i+1} formulas, $Y \subseteq APV$, $y \in Y$, and no free variable of φ'_{i+1} is in Y . In other words, to form Σ_{i+1} , we take $\Sigma_i \cup \Pi_i$ and close under Boolean and modal operations, $\mu y.\varphi(y)$ for $\varphi \in \Sigma_{i+1}$, and substitution of a free variable of $\varphi \in \Sigma_{i+1}$ by a formula $\varphi' \in \Sigma_{i+1}$ provided that no free variable of φ' is captured by φ .
- A formula is in Π_{i+1} if it is one of the following θ_i , $\theta_i \wedge \theta'_i$, $\theta_i \vee \theta'_i$, $\Diamond\theta_i$, $\Box\theta_i$, $\nu y.\psi_{i+1}(y)$, $\psi_{i+1}(Y)[y \leftarrow \psi'_{i+1}]$, where θ_i and θ'_i are $\Sigma_i \cup \Pi_i$ formulas, ψ_{i+1} and ψ'_{i+1} are Π_{i+1} formulas, $Y \subseteq APV$, $y \in Y$, and no free variable of ψ'_{i+1} is in Y .

Note that the “substitution step” suggests that the formula $\psi = \nu y.(\Diamond(y \wedge (\mu z.p \vee \Diamond z)))$ is in both Π_2 and Σ_2 . To see that ψ is in Σ_2 (it is easy to see that $\psi \in \Pi_2$), note that $\mu z.p \vee \Diamond z$ is in Σ_1 , and hence also in Σ_2 . In addition, the formula $\nu y.(\Diamond(y \wedge x))$, for $x \in APV$, is in Π_1 , and hence also in Σ_2 . The formula $\mu z.p \vee \Diamond z$ has no free variables. Then, we can substitute x by it, get ψ , and stay in Σ_2 . Note that for classifications that do not allow such a substitution, the formula ψ is not in Σ_2 . Note also that ψ is neither in Π_1 nor Σ_1 .

Finally, we say that a formula is in Δ_i if it is one of the following $\theta_i, \theta_i \wedge \theta'_i, \theta_i \vee \theta'_i, \Diamond \theta_i, \Box \theta_i, \theta(Y)[y \leftarrow \theta'_i]$, where θ_i and θ'_i are $\Sigma_i \cup \Pi_i$ formulas, $Y \subseteq APV$, $y \in Y$, and no variable of θ'_i is in Y . In other words, to form Δ_i , we take $\Sigma_i \cup \Pi_i$ and close under Boolean and modal operations, and under substitution that does not increase the alternation depth. Note that Δ_0 is ML and Δ_1 is AFMC.

Essentially, Σ_i contains all Boolean and modal combinations of formulas in which there are at most $i - 1$ alternations of μ and ν , with the external fixpoint being a μ . Similarly, Π_i contains all Boolean and modal combinations of formulas in which there are at most i alternations of μ and ν , with the external fixpoint being a ν . A μ -calculus formula is *alternation free* if, for all atomic propositional variables y , there are no occurrences of ν (μ) on any syntactic path from an occurrence of μy (νy , respectively) to an occurrence of y . For example, the formula $\mu x.(p \vee \mu y.(x \vee EXy))$ is alternation free (and is in Σ_1) and the formula $\nu x.\mu y.((p \wedge x) \vee EXy)$ is not alternation free (and is in Π_2). The *alternation-free μ -calculus* is a subset of μ -calculus containing only alternation-free formulas. The alternation-free μ -calculus is a strict syntactic fragment of $\Pi_2 \cap \Sigma_2$. We now use Theorem 2 in order to show that $\Pi_2 \cap \Sigma_2$ is not more expressive than the alternation free μ -calculus. Thus, every formula in $\Pi_2 \cap \Sigma_2$ has an equivalent formula in AFMC.

For the alternation-free μ -calculus, an automata-theoretic characterization in terms of symmetric alternating weak automata is well known (a similar result is proven in [AN92] for directed trees):

Theorem 3. [KV98] *A set $\mathcal{T} \subseteq \text{trees}(\Sigma)$ can be expressed in AFMC iff $\mathcal{T}$ can be recognized by a symmetric weak alternating automaton.*

In [Kai95], Kaivola considered μ -calculus formulas in which the $\Diamond$ modality is parameterized with directions and translates Π_2 formulas to NBT. In order to apply Theorem 2, we should translate Π_2 formulas to symmetric monotonic NBT. For that, we first use a known translation of Π_2 formulas to symmetric ABT (Theorem 4; a similar translation for the directed case is described in [Niw86, Tak86]), and then remove alternation, with symmetry preserved (Theorem 5).

Theorem 4. [KVV00] *Given a Π_2 formula ψ , there is a symmetric alternating Büchi tree automaton $\mathcal{A}_\psi$ that accepts exactly all trees that satisfy ψ .*

Miyano and Hayashi described a translation of alternating Büchi word automata to equivalent nondeterministic Büchi word automata [MH84]. Mostowski extended the translation to tree automata [Mos84], and we extend it further to

symmetric tree automata. Since the nondeterministic automaton needs to keep track of the states visited in each path of the run tree of the alternating automaton, the symmetry of the automaton poses real technical challenges.

Theorem 5. *Let $\mathcal{A}$ be a symmetric alternating Büchi tree automaton. There is a symmetric monotonic nondeterministic Büchi tree automaton $\mathcal{A}'$, with exponentially many states, such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Proof. Let $\mathcal{A} = \langle \Sigma, S, s_{in}, \delta, \alpha \rangle$. Then $\mathcal{A}' = \langle \Sigma, Q, \{\langle s_{in}, 2 \rangle\}, \delta', \alpha' \rangle$, where

- $Q = 2^{S \times \{1,2\}}$. For a state $q \in Q$, let $q[1] = \{s : \langle s, 1 \rangle \in q\}$ and $q[2] = \{s : \langle s, 2 \rangle \in q\}$. Intuitively, the automaton $\mathcal{A}'$ guesses a run of $\mathcal{A}$. At a given node x of a run of $\mathcal{A}'$, it keeps in its memory the set of all the states of $\mathcal{A}$ that visit x in the guessed run. As it reads the next input letter, it guesses the way in which an accepting run of $\mathcal{A}$ proceeds from all of these states. This guess induces the states that the run of $\mathcal{A}'$ visit in the children of x . In order to make sure that every infinite path visits states in α infinitely often, the states are tagged by 1 or 2. States tagged by 1 correspond to copies that have already visit α , and states tagged by 2 correspond to copies that owe a visit to α . When all the copies visit α (that is, all the states are tagged by 1), we change the tag of all states to 2.
- Given $S' \subseteq S$, $\sigma \in \Sigma$, and a pair $\langle U, E \rangle$ of subsets of S , we say that $\langle U, E \rangle$ covers S' and σ if the set $\{\Box s : s \in U\} \cup \{\Diamond s : s \in E\}$ satisfies $\bigwedge_{s' \in S'} \delta(s', \sigma)$. Now, $\delta' : Q \times \Sigma \rightarrow 2^{Q \times Q}$ is defined, for all $q \in Q$ and $\sigma \in \Sigma$, as follows.
 - If $q[2] \neq \emptyset$, then $\delta'(q, \sigma)$ contains all pairs $\langle U, E \rangle$ such that there is $\langle U_1, E_1 \rangle$ that covers $q[1]$ and σ , and there is $\langle U_2, E_2 \rangle$ that covers $q[2]$ and σ , and the following hold.
 - * $U = \{\langle s, 1 \rangle : s \in U_1 \cup (U_2 \cap \alpha)\} \cup \{\langle s, 2 \rangle : s \in U_2 \setminus \alpha\}$.
 - * $E = \{\langle s, 1 \rangle : s \in E_1 \cup (E_2 \cap \alpha)\} \cup \{\langle s, 2 \rangle : s \in E_2 \setminus \alpha\}$.
 - If $q[2] = \emptyset$, then $\delta'(q, \sigma)$ contains all pairs $\langle U, E \rangle$ such that there is $\langle U_1, E_1 \rangle$ that covers $q[1]$ and σ and the following hold.
 - * $U = \{\langle s, 1 \rangle : s \in U_1 \cap \alpha\} \cup \{\langle s, 2 \rangle : s \in U_1 \setminus \alpha\}$.
 - * $E = \{\langle s, 1 \rangle : s \in E_1 \cap \alpha\} \cup \{\langle s, 2 \rangle : s \in E_1 \setminus \alpha\}$.
- $\alpha' = \{q : q[2] = \emptyset\}$. Note that a sequence of states of $\mathcal{A}$, which corresponds to the behavior of a copy of $\mathcal{A}$, changes the tag of its states from 2 to 1 when the copy visits a state in α . Also, once all the sequences change the tag of their states to 1, the attribution is changed back to 2. Thus, α' guarantees that all sequences visit α infinitely often.

It is easy to see that $\mathcal{A}$ is monotonic. Indeed, if $q \subseteq q'$, then $q[1] \subseteq q'[1]$ and $q[2] \subseteq q'[2]$. Thus, if a pair $\langle U, E \rangle$ covers $q'[1]$ and σ , then $\langle U, E \rangle$ also covers $q[1]$ and σ , and similarly for $q'[2]$ and $q[2]$. Hence, given an accepting run of $\mathcal{A}'$, we can make it an accepting run of $\mathcal{A}^q$ by changing the labels of the root from (ε, q') to (ε, q) . In addition, if $q'[2]$ is empty, so is $q[2]$.

Remark 1. A related approach for translating μ -calculus formulas into symmetric automata is taken in [JW95] (see also [AN01]). First, μ -calculus formulas are transformed into a disjunctive form. The removal of conjunctions described there is similar to the removal of universal branches in alternating tree automata (and indeed it involves the same determinization construction that is present in the automata-theoretic approach [MS87]). It is then shown that disjunctive μ -calculus formulas correspond to μ -automata. Our focus here is on the translation of Π_2 formulas to symmetric monotonic nondeterministic Büchi tree automata. It is possible to recast our proof in an extension of the framework of μ -automata [Wal03], but we find our notion of symmetric nondeterministic automata more transparent.

Theorem 6. $\Pi_2 \cap \Sigma_2 \equiv AFMC$.

Proof. Since AFMC is a syntactic fragment of $\Pi_2 \cap \Sigma_2$, one direction is trivial. Let ξ be a property expressible in $\Pi_2 \cap \Sigma_2$. Given $\theta \in \Pi_2$ expressing ξ , we can construct, by Theorems 4 and 5, a symmetric monotonic NBT $\mathcal{U}_\theta$ that accepts exactly all trees that satisfy θ . Also, $\xi \in \Sigma_2$ implies that there is $\psi \in \Pi_2$ that is equivalent to $\neg\theta$, so we can also construct a symmetric monotonic NBT $\mathcal{U}_\psi$ that accepts exactly all trees that do not satisfy θ . Clearly, $\mathcal{L}(\mathcal{U}_\psi) = \text{comp}(\mathcal{L}(\mathcal{U}_\theta))$. Hence, by Theorem 2, there is a symmetric alternating weak automaton $\mathcal{A}_\theta$ that is equivalent to $\mathcal{U}_\theta$. By Theorem 3, the automaton $\mathcal{A}_\theta$ can be translated to a formula φ in AFMC such that a tree satisfies φ iff it is accepted by $\mathcal{U}_\theta$ iff it is not accepted by $\mathcal{U}_\psi$. We claim that φ is logically equivalent to θ over arbitrary structures (in particular, structures with an infinite branching degree). To see this, assume, by way of contradiction, that φ is not logically equivalent to θ . Then, either $\theta \wedge \neg\varphi$ or $\varphi \wedge \psi$ is satisfiable in some general structure. But then, either $\theta \wedge \neg\varphi$ or $\varphi \wedge \psi$ is satisfiable by a tree model [SE84] of a finite branching degree, contradicting the fact that a tree satisfies φ iff it is accepted by $\mathcal{U}_\theta$ iff it is not accepted by $\mathcal{U}_\psi$.

Remark 2. Since it is also known that the μ -calculus has the *finite-model property* [KP84], it follows that Theorem 6 can also be relativized to finite Kripke structures.

5 Concluding Remarks

We showed that $\Sigma_2 \cap \Pi_2 \equiv AFMC$. In other words, if we can specify a property ψ both as a least fixpoint nested inside a greatest fixpoint and as a greatest fixpoint nested inside a least fixpoint, we should be able to specify ψ also with no alternation between greatest and least fixpoints. This offers an elegant characterization of alternation freedom. The key to our results is a development of a theory of *symmetric nondeterministic Büchi* tree automata. A technical outcome of this theory is that the blow-up of our construction, i.e., going from

formulas in $\Sigma_2 \cap \Pi_2$ to equivalent formulas in AFMC is doubly exponential. It would be interesting to try to improve this complexity or to prove its optimality.

Combining our result here with the result in [KV01] ($\Sigma_1 \cap \Pi_1 \equiv \text{ML}$) suggests the possibility of a general coalescence result for the μ -calculus hierarchy. Recall the definition of Δ_i as the closure of $\Sigma_i \cap \Pi_i$ under Boolean and modal operations and under alternation-preserving substitutions. Then we have that $\Sigma_i \cap \Pi_i \equiv \Delta_{i-1}$ for $i = 1, 2$. It is tempting to conjecture that this holds for all $i > 0$, in analogy for such coalescence for the quantifier alternation hierarchy of first-order logic (cf. [Add62]). As is shown, however, in [AS03], this is not the case for $i > 2$.

Acknowledgements. We are grateful to J.W. Addison for valuable discussions regarding the first-order quantifier-alternation hierarchy and to I. Walukiewicz for discussions regarding μ -automata.

References

- [Add62] J.W. Addison, The theory of hierarchies. *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 26–37, Stanford University Press, 1962.
- [AN90] A. Arnold and D. Niwiński. Fixed point characterization of Büchi automata on infinite trees. *Information Processing and Cybernetics*, 8–9:451–459, 1990.
- [AN92] A. Arnold and D. Niwiński. Fixed point characterization of weak monadic logic definable sets of trees. In *Tree Automata and Languages*, pages 159–188, Elsevier, 1992.
- [AN01] A. Arnold and D. Niwiński. *Rediments of μ -calculus*. Elsevier, 2001.
- [AS03] A. Arnold and L. Santocanale, On ambiguous classes in the μ -calculus hierarchy of tree languages, *Proc. Workshop on Fixed Points in Computer Science*, Warsaw, Poland, 2003.
- [Ben91] J. Benthem. Languages in actions: categories, lambdas and dynamic logic. *Studies in Logic*, 130, 1991.
- [Bra98] J.C. Bradfield. The modal μ -calculus alternation hierarchy is strict. *TCS*, 195(2):133–153, March 1998.
- [BRS99] R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Proc. 11th CAV*, LNCS 1633, pages 222–235. 1999.
- [CS91] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal μ -calculus. In *Proc. 3rd CAV*, LNCS 575, pages 48–58, 1991.
- [EL85] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, 1985.
- [EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ -calculus. In *Proc. 1st LICS*, pages 267–278, 1986.
- [FL79] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18:194–211, 1979.
- [GJ79] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. Freeman and Co., San Francisco, 1979.

- [Jur00] M. Jurdzinski. Small progress measures for solving parity games. In *17th STACS*, LNCS 1770, pages 290–301. 2000.
- [JW95] D. Janin and I. Walukiewicz. Automata for the modal μ -calculus and related results. In *Proc. 20th MFCS*, LNCS, pages 552–562. 1995.
- [Kai95] R. Kaivola. On modal μ -calculus and Büchi tree automata. *IPL*, 54:17–22, 1995.
- [Koz83] D. Kozen. Results on the propositional μ -calculus. *TCS*, 27:333–354, 1983.
- [KP84] D. Kozen and R. Parikh. A decision procedure for the propositional μ -calculus. In *Logics of Programs*, LNCS 164, pages 313–325, 1984.
- [KV98] O. Kupferman and M.Y. Vardi. Freedom, weakness, and determinism: from linear-time to branching-time. In *Proc. 13th LICS*, pages 81–92, June 1998.
- [KV99] O. Kupferman and M.Y. Vardi. The weakness of self-complementation. In *Proc. 16th STACS*, LNCS 1563, pages 455–466. 1999.
- [KV01] O. Kupferman and M.Y. Vardi. On clopen specifications. In *Proc. 8th LPAR*, LNCS 2250, pages 24–38. 2001.
- [KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.
- [McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
- [MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω -words. *TCS*, 32:321–330, 1984.
- [Mos84] A.W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, LNCS 208, pages 157–168. 1984.
- [MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *TCS*, 54:267–276, 1987.
- [MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th ICALP*, LNCS 226, 1986.
- [Niw86] D. Niwiński. On fixed point clones. In *Proc. 13th ICALP*, LNCS 226, pages 464–473. 1986.
- [Rab70] M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pages 1–23, 1970.
- [Rog67] H. Rogers. *Theory of recursive functions and effective computability*. McGraw-Hill, 1967.
- [SE84] R.S. Street and E.A. Emerson. An elementary decision procedure for the μ -calculus. In *Proc. 11th ICALP*, LNCS 172, pages 465–472, 1984.
- [Tak86] M. Takahashi. The greatest fixed-points and rational ω -tree languages. *TCS* 44, pp. 259–274, 1986.
- [VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pages 332–344, 1986.
- [Wal03] I. Walukiewicz. Private communication, 2003.
- [Wil99] T. Wilke. CTL^+ is exponentially more succinct than CTL . In *Proc. 19th FST&TCS*, LNCS 1738, pages 110–121, 1999.

Upper Bounds for a Theory of Queues

Tatiana Rybina and Andrei Voronkov

University of Manchester
{rybina,voronkov}@cs.man.ac.uk

Abstract. We prove an upper bound result for the first-order theory of a structure $\mathbf{W}$ of queues, i.e. words with two relations: addition of a letter on the left and on the right of a word. Using complexity-tailored Ehrenfeucht games we show that the witnesses for quantified variables in this theory can be bound by words of an exponential length. This result, together with a lower bound result for the first-order theory of two successors [6], proves that the first-order theory of $\mathbf{W}$ is complete in $\text{LATIME}(2^{O(n)})$: the class of problems solvable by alternating Turing machines running in exponential time but only with a linear number of alternations.

1 Introduction

Theories of words are fundamental to computer science. Decision procedures for various theories of words are used in many areas of computing, for example in verification. Closely related to words are *queues* which can be regarded as words with two operations: deleting a letter on the left and adding a letter on the right. In this paper we prove upper bounds on the complexity of the first-order theory of queues. The upper bound is tight, i.e., it coincides with the respective lower bound up to a constant factor.

Denote by $\{0,1\}^*$ the set of all words over the finite alphabet $\{0,1\}$, by $\ln(w)$ the length of the word w and by λ the empty word. We call the elements of $\{0,1\}^*$ simply *words*. By “.” we denote concatenation of words. Define the following four relations on words:

$$\begin{aligned} l_0(a, b) &\leftrightarrow b = 0 \cdot a; & l_1(a, b) &\leftrightarrow b = 1 \cdot a; \\ r_0(a, b) &\leftrightarrow b = a \cdot 0; & r_1(a, b) &\leftrightarrow b = a \cdot 1. \end{aligned}$$

The first-order structure $\mathbf{W} = \langle \{0,1\}^*, r_0, r_1, l_0, l_1 \rangle$ is called the *queue structure*. The *first-order theory of queues* is the first-order theory of $\mathbf{W}$.

Let us formulate the main result of this paper. See [11,10] for the precise definition of the *complexity class* $\text{LATIME}(2^{O(n)})$: it is the class of problems solvable by alternating Turing machines running in time $2^{O(n)}$ but only with a linear number of alternations. Of course, for this class polynomial-time or LOGSPACE reductions are too coarse, this class is closed with respect to LOGLIN-reductions [11], i.e., LOGSPACE reductions giving at most linear increase in length. The main result of this paper is the following.

THEOREM 1 *The first-order theory of $\mathbf{W}$ is complete in $\text{LATIME}(2^{O(n)})$ with respect to LOGLIN -reductions. $\square$*

This theorem will be proved using complexity-tailored Ehrenfeucht games. We will show that in the first-order theory of $\mathbf{W}$ in every sentence witnesses for quantified variables can be bound by words of the size exponential in the size of the sentence.

The decidability of the first-order theory of this structure follows from the decidability of the first-order theory of two successors with the predicates of equal length and prefix [9]. It also immediately follows from the fact that this structure is automatic [7,4].

A lower bound on the first-order theory of $\mathbf{W}$ can be derived from the lower bound on the first-order theory of two successors, i.e., of the structure $\{0, 1\}^*, r_0, r_1$, proved in [11] based on [6] (a technique for proving lower bounds is also described in [5], some simple generalizations can be found in [12]). Expressive power of several theories of words, including the first-order theory of $\mathbf{W}$, is discussed in [1].

For us the main motivation for these results was our case study of verification of a protocol with queues. Verification with queues was also extensively studied in [3,2]. In [8] we proved that first-order theories of some structures containing trees and queues are decidable. Our results were based on quantifier elimination and imply a non-elementary upper bound (a non-elementary lower bound also follows for the theory of trees [6]). However, if we consider a theory with queues only, it was clear that a non-elementary upper bound could be avoided. Indeed, the quantifier elimination arguments of [8] show that the main difference in expressive power between queues and stacks is periodicity constraints. However, these periodicity constraints, though they can express “deep” properties of queues (e.g., that all elements of a queue are 0’s), still cannot distinguish queues which are indistinguishable by their “short” prefixes. Motivated by this observation we undertake a characterization of the exact complexity of the first-order theory of $\mathbf{W}$.

In the proof of the upper bound for $\mathbf{W}$ we show, like [6], that all quantifiers in a formula can be replaced by quantifiers of an exponential size. However, our arguments are more technically involved. Moreover, some lemmas of [6] do not hold any more in this context.

2 Ehrenfeucht Games

$\mathbb{N}$ denotes the set of natural numbers. By $\bar{a}_k$ we denote the sequence $a_1, \dots, a_k$ of k elements, and similar for other letters instead of a .

DEFINITION 2 (Norm) Let A be a structure. A *norm* on A , denoted $\|\cdot\|$, is a function from the domain of the structure A to $\mathbb{N}$. For an element a of A we write $\|a\|$ to denote the norm of a . $\square$

The following definitions are similar to those of Ferrante and Racko [6].

DEFINITION 3 (Ehrenfeucht Equivalence) Let $n, k \in \mathbb{N}$, A be a structure, and $\bar{a}_k, \bar{b}_k$ be sequences of elements of A . Then we write $\bar{a}_k \equiv_{n,k} \bar{b}_k$ if for all formulas $F(x_1, \dots, x_k)$ of quantifier depth at most n , $\bar{a}_k$ satisfies $F(x_1, \dots, x_k)$ in A if and only if $\bar{b}_k$ satisfies $F(x_1, \dots, x_k)$ in A . (In particular $\bar{a}_k \equiv_{0,k} \bar{b}_k$ means that $\bar{a}_k$ and $\bar{b}_k$ satisfy the same quantifier-free formulas.) $\square$

DEFINITION 4 (Boundedness) Let A be a structure with a norm $\|\cdot\|$ on it and $H : \mathbb{N}^3 \rightarrow \mathbb{N}$ be a function. We say that A is H -bounded if for all natural numbers k, n, m , a sequence $\bar{a}_k$ of elements of A , and formula $F(x_1, \dots, x_{k+1})$ of quantifier depth $\leq n$ the following property holds. If for all $i \leq k$ we have $\|a_i\| \leq m$ and $A \models x_{k+1}F(\bar{a}_k, x_{k+1})$, then there exists $a_{k+1} \in A$ such that $\|a_{k+1}\| \leq H(n, k, m)$ and $A \models F(\bar{a}_k, a_{k+1})$. $\square$

THEOREM 5 (Ferrante and Racko [6]) Let A be a structure and $H : \mathbb{N}^3 \rightarrow \mathbb{N}$ be a function. Let $E_{n,k}$ be relations such that for all natural numbers n, k, m and sequences of elements $\bar{a}_k, \bar{b}_k$ of A the following properties are true:

1. if $E_{0,k}(\bar{a}_k, \bar{b}_k)$ then $\bar{a}_k \equiv_{0,k} \bar{b}_k$;
2. if $E_{n+1,k}(\bar{a}_k, \bar{b}_k)$ and for all $i \leq k$ we have $\|b_i\| \leq m$ then for all $a_{k+1} \in A$ there exists $b_{k+1} \in A$ such that $E_{n,k+1}(\bar{a}_{k+1}, \bar{b}_{k+1})$ and $\|b_{k+1}\| \leq H(n, k, m)$.

Then:

1. $E_{n,k}(\bar{a}_k, \bar{b}_k) \iff \bar{a}_k \equiv_{n,k} \bar{b}_k$ for all $n, k \in \mathbb{N}$,
2. the structure A is H -bounded. $\square$

3 Main Argument

An Ehrenfeucht game decision procedure for $\mathbf{W}$ consists of defining a set of equivalence relations $E_{n,k}$, which turn out to be refinements of the relations $\equiv_{n,k}$, defining a function $H(n, k, m)$, and showing H -boundedness of the structure $\mathbf{W}$. Since the structure is H -bounded, then the witnesses for quantifiers in formulas can be restricted by elements of a fixed depth. If the number of elements of every norm is finite, then we obtain a decision procedure.

Let a, v_1, v_2 be words. By $v_1[a]_{v_2}$ we denote the word w , if it exists, such that $a = v_1 \cdot w \cdot v_2$.

In the sequel we will extensively use partial functions. Let us make a notational convention about their use.

CONVENTION 6 Let e, e_1, e_2 be any expressions over words. We write $e \downarrow$ to denote that e exists. When we write $e_1 = e_2$, we mean that both e_1 and e_2 are defined and equal. If S is a set of words and we write $e \in S$, we mean that e is defined and e is a member of S . $\square$

DEFINITION 7 (ε -word, ε -length, ε -correction) Let $\varepsilon \in \mathbb{N}$. A number $n \in \mathbb{N}$ is said to be an ε -length if $n \leq \varepsilon$. We will normally use this terminology when we speak about lengths of words. A word is an ε -word if its length is an ε -length. An ε -correction is a partial function α such that for some ε -words v_1, v_2, w_1, w_2 and for all words a we have $\alpha(a) = w_2 \cdot w_1[a]_{v_1} \cdot v_2$. An ε -correction α is called *trivial* if for some words v, w and all words a we have $\alpha(a) = w \cdot w[a]_v \cdot v$. $\square$

Let us note some useful properties of this definition.

LEMMA 8 *The following statements hold for ε -corrections.*

1. *If a is an ε_1 -word and b is ε_2 -word, then $a \cdot b$ is an $(\varepsilon_1 + \varepsilon_2)$ -word.*
2. *If a is a ε_1 -word and α is an ε_2 -correction, then $\alpha(a)$ is an $(\varepsilon_1 + 2\varepsilon_2)$ -word.*
3. *For every ε -correction α there exists an ε -correction inverse to α , denoted α^{-1} , such that for every word a we have*
 - a) *if $\alpha(a) \downarrow$ then $\alpha^{-1}(\alpha(a)) = a$;*
 - b) *if $\alpha(a) = b$ then $a = \alpha^{-1}(b)$ and $\alpha(\alpha^{-1}(b)) = b$;*
 - c) *if $\alpha^{-1}(a) = b$ then $a = \alpha(b)$ and $\alpha^{-1}(\alpha(b)) = b$.*
4. *If α is an ε_1 -correction, β is an ε_2 -correction, and $\alpha(\beta(v))$ is defined for at least one word v , then their composition $\alpha\beta$ is an $(\varepsilon_1 + \varepsilon_2)$ -correction.* $\square$

In the sequel we will often use this lemma implicitly.

For every word a , denote $a^n = \{a^n \mid n \in \mathbb{N}\}$.

LEMMA 9 (see [8]) *Let a, b, c be words such that $a \cdot b = b \cdot c$. Then*

1. *if $a = c$ then there exist words a_1 and a_2 such that $a = a_1 \cdot a_2$, $c = a_2 \cdot a_1$ and $b \in \{a^n \cdot a_1 \mid n \in \mathbb{N}\}$;*
2. *if $a \neq c$ then there exists a word s such that $c \in s$ and $b \in s$.* $\square$

LEMMA 10 *For every non-trivial ε -correction and word a , if $\alpha(a) = a$, then for some ε -word s and ε -correction γ we have $\gamma(a) \in s$.*

PROOF. By the definition of α there exist ε -words w_1, w_2, v_1, v_2 such that $\alpha(a) = w_2 \cdot w_1[a]_{v_1} \cdot v_2$. On the other hand, we have $a = w_1 \cdot w_1[a]_{v_1} \cdot v_1$. Thus $w_2 \cdot w_1[a]_{v_1} \cdot v_2 = w_1 \cdot w_1[a]_{v_1} \cdot v_1$. Since α is non-trivial, we have $w_1 = w_2$ and $v_1 = v_2$. The equality $\alpha(a) = a$ implies that either $\ln(w_2) < \ln(w_1)$, $\ln(v_2) > \ln(v_1)$, or $\ln(w_2) > \ln(w_1)$, $\ln(v_2) < \ln(v_1)$. Let us consider the case $\ln(w_2) < \ln(w_1)$, $\ln(v_2) > \ln(v_1)$, the other case is similar. In this case there exist words b and c such that $w_1 = w_2 \cdot b$ and $v_2 = c \cdot v_1$. Since w_1, v_2 are ε -words, b, c must be ε -words too. We have $w_2 \cdot w_1[a]_{v_1} \cdot c \cdot v_1 = w_2 \cdot b \cdot w_1[a]_{v_1} \cdot v_1$, hence $b \cdot w_1[a]_{v_1} = w_1[a]_{v_1} \cdot c$. By Lemma 9 there exist words s_1 and s_2 such that $b = s_1 \cdot s_2$, $c = s_2 \cdot s_1$ and $w_1[a]_{v_1} \in \{(s_1 \cdot s_2)^n \cdot s_1 \mid n \in \mathbb{N}\}$. Evidently, s_1, s_2 are ε -words. Define $s = b$ and define γ as follows: for all v we have

$$\gamma(v) \stackrel{\text{def}}{=} \lambda \cdot w_2[v]_{v_1} \cdot s_2.$$

The property $\gamma(a) \in s$ is not hard to check. $\square$

LEMMA 11 Let b, c be ε -words, α, β be ε -corrections, and a be an arbitrary word.

1. If $\alpha(a) = b$ then for all $w = b$ such that $\ln(w) \geq 2\varepsilon$ we have $\alpha^{-1}(w) \downarrow$.
2. If $\ln(a) \geq 4\varepsilon$, $\alpha(a) = b$, and $\beta(a) = c$ then for all words $w = b$ and $v = c$ such that $\ln(w), \ln(v) \geq 4\varepsilon$ we have $\beta(\alpha^{-1}(w)) = c$ and $\alpha(\beta^{-1}(v)) = b$. $\square$

The proof is straightforward but tedious.

The following definition of *indistinguishability* is the main technical notion of this paper. Define the following function L of two integer arguments: $L(n, k) = 2^{3n+k}$.

DEFINITION 12 (Indistinguishability) Let $\bar{a}_k$ and $\bar{b}_k$ be sequences of words and n be a natural number. We say that $\bar{a}_k$ and $\bar{b}_k$ are $E_{n,k}$ -indistinguishable, denoted $\bar{a}_k E_{n,k} \bar{b}_k$, if the following conditions hold for all $i, j \in \{1, \dots, k\}$. Let $\varepsilon = L(n, k)$.

1. For every ε -correction α we have $\alpha(a_i) = a_j$ if and only if $\alpha(b_i) = b_j$.
2. If either a_i or b_i is a 4ε -word, then $a_i = b_i$.
3. For every ε -correction α and ε -word a , $\alpha(a_i) = a$ if and only if $\alpha(b_i) = a$.

Prefix (respectively suffix) of the length ε of a word a , if it exists, is denoted $\text{prefix}(\varepsilon, a)$ (respectively $\text{suffix}(\varepsilon, a)$).

LEMMA 13 Let $\bar{a}_k E_{n,k} \bar{b}_k$. Define $\varepsilon = L(n, k)$. Then for every i

1. either $a_i = b_i$, or $\text{prefix}(\varepsilon, a_i) = \text{prefix}(\varepsilon, b_i)$ and $\text{suffix}(\varepsilon, a_i) = \text{suffix}(\varepsilon, b_i)$;
2. for every ε -correction α , $\alpha(a_i) \downarrow$ if and only if $\alpha(b_i) \downarrow$.

PROOF. The second clause evidently follows from the first one, so we will only prove the first clause.

If $\ln(a_i) \leq 4\varepsilon$ then, by Clause 2 of Definition 12, $a_i = b_i$. Otherwise we have $\ln(a_i) > 4\varepsilon$. Define an ε -correction α by

$$\alpha(v) \stackrel{\text{def}}{=} \text{prefix}(\varepsilon, a_i) \cdot \text{suffix}(\varepsilon, a_i)[v]_{\text{suffix}(\varepsilon, a_i)} \cdot \text{suffix}(\varepsilon, a_i).$$

It is easy to see that $\alpha(a_i) = a_i$, hence, by Clause 1 of Definition 12, $\alpha(b_i) = b_i$. Then $\alpha(b_i)$ is defined, hence $\text{prefix}(\varepsilon, b_i) = \text{prefix}(\varepsilon, a_i)$ and $\text{suffix}(\varepsilon, b_i) = \text{suffix}(\varepsilon, a_i)$. $\square$

By routine inspection of the definition of $E_{n,k}$, we can also prove the following result.

COROLLARY 14 $E_{n,k}$ is an equivalence relation. $\square$

The following lemma is a key to proving that $\mathbf{W}$ is H -bounded.

LEMMA 15 Let k, n be natural numbers and $\bar{a}_k, \bar{b}_k$ be sequences of words such that $\bar{a}_k E_{n+1,k} \bar{b}_k$. Then for every word a_{k+1} there exists a word b_{k+1} such that $\bar{a}_{k+1} E_{n,k+1} \bar{b}_{k+1}$.

PROOF. Let $\varepsilon = L(n, k + 1)$. In the proof we will construct the word b_{k+1} and prove $E_{n,k+1}$ -indistinguishability of $\bar{a}_{k+1}$ and $\bar{b}_{k+1}$ using the hypothesis about $E_{n+1,k}$ -indistinguishability of $\bar{a}_k$ and $\bar{b}_k$. In this respect note that $L(n + 1, k) = 4 \cdot L(n, k + 1)$. Therefore, in the proof we will use hypothesis about 4ε -words and prove statements about ε -words.

Let us note that while verifying Clauses 1–3 of Definition 12 for $\bar{a}_{k+1}$ and $\bar{b}_{k+1}$ we have to consider only the case $i = k + 1$ or $j = k + 1$ for Clause 1 and the case $i = k + 1$ for Clauses 2–3. Moreover, for Clause 1 the proofs for the case $i = k + 1$ are similar to the proofs for the case $j = k + 1$, so we will only consider the case $i = k + 1$.

Our choice of b_{k+1} depends on the properties of a_{k+1} , so we proceed by cases.

Case 1: a_{k+1} is a 4ε -word. We choose $b_{k+1} = a_{k+1}$.

Let us prove Clauses 1–3 of Definition 12 for $\bar{a}_{k+1}$ and $\bar{b}_{k+1}$.

1. Suppose α is an ε -correction $\alpha(a_i) = a_{k+1}$. We have to prove $\alpha(b_i) = b_{k+1}$. We only verify the case $i \leq k$ since the case $i = k + 1$ is trivial. We know that a_{k+1} is 4ε -word and $\alpha^{-1}(a_{k+1}) = a_i$. Then a_i is 6ε -word. By the hypothesis, if a_i is 16ε -word, then $a_i = b_i$. Therefore, $\alpha(b_i) = b_{k+1}$.
2. We have to prove that if a_{k+1} is a 4ε -word or b_{k+1} is a 4ε -word, then $a_{k+1} = b_{k+1}$. But we have $a_{k+1} = b_{k+1}$ by our construction.
3. By our choice $a_{k+1} = b_{k+1}$, therefore for every ε -correction α and ε -word a : $\alpha(a_{k+1}) = a$ if and only if $\alpha(b_{k+1}) = a$.

Case 2: a_{k+1} is not a 4ε -word but there exist $j \leq k$ and ε -correction β such that $\beta(a_j) = a_{k+1}$. By Lemma 13, $\beta(b_j)$ is defined. We choose $b_{k+1} = \beta(b_j)$. Let us show that our choice of b_{k+1} satisfies the definition of $E_{n,k+1}$ -indistinguishability.

1. Suppose that α is an ε -correction and $i \leq k + 1$. We need to verify that $\alpha(a_i) = a_{k+1}$ if and only if $\alpha(b_i) = b_{k+1}$.
To prove the “only if” direction, suppose $\alpha(a_i) = a_{k+1}$. Since $a_{k+1} = \beta(a_j)$, we have $\beta^{-1}(a_{k+1}) = a_j$, hence $\beta^{-1}(\alpha(a_i)) = a_j$. Consider two cases: $i = k + 1$ and $i = k + 1$.
Suppose $i = k + 1$. Since $\beta^{-1}\alpha$ is a 2ε -correction, by the hypothesis we have $\beta^{-1}(\alpha(b_i)) = b_j$. This implies $\alpha(b_i) = \beta(b_j) = b_{k+1}$.
Now suppose that $i = k + 1$, then $\alpha(a_{k+1}) = a_{k+1}$, that is $\alpha(\beta(a_j)) = \beta(a_j)$, hence $\beta^{-1}(\alpha(\beta(a_j))) = a_j$. By the hypothesis, since $\beta^{-1}\alpha\beta$ is a 3ε -correction, $\beta^{-1}(\alpha(\beta(b_j))) = b_j$, hence $\alpha(\beta(b_j)) = \beta(b_j)$. But $\beta(b_j) = b_{k+1}$, so $\alpha(b_{k+1}) = b_{k+1}$.
The “if” direction is similar.
2. Since a_{k+1} is not a 4ε -word to verify Clause 2 we have to show that b_{k+1} is not a 4ε -word. By our choice of b_{k+1} , $\beta^{-1}(b_{k+1}) = b_j$. Suppose that b_{k+1} is a 4ε -word, then b_j is a 6ε -word, so by our hypothesis $a_j = b_j$. Therefore, $\beta(a_j) = \beta(b_j)$, that is $a_{k+1} = b_{k+1}$. But then a_{k+1} would be a 4ε -word. Contradiction.

3. To verify Clause 3 we only have to show that for every ε -word a and every ε -correction α the following holds:

$$\alpha(a_{k+1}) \cdot a \leftrightarrow \alpha(b_{k+1}) \cdot a.$$

Suppose that $\alpha(a_{k+1}) \cdot a$, then $\alpha(\beta(a_j)) \cdot a$ and $\alpha\beta$ is a 2ε -correction. By the hypothesis, we have $\alpha(\beta(b_j)) \cdot a$, that is $\alpha(b_{k+1}) \cdot a$.

Case 3: a_{k+1} is not a 4ε -word and there are no $j \leq k$ and ε -correction α such that $\alpha(a_j) = a_{k+1}$ but there exist an ε -correction γ and an ε -word c such that $\gamma(a_{k+1}) = c$.

If $\gamma(a_{k+1})$ is a 4ε -word, then a_{k+1} is a 5 -word and we can choose $b_{k+1} = a_{k+1}$ and repeat the proof of Case 1.

Suppose that $\gamma(a_{k+1})$ is not a 4ε -word. Let n be a natural number such that

$$\ln(c^{-n}) \leq \max(6\varepsilon, 4\varepsilon + \max_{i \leq k} \ln(b_i)) < \ln(c).$$

Then we choose $b_{k+1} = \gamma^{-1}(c)$ (notice that $\ln(c) > 4\varepsilon$ and hence by Lemma 11 $\gamma^{-1}(c)$ is defined).

Let us prove some simple estimations on the length of b_{k+1} . Note that by our definition for all $i \leq k$ we have $\ln(c) - \ln(b_i) > 4\varepsilon$. Since b_{k+1} is an ε -correction of c , this implies $\ln(b_{k+1}) - \ln(b_i) > 2\varepsilon$. In a similar way we can establish

$$\ln(b_{k+1}) < \max(9\varepsilon, 7\varepsilon + \max_{i \leq k} \ln(b_i)). \quad (1)$$

Let us prove Clauses 1–3 of Definition 12 for $\bar{a}_{k+1}$ and $\bar{b}_{k+1}$.

1. Let α be an ε -correction. We have to prove that $\alpha(a_i) = a_{k+1}$ if and only if $\alpha(b_i) = b_{k+1}$. Consider two cases: $i \leq k$ and $i = k + 1$.
Let $i \leq k$. By the assumption $\alpha(a_i) = a_{k+1}$, so we have to prove $\alpha(b_i) = b_{k+1}$. Suppose, by contradiction, $\alpha(b_i) = b_{k+1}$. Then $\ln(b_{k+1}) - \ln(b_i) \leq 2\varepsilon$ which contradicts to $\ln(b_{k+1}) - \ln(b_i) > 2\varepsilon$.
Let $i = k + 1$. We have to prove that $\alpha(a_{k+1}) = a_{k+1}$ if and only if $\alpha(b_{k+1}) = b_{k+1}$. Suppose that $\alpha(a_{k+1}) = a_{k+1}$. By assumption, we have $\gamma(a_{k+1}) = c$, i.e. there exists natural number z such that $\gamma(a_{k+1}) = c^z$. Without loss of generality we assume that c is non-periodic. Then $\alpha(\gamma^{-1}(c^z)) = \gamma^{-1}(c^z)$. This implies $\gamma(\alpha(\gamma^{-1}(c^z))) = c^z$. It is not hard to argue $\gamma\alpha\gamma^{-1}$ is a 2ε -correction. Since $\gamma(\alpha(\gamma^{-1}(c^z))) = c^z$, $\gamma\alpha\gamma^{-1}$ either is a trivial correction or for all $w \in c$ there exists $z_1 \in \mathbb{N}$ such that $\ln(c^{z_1}) \leq 2\varepsilon$ and $\gamma\alpha\gamma^{-1}(w) = \lambda \cdot c^{z_1}[w]_\lambda \cdot c^{z_1}$. Thus $\gamma\alpha\gamma^{-1}(c) \downarrow$. It is now easy to see that $\gamma\alpha\gamma^{-1}(c) = c$. In the other direction the proof is similar.
2. Since $\ln(a_{k+1}) > 4\varepsilon$ and $\ln(b_{k+1}) > 4\varepsilon$ there is no need to verify Clause 2.
3. Suppose that $\alpha(a_{k+1}) \cdot a$ for some ε -correction α and ε -word a . We have to show $\alpha(b_{k+1}) \cdot a$. Since $\gamma(a_{k+1}) = c$, by Lemma 11, we have $\alpha(\gamma^{-1}(c)) \cdot a$.

Case 4: a_{k+1} is not a 4ϵ -word, there are no $j \leq k$ and ϵ -correction α such that $\alpha(a_j) = a_{k+1}$ and for every ϵ -correction α and ϵ -word a : $\alpha(a_{k+1}) \neq a$. Define the set of words:

$$W = \{\text{prefix}(\epsilon, a_{k+1}) \cdot q \cdot \text{suffix}(\epsilon, a_{k+1}) \mid \ln(q) = 2\epsilon + 1\}.$$

Note that $|W| = 2^{2\epsilon+1}$. It is not hard to argue that for all $c, d \in W$ and ϵ -corrections α, β the following holds:

$$\alpha(c) = \beta(d) \leftrightarrow c = d.$$

Therefore for every $i \leq k$ there exists at most one element $c \in W$ which can be obtained by an ϵ -correction from b_i .

Let us count the number of words $w \in W$ such that for some ϵ -correction α and ϵ -word a we have $\alpha(a) = w$. It is not hard to argue that the number of such words is not greater than the number of ϵ -words, that is $2^{\epsilon+1}$.

Now define the following set of words:

$$W' = \{d \in W \mid \text{for all } i \leq k, \epsilon\text{-words } a \text{ and } \epsilon\text{-corrections } \beta : \beta(b_i) = d \text{ and } \beta(d) \neq a\}.$$

Let us prove that W' is non-empty. Indeed, W' is obtained from W by removing all ϵ -corrections of the words b_i and all ϵ -corrections of words belonging to some a , where a is an ϵ -word. Therefore, the cardinality of W' is at least $2^{2\epsilon+1} - k - 2^{\epsilon+1}$. We have

$$2^{2\epsilon+1} - k - 2^{\epsilon+1} > 2^{2\epsilon+1} - 2^{\epsilon+2} \geq 0,$$

so W' contains at least one element. Choose b_{k+1} to be any element of W' . Let us check that our choice of b_{k+1} satisfies the definition of $E_{n,k+1}$ -indistinguishability.

1. Let α be an ϵ -correction. By our assumption, for every $j \leq k$ we have $\alpha(a_j) = a_{k+1}$. By our construction of b_{k+1} we have $\alpha(b_j) = b_{k+1}$. So it remains to check that $\alpha(a_{k+1}) = a_{k+1}$ if and only if $\alpha(b_{k+1}) = b_{k+1}$. If α is trivial, then this property is straightforward, so assume that α is non-trivial. If $\alpha(a_{k+1}) \neq a_{k+1}$, then by Lemma 10 for some ϵ -word a and ϵ -correction β we would have $\beta(a_{k+1}) = a$. This would contradict to our assumption, so we have $\alpha(a_{k+1}) = a_{k+1}$. Then we have to prove $\alpha(b_{k+1}) = b_{k+1}$. Suppose, by contradiction, $\alpha(b_{k+1}) \neq b_{k+1}$. Then by Lemma 10 for some ϵ -word a and ϵ -correction β we would have $\beta(b_{k+1}) = a$. But this is impossible since $b_{k+1} \in W'$.
2. Since $\ln(a_{k+1}) > 4\epsilon$ and $\ln(b_{k+1}) > 4\epsilon$ there is no need to verify Clause 2.
3. We have to show that for every ϵ -correction α and ϵ -word a we have $\alpha(b_{k+1}) \neq a$. This is immediate by our choice of b_{k+1} .

The proof of Lemma 15 is completed. □

LEMMA 16 *For all natural numbers k, n , all sequences of words $\bar{a}_k$ and $\bar{b}_k$, if $\bar{a}_k E_{n+1,k} \bar{b}_k$ then for every word a_{k+1} there exists word b_{k+1} such that $\bar{a}_{k+1}$ and $\bar{b}_{k+1}$ are $E_{n,k+1}$ -indistinguishable and either*

1. $\ln(b_{k+1}) \leq 9 \cdot 2^{3n+k}$, or
2. for some $i \leq k$, $\ln(b_{k+1}) \leq \ln(b_i) + 7 \cdot 2^{3n+k}$.

PROOF. By routine inspection of the proof of Lemma 15. These bounds appear from (1), other parts of the proof give lower bounds. $\square$

For $w \in \{0, 1\}^*$ and $n, k, m \in \mathbb{N}$, define $\|w\| = \ln(w)$ and $H(n, k, m) = m + 9 \cdot 2^{3n+k}$.

LEMMA 17 *For all natural numbers k, n, m , all sequences of words $\bar{a}_k$ and $\bar{b}_k$, if $\bar{a}_k E_{n+1,k} \bar{b}_k$ and $\|b_i\| \leq m$ for all $i \leq k$ then for every word a_{k+1} there exists word b_{k+1} such that $\bar{a}_{k+1}$ and $\bar{b}_{k+1}$ are $E_{n,k+1}$ -indistinguishable and $\|b_{k+1}\| \leq H(n, k, m)$.* $\square$

This lemma proves the second conditions of Theorem 5, to prove the first condition note the following result.

LEMMA 18 *Let $\bar{a}_k, \bar{b}_k$ be sequences of words such that $\bar{a}_k E_{0,k} \bar{b}_k$. Then $\bar{a}_k \equiv_{0,k} \bar{b}_k$.*

PROOF. Since $\bar{a}_k E_{0,k} \bar{b}_k$, then for all $i, j \leq k$ the following equivalences hold:

$$\begin{aligned} \lambda[a_i]_0 = a_j &\leftrightarrow \lambda[b_i]_0 = b_j; \lambda[a_i]_1 = a_j \leftrightarrow \lambda[b_i]_1 = b_j; \\ 0[a_i]_\lambda = a_j &\leftrightarrow 0[b_i]_\lambda = b_j; 1[a_i]_\lambda = a_j \leftrightarrow 1[b_i]_\lambda = b_j. \end{aligned}$$

Thus

$$\begin{aligned} r_0(a_j, a_i) &\leftrightarrow r_0(b_j, b_i); r_1(a_j, a_i) \leftrightarrow r_1(b_j, b_i); \\ l_0(a_j, a_i) &\leftrightarrow l_0(b_j, b_i); l_1(a_j, a_i) \leftrightarrow l_1(b_j, b_i). \end{aligned}$$

Using Definition 3, we conclude $\bar{a}_k \equiv_{0,k} \bar{b}_k$. $\square$

4 Main Results

Lemma 17 and Lemma 18 prove the conditions for Theorem 5. Therefore, by this theorem we have the following key result.

THEOREM 19 *For all $n, k, m \in \mathbb{N}$:*

1. for all sequences of words $\bar{a}_k$ and $\bar{b}_k$, if $\bar{a}_k$ and $\bar{b}_k$ are $E_{n,k}$ -indistinguishable then $\bar{a}_k \equiv_{n,k} \bar{b}_k$ for all $n, k \in \mathbb{N}$,
2. the structure $\mathbf{W}$ is H -bounded. $\square$

Let us extend the first-order language by bounded quantifiers $(\forall v \leq C)$ and $(\exists v \leq C)$ for all natural numbers C with the following interpretation: $(\forall v \leq C)A(v)$ holds if there exists a C -word such v that $A(v)$, and similar for $(\exists v \leq C)$.

LEMMA 20 *Let $Q_1x_1 \dots Q_nx_n F(\bar{x}_n)$ be a sentence such that $Q_i \in \{\forall, \exists\}$ and $F(\bar{x}_n)$ is quantifier-free. Let $C = 9 \cdot 2^{3n+1}$. Then*

$$W \models Q_1x_1 \dots Q_nx_n F(\bar{x}_n) \leftrightarrow W \models Q_1x_1 \leq C \dots Q_nx_n \leq C F(\bar{x}_n). \quad (2)$$

PROOF. Define $C_1 = 9 \cdot 2^{3n}$ and for all $i > 1$, $C_{i+1} = C_i + 9 \cdot 2^{3(n-i)+i}$. It follows from Theorem 19 that each of the quantifiers Q_ix_i can be equivalently replaced by $(Q_ix_i \leq C_i)$. It is not hard to argue that $C_i < C$ for all i , which proves (2). $\square$

Now we can prove our main result: Theorem 1.

PROOF (of Theorem 1). Recall that we have to prove that the first-order theory of W is complete in $\text{LATIME}(2^{O(n)})$. It is known that the first-order theory of W is $\text{LATIME}(2^{O(n)})$ -hard already for formulas without the relations l_0, l_1 (see [6,11,10,12], so we should prove that the first-order theory of W belongs to the class $\text{LATIME}(2^{O(n)})$. This can be proved by the following procedure running in exponential time by alternating Turing machines with a linear number of alternations: first, using Lemma 20, replace all quantifiers by quantifiers bound by words of length $2^{O(n)}$, and then “guess” the corresponding words using alternating Turing machines. The number of alternations is less than the number of quantifiers in the formula, and is therefore at most linear in n . $\square$

Acknowledgments. We thank Bakhadyr Khousainov, Leonid Libkin, and Wolfgang Thomas for helpful remarks related to the first-order theory of W .

References

1. M. Benedikt, L. Libkin, T. Schwentick, and L. Segoufin. A model-theoretic approach to regular string relations. In *Proc. 16th Annual IEEE Symposium on Logic in Computer Science, LICS 2001*, pages 431–440, 2001.
2. N.S. Bjørner. *Integrating Decision Procedures for Temporal Verification*. PhD thesis, Computer Science Department, Stanford University, 1998.
3. N.S. Bjørner. Reactive verification with queues. In *ARO/ONR/NSF/DARPA Workshop on Engineering Automation for Computer-Based Systems*, pages 1–8, Carmel, CA, 1998.
4. A. Blumensath and E. Grädel. Automatic structures. In *Proc. 15th Annual IEEE Symp. on Logic in Computer Science*, pages 51–62, Santa Barbara, California, June 2000.
5. K.J. Compton and C.W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Annals of Pure and Applied Logic*, 48:1–79, 1990.

6. J. Ferrante and C.W. Racko. *The computational complexity of logical theories*, volume 718 of *Lecture Notes in Mathematics*. Springer-Verlag, 1979.
7. B. Khoussainov and A. Nerode. Automatic presentations of structures. In Daniel Leivant, editor, *Logic and Computational Complexity, International Workshop LCC '94*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer Verlag, 1995.
8. T. Rybina and A. Voronkov. A decision procedure for term algebras with queues. *ACM Transactions on Computational Logic*, 2(2):155–181, 2001.
9. W. Thomas. Infinite trees and automaton definable relations over omega-words. *Theoretical Computer Science*, 103(1):143–159, 1992.
10. H. Volger. A new hierarchy of elementary recursive decision problems. *Methods of Operations Research*, 45:509–519, 1983.
11. H. Volger. Turing machines with linear alternation, theories of bounded concatenation and the decision problem of first order theories (Note). *Theoretical Computer Science*, 23:333–337, 1983.
12. S. Vorobyov and A. Voronkov. Complexity of nonrecursive logic programs with complex values. In *PODS'98*, pages 244–253, Seattle, Washington, 1998. ACM Press.

Degree Distribution of the FKP Network Model

Noam Berger¹, Béla Bollobás^{2,3}, Christian Borgs⁴, Jennifer Chayes⁴, and Oliver Riordan³

¹ Department of Statistics, University of California, Berkeley, CA 94720 [‡]

² Department of Mathematical Sciences, University of Memphis,
Memphis TN 38152 [§]

³ Trinity College, Cambridge CB2 1TQ, UK, and Royal Society research fellow
Department of Pure Mathematics, Cambridge [¶]

⁴ Microsoft Research, One Microsoft Way, Redmond, WA 98122.

noam@stat.berkeley.edu, {B.Bollobas,O.M.Riordan}@dpmms.cam.ac.uk,
{borgs,jchayes}@microsoft.com

Abstract. Recently, Fabrikant, Koutsoupias and Papadimitriou [7] introduced a natural and beautifully simple model of network growth involving a trade-off between geometric and network objectives, with relative strength characterized by a single parameter which scales as a power of the number of nodes. In addition to giving experimental results, they proved a power-law lower bound on *part* of the degree sequence, for a wide range of scalings of the parameter. Here we prove that, despite the FKP results, the overall degree distribution is very far from satisfying a power law.

First, we establish that for almost all scalings of the parameter, either all but a vanishingly small fraction of the nodes have degree 1, or there is exponential decay of node degrees. In the former case, a power law can hold for only a vanishingly small fraction of the nodes. Furthermore, we show that in this case there is a large number of nodes with almost maximum degree. So a power law fails to hold even approximately at either end of the degree range. Thus the power laws found in [7] are very different from those given by other internet models or found experimentally [8].

1 Introduction

In the last few years there has been an explosion of interest in ‘scale-free’ random networks, based on measurements indicating that many large real-world networks have certain scale-free properties, for example power-law distributions of degrees and other parameters. The original observations of Faloutsos, Faloutsos and Faloutsos [8], and later many others, have led to a host of proposals for random graph models to explain these power laws, and to better understand the

[‡] Research undertaken during an internship at Microsoft Research.

[§] Research supported by NSF grant DSM 9971788 and DARPA grant F33615-01-C-1900.

[¶] Research undertaken while visiting Microsoft Research.

mechanisms at work in the growth of real-world networks such as the internet or web graphs; see [2,3,9] for a few examples. For extensive surveys of the huge amount of work in this area, see Albert and Barabási [1] and Dorogovtsev and Mendes [6]; for a survey of the rather smaller quantity of mathematical work see [4].

Most of the models introduced use a small number of basic mechanisms, mainly preferential attachment or copying, to produce power laws, and do not involve any reference to underlying geometry. Thus, while they may be appropriate for the web graph, for example, they do not seem to be suitable for the internet graph itself.

In [7], Fabrikant, Koutsoupias and Papadimitriou (FKP) proposed a new paradigm for power law behaviour, which they called ‘heuristically optimized trade-offs’: power laws may result from ‘complicated optimization problems with multiple and conflicting objectives.’ Their paradigm generalizes previous work of Carlson and Doyle [5] on ‘highly optimized tolerance,’ in which reliable design is one of the objectives.

In order to illustrate this paradigm, Fabrikant, Koutsoupias and Papadimitriou introduced a simple, natural network model with such a mechanism. As in many models, a network is grown one node at a time, and each node chooses a previous node to which it connects. However, in contrast to other network 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 which is ‘central’ in the network as a graph. Centrality may be measured by using, for example, the graph distance to the initial node.

Several variants of the basic model are considered by Fabrikant, Koutsoupias and Papadimitriou in [7]. The precise version we shall consider here is the principal version studied in [7]: fix a region $\mathcal{D}$ of area one in the plane, for example a disk or a unit square. The model is then determined by the number of nodes, $n + 1$, and a parameter, α . We start with a point x_0 of $\mathcal{D}$ chosen uniformly at random, and set $W(x_0) = 0$. For $i = 1, 2, \dots, n$ we choose a new point x_i of $\mathcal{D}$ uniformly at random, and connect x_i to an earlier point x_j chosen to minimize

$$W(x_j) + \alpha d(x_i, x_j)$$

over $0 \leq j < i$. Here $d(.,.)$ is the usual Euclidean distance. Having chosen x_j , we set $W(x_i) = W(x_j) + 1$. At the end we have a random tree $T = T(n, \alpha)$ on $n + 1$ nodes $x_0, \dots, x_n$, where each node has a weight $W(x_i)$ which is just its graph distance in the tree from x_0 .

As in [7], we consider $n \rightarrow \infty$ with α some function of n , typically a power.

One might think from the title or a first reading of [7] that the form of the degree sequence of this model has been essentially established. In fact, as we shall describe in the next section, this is not the case. Indeed, two of our results, while of course consistent with the actual results of [7], go against the impression given there that the entire degree sequence follows a power law.

2 Results

As in [7] we consider α in two ranges. Roughly speaking, *large* α will mean $\alpha > n^{1/2}$, and *small* α will mean $\alpha < n^{1/2}$. In fact, to keep things simple we will allow ourselves a logarithmic gap.

Most of the time we will work in terms of the tail of the distribution. Let $\alpha = \alpha(n)$ be given. For each $k = 1, 2, \dots$, let $q_k(\alpha, n)$ be the expected number of nodes of $T(n, \alpha)$ with degree at least k , and let $\rho_k(\alpha) = \lim_{n \rightarrow \infty} q_k(\alpha, n)/n$ be the limiting proportion of nodes with degree at least k .

2.1 Small α

The impression given on first reading [7] is that for small α the whole degree distribution follows a power law. However, the experimental results of [7] strongly suggest that there is a new kind of power law, holding over a large range of degrees, from 2 up to a little below the maximum degree, but involving only a very small proportion of the vertices.

On a second look the situation is more confusing. Quoting the relevant part of the theorem (changing D to k for consistency with our notation):

If $\alpha \geq 4$ and $\alpha = o(\sqrt{n})$, then the degree distribution of T is a power law; specifically, the expected number of nodes with degree at least k is greater than $c \cdot (k/n)^{-\beta}$ for some constants c and β (that may depend on α): $E[|\{i : \text{degree of } i \geq k\}|] > c(k/n)^{-\beta}$. Specifically, for $\alpha = o(\sqrt[3]{n^{1-\epsilon}})$ the constants are: $\beta \geq 1/6$ and $c = O(\alpha^{-1/2})$.

The usual form of a power law would be that a proportion $k^{-\beta}$ of vertices have degree at least k , which is not what is claimed above. There are other problems: the constant c depends on α which depends on n , so c is not a constant. Allowing c to be variable, the claim may then become meaningless if c is very small.

Turning to the proof in [7], a nice geometric argument is given to show that, for $\alpha = o(n^{(1-\epsilon)/3})$ and $k \leq n^{1-\epsilon}/(C\alpha^3)$, which is far below the maximum degree, the expected number $q_k(\alpha, n)$ of vertices with degree at least k is at least $cn^{1/6}\alpha^{-1/2}k^{-1/6}$, where c and C are absolute constants. This supports the experimental results, showing that this interesting new model does indeed give power laws over a wide range; however, it tells us nothing about the vast majority of the vertices, namely all but $O(n^{1/6})$.

Now, in many examples of real-world networks, and in the preferential attachment and copying models of [2,9] and others, the power-law degree distribution involves almost all vertices, and, less clearly, holds very nearly up to the maximum degree. (In the latter case, the power law is often called a ‘Zipf law’, though in fact Zipf’s law is a power law with a particular power.) Thus it is interesting to see whether this is the case for the FKP model.

Theorem 1. *Let $\alpha = o(n^{1/2}/(\log n)^2)$. Then, whp the tree $T(n, \alpha)$ has at least $n - O(\alpha^{1/2}n^{3/4} \log n) = n - o(n)$ leaves.*

In other words, almost all vertices of $T(n, \alpha)$ have degree 1; in particular, when $\alpha = n^a$ for some constant $a < 1/2$, the number of vertices with degree more than 1 is at most n^b for some constant $b < 1$. This contrasts strongly with the usual sense of power-law scaling, namely that the proportion of vertices with degree k converges to a function $f(k)$ which in turn decays like a power of k . This notion is implicit in [8] and [1], for example.

Our second result concerns the high degree vertices, showing that a ‘Zipf-like’ law does not hold. As usual, we write $O^*(\cdot)$ for $O((\log n)^C)$, suppressing constant powers of $\log n$, and similarly for $\Theta^*(\cdot)$. We write **whp** to mean *with high probability*, i.e., with probability $1 - o(1)$ as $n \rightarrow \infty$.

Theorem 2. *Suppose that $(\log n)^7 \leq \alpha \leq n^{1/2}/(\log n)^4$. Then there are constants $c, C > 0$ such that **whp** the maximum degree of $T(n, \alpha)$ is at most Cn/α^2 , while $T(n, \alpha)$ has $\Theta^*(\alpha^2)$ nodes of degree at least cn/α^2 .*

Taking $\alpha = n^a$ for a constant, $0 < a < 1/2$, for example, this says that there are many (a power of n) vertices with degree close to (within a constant factor of) the maximum degree. This contrasts sharply with a so-called Zipf distribution, where there would be a constant number of such vertices. In fact, our method will even show that there are many vertices with degree $(1 - o(1))$ times the maximum.

2.2 Large α

We now turn to the simpler case of large α . This case is interesting for three reasons: one is simply completeness. The second is that the case $\alpha = \infty$, while involving no trade-offs, is a very nice geometric model in its own right. Finally, the large α results will turn out to be useful in studying the small α case.

Theorem 3. *Suppose that $\alpha = \alpha(n)$ satisfies $\alpha/(\sqrt{n} \log n) \rightarrow \infty$. Then there are positive constants A, A', C, C' such that*

$$A'e^{-C'k} \leq \rho_k(\alpha) \leq Ae^{-Ck}$$

holds for every $k \geq 1$.

In other words, for large α the tail of the degree distribution decays exponentially, as for classical random graphs with constant average degree.

Our theorem strengthens the upper bound in [7], which says that $q_k(\alpha, n) \leq O(n^2)e^{-Ck}$, or, loosely speaking, that $\rho_k(\alpha) \leq O(n)e^{-Ck}$. Note that the upper bound of [7] gives information only for k larger than a constant times $\log n$, i.e., a vanishing fraction of the nodes. Furthermore, we complement our stronger upper bound with a matching lower bound.

We remark again that our results contain logarithmic factors that are presumably unnecessary; these help keep the proofs relatively simple.

3 The Pure Geometric Model

In this section we consider the case $\alpha = \infty$. In this case, each node x_i simply connects to the closest node among $x_0, \dots, x_{i-1}$. Although this model is not our main focus, it is of interest in its own right, and it is somewhat surprising that it does not seem to have been extensively studied, unlike related objects such as the minimal spanning tree, for example (see [11,12]). We study this case for two reasons. First, for large α , $T(n, \alpha)$ approximates $T(n, \infty)$. Second, certain results about $T(n, \infty)$ will be useful to study $T(n, \alpha)$ even for very small α . We start with a simple but surprising exact result.

Lemma 1. *In the random tree $T(n, \infty)$, for $1 \leq t \leq n$ the probability that x_t is at graph distance r from x_0 , i.e., has weight r , is exactly*

$$\sum_{1 \leq i_1 < i_2 < \dots < i_{r-1} < t} \frac{1}{i_1 i_2 \dots i_{r-1} t}$$

Proof. We write $i \rightarrow j$ if $j < i$ and x_i is adjacent (joined directly) to x_j . The key observation is as follows: suppose we fix the points $x_s, x_{s+1}, \dots, x_n$, and also the set of points $S_{s-1} = \{x_0, x_1, \dots, x_{s-1}\}$, leaving undetermined the order of the points in S_{s-1} . Then x_s is joined to the closest point in S_{s-1} , which is a certain point x . When we choose the ordering of the points in S_{s-1} , the point x is equally likely to be x_0, x_1 , or any other $x_j, j < s$. Taking $s = t$, it follows that the probability that $t \rightarrow j$ is exactly $1/t$. Using the same observation for $s = j$ we see that, given $t \rightarrow j$, the probability that $j \rightarrow k$ is $1/j$. Continuing, the probability that $t \rightarrow i_{r-1} \rightarrow i_{r-2} \rightarrow \dots \rightarrow i_1 \rightarrow 0$ is $1/(t i_{r-1} i_{r-2} \dots i_1)$. As these events are disjoint for different sequences, the lemma follows.

Another way of stating the lemma is that, for any fixed t , the distribution of the graph distance from t to 0 is the same as in a uniform random recursive tree. These are trees grown one node at a time, in which each new node is joined to an earlier node chosen uniformly at random. Such objects have been studied for some time; see, for example, the survey [10]. The radius (here, maximum node weight) of such a tree was shown by Pittel [13] to be $(c + o(1)) \log n$ for a certain constant $c = 1.79..$ given by a root of an equation. This result does not apply to $T(n, \alpha)$ because the dependence between nodes is different. We shall just give an upper bound.

Lemma 2. *Let $\alpha = \alpha(n)$ be arbitrary. Then as $n \rightarrow \infty$, whp every point in $T(n, \alpha)$ has weight at most $3 \log n$.*

Proof. For $\alpha = \infty$ this follows from Lemma 1 by straightforward calculation: the expected number of points with weight r is

$$\sum_{1 \leq i_1 < i_2 < \dots < i_{r-1} < t \leq n} \frac{1}{i_1 i_2 \dots i_{r-1} t} \leq \frac{1}{r!} \left(\sum_{i=1}^n \frac{1}{i} \right)^r \leq \frac{(1 + \log n)^r}{r!} \leq (e(1 + \log n)/r)^r.$$

Set $r = \lfloor 3 \log n \rfloor$. Then the expectation above tends to zero, so **whp** there are no points with weight r , and the radius, or maximum weight, is at most $r - 1$.

We can compare finite α with $\alpha = \infty$. Consider the sequence of points as fixed, let $W(x_i)$ be the weights for some finite $\alpha = \alpha(n)$, and let $W_\infty(x_i)$ be the weights obtained with $\alpha = \infty$. For any α , the weight of a point x_i is always at most one more than the weight of the nearest earlier point x_j : if we connect to a more distant point x_k it must have smaller weight than x_j . Since we have equality for $\alpha = \infty$, it follows that for any α we have $W(x_i) \leq W_\infty(x_i)$. As shown at the start of the proof, **whp** we have $W_\infty(x_i) \leq 3 \log n$ for every i , so we are done.

The lemma has a simple heuristic explanation: for $\alpha = \infty$ the closest earlier x_j to x_i will typically have index j around $i/2$, so it will take order $\log n$ steps to reach the origin. For finite α , any bias is towards earlier points. One might expect monotonicity of the weights as α decreases from one finite value to another, but this does not hold in general.

3.1 Degrees for $\alpha = \infty$

Here we are interested in the quantities $\rho_k(\infty)$ defined in section 2; our aim is to prove the $\alpha = \infty$ case of Theorem 3.

This result easy to see intuitively. As noted above, for $i < t \leq n$ the probability that $t \rightarrow i$ is exactly $1/t$. Thus the expected degree of node i in $T(n, \infty)$ is exactly

$$\frac{1}{i+1} + \frac{1}{i+2} + \cdots + \frac{1}{n} = \log(n/i) + O(i^{-1}).$$

If every degree were close to its expectation, this would give the result. In fact, it turns out that the probability of the degree of node i exceeding its expectation by some amount x decreases exponentially with x . To see this heuristically we use the notion of Voronoi cells: given a region $\mathcal{D}$ and a set of points X in $\mathcal{D}$, the region $\mathcal{D}$ is tiled by Voronoi cells V_x , one for each $x \in X$, defined as the set of points of $\mathcal{D}$ closer to x than to any other $y \in X$.

Here we consider $V_{i,t}$, the Voronoi cell of x_i with respect to $\{x_0, x_1, \dots, x_t\}$. Note that $t \rightarrow i$ if and only if x_t is in $V_{i,t-1}$. Keeping i fixed, as t increases $V_{i,t}$ shrinks whenever x_t lands close enough to x_i . In particular, $V_{i,t}$ gets smaller whenever x_t lands in $V_{i,t-1}$ itself; the key point is that in this case the area of $V_{i,t}$ is on average less than that of $V_{i,t-1}$ by a factor f strictly less than 1. On average, $V_{i,i}$ has area $1/(i+1)$, and $V_{i,n}$ area $1/(n+1)$. Hence it is very unlikely that i has degree much bigger than $\log(n/i)$; otherwise the area of $V_{i,t}$ would decrease by too much as t increases from i to n .

Proof (of Theorem 3 for $\alpha = \infty$). We make the argument outlined above rigorous. The key observation is as follows: let V be a convex region and C a fixed point of V . Let X be a point of V chosen uniformly at random, and let V' be the set of points of V closer to C than to X . Then the expected area of V' is at most $15/16$ times the area of V . To see this, taking C as the origin divide

V into four parts Q_1, Q_2, Q_3, Q_4 , the intersections of V with the four quadrants of $\mathbf{R}^2$. Suppose X falls in a certain Q_i . If Y is any other point of Q_i then $(X + Y)/2$ is closer to X than to C . This is easy to see geometrically: the vector $(X + Y)/2 - X = (Y - X)/2$ is shorter than $(Y + X)/2$, as the angle between X and Y is less than 90 degrees. Hence $V \setminus V'$ contains a copy of Q_i shrunk by a factor two in each direction, so in this case $\text{area}(V \setminus V') \geq \text{area}(Q_i)/4$. Averaging, noting that the probability that X lies in Q_i is proportional to $\text{area}(Q_i)$,

$$\mathbf{E}(\text{area}(V \setminus V')) \geq \sum_{i=1}^4 \frac{\text{area}(Q_i)^2}{4 \text{area}(V)} \geq \frac{\text{area}(V)}{16},$$

where the last step follows by convexity. Thus $\mathbf{E}(\text{area}(V')) \leq \frac{15}{16} \text{area}(V)$. Hence, fixing $x_0, \dots, x_{t-1}$, conditional on $t \rightarrow i$, i.e., on $x_t \in V_{i,t-1}$, the expected area of $V_{i,t}$ is at most $\frac{15}{16}$ times the area of $V_{i,t-1}$.

Fix $0 \leq i \leq n$. Continuing the construction of $T(n, \infty)$ indefinitely, let $t_1 < t_2 < t_3 < \dots$ be the points that send edges to i . Let $W_0 = V_{i,i}$ and $W_j = V_{i,t_j}$ be the Voronoi cells of i looked at at time i , and at each time when a new node joins to i . Note that $\mathbf{E}(\text{area}(W_0)) = 1/(i + 1)$ as this is the cell corresponding to one of $i + 1$ points chosen independently. It may be that the Voronoi cell containing i shrinks at intermediate times as well, but certainly given W_j , we have $\mathbf{E}(\text{area}(W_{j+1})) \leq \frac{15}{16} \text{area}(W_j)$. Hence

$$\mathbf{E}(\text{area}(W_k)) \leq \frac{1}{i + 1} (15/16)^k. \tag{1}$$

We now consider time n : fix x_i and consider the n remaining points of $x_0, \dots, x_n$ as random. Ignoring effects from the boundary of the region, if no other point lies within distance d of x_i , then the Voronoi cell $V_{i,n}$ contains a circle of radius $d/2$. In other words, for $\text{area}(V_{i,n})$ to be smaller than $\pi(d/2)^2$, one of the n points must lie in a disk of radius d , with area πd^2 , an event with probability at most $n\pi d^2$. It turns out that boundary effects go the right way, so

$$\Pr(\text{area}(V_{i,n}) \leq x) \leq 4nx. \tag{2}$$

Finally, if i has degree at least $k + 1$ in $T(n, \infty)$ then at least k of the first n points join to i , so $t_k \leq n$, and $\text{area}(V_{i,n}) \leq \text{area}(W_k)$. For any x , the probability of this is at most

$$\Pr(\text{area}(W_k) \geq x) + \Pr(\text{area}(V_{i,n}) \leq x),$$

which is at most

$$\frac{1}{x} \frac{1}{i + 1} (15/16)^k + 4nx,$$

from (1), Markov's inequality and (2). The optimum choice

$$x = (15/16)^{k/2} / \sqrt{4n(i + 1)}$$

yields

$$\Pr(\deg(i) \geq k+1) \leq 4\sqrt{\frac{n}{i+1}}(15/16)^{k/2}. \quad (3)$$

Summing over i by comparison with an integral, the expected number of nodes with degree at least $k+1$ is at most $(8+o(1))n(15/16)^{k/2}$, so $\rho_{k+1} \leq 8(15/16)^{k/2}$, proving the upper bound.

The lower bound also follows easily; the bound (3) shows that an individual degree is very unlikely to be much larger than its expectation. It follows that $\deg(i)$ has a significant (at least 1%, say) chance of being at least half its expectation, and the lower bound follows.

4 Observation

In the remaining proofs we will use again and again the following simple observation. At time t the points currently placed approximate a Poisson process with density $1/t$, so the closest earlier point x_j to x_t is ‘typically’ at distance $\Theta(1/\sqrt{t})$. In particular, for a fixed $t > 0$, if $\omega \rightarrow \infty$ then **whp** $\omega^{-1}t^{-1/2} \leq d(x_t, x_j) \leq \omega t^{-1/2}$.

Furthermore, for any positive constant c , **whp** at time t every disk of radius $c \log nt^{-1/2}$ contains a point already placed. (This is easy to check, and also follows from a more general and more precise result of Penrose [11].)

5 Large α

Proof (of Theorem 3). The case $\alpha = \infty$ was proved in section 3; to extend this result to α large requires only a little further work.

Suppose that $\alpha/(\sqrt{n} \log n) \rightarrow \infty$. Fix $\delta > 0$, and consider a point x_i with $i \geq \delta n$, and the nearest earlier point x_j . Since all weights are within $3 \log n$ of one another, for x_i to join to some other point x_k we must have

$$d(x_i, x_k) \leq d(x_i, x_j) + 3 \log n / \alpha = d(x_i, x_j) + o(n^{-1/2}). \quad (4)$$

As noted above, **whp** we have $d(x_i, x_j) \leq \omega i^{-1/2}$. Considering x_i and x_j as fixed, given that x_j is the closest earlier point to x_i , the other x_k , $k < i$, are distributed uniformly outside the circle centered at x_i with radius $d(x_i, x_j)$, and for a particular x_k to satisfy (4) it must lie in an annulus around this circle with thickness $o(n^{-1/2})$. This annulus has area $o(d(x_i, x_j)n^{-1/2}) = o((in)^{-1/2})$ (taking $\omega \rightarrow \infty$ slowly enough). Since there are $i-1$ points to consider, the probability that x_i does not join to the closest point x_j is at most $o(\sqrt{i/n}) = o(1)$. Thus, **whp** almost all points join to the nearest earlier point. In particular, the final tree $T(n, \alpha)$ differs in only $o(n)$ edges from $T(n, \infty)$, and hence the numbers ρ_k are the same as for $\alpha = \infty$.

The conclusion that $\rho_k(\alpha) = \rho_k(\infty)$ should hold provided only that $\alpha/\sqrt{n} \rightarrow \infty$; this is likely to be harder to show.

6 Critical α

If $\alpha = \Theta(\sqrt{n})$ then we expect the behaviour of the tree to be similar to that for $\alpha = \infty$. In particular, for $\alpha = cn^{1/2}$, $c > 0$, we expect limiting proportions $\rho_k = \rho_k(c)$ with $\rho_k(c) \rightarrow \rho_k(\infty)$ as $c \rightarrow \infty$ but $\rho_k(c)$ not in general equal to $\rho_k(\infty)$. Also, the radius, or maximum weight, should be $A(c) \log n$. We have not stated a result for this case, which is likely to be harder to analyze precisely.

Note that one might hope for a complete power law in the critical case, but this does not happen, as shown by, for example, the weak exponential upper bound in [7].

7 Small α

This case is the heart of our paper. Here *small* would ideally mean $o(n^{1/2})$; in fact, for simplicity we shall work with extra logarithmic factors. Throughout this section it will be convenient to re-scale by a factor of α : rather than choosing points in the unit square or disk, we choose points in a square $\mathcal{D}$ of side α ; correspondingly, we join x_i to the earlier point x_j minimizing $W(x_j) + d(x_i, x_j)$. Note that the final density n/α^2 of points is high (compared to 1). The reason to consider this scaling is that differences in re-scaled distances of order 1 are what is relevant; in particular, as all weights are within $3 \log n$ of each other, no point ever connects to a point more than $3 \log n$ further away than its nearest point.

Considering the process defining $T(n, \alpha)$ as points arrive one by one, there is a transition in the behaviour around time $t = \alpha^2$. This is because in the re-scaled process, the density of points at time t is t/α^2 . At times much earlier than α^2 , this density is very small, so distances and their differences are typically large, and the process looks very much like the $\alpha = \infty$ case of connecting to the nearest point.

On the other hand, at times much later than α^2 , the density of points is already very high. We expect that certain ‘attractive’ early points will have established ‘regions of attraction’ of order unit size; almost all later points then just join to the nearest attractive point by a short edge. In particular, almost all later points will themselves never be joined to.

7.1 Small Degrees

We now prove Theorem 1 from section 2, a precise version of the final observation from the paragraph above, that almost all points are leaves in $T(n, \alpha)$, i.e., have degree 1. In the proof we shall use the following simple geometric lemma.

Lemma 3. *Let $\mathcal{D}$ be a convex set in the plane, and let $X = \{x_0, \dots, x_{k-1}\}$ be a set of points in $\mathcal{D}$. For $r > 0$ let $X(r)$ be the set of points in $\mathcal{D}$ at distance at most r from some x_i . For $0 < r_1 < r_2$ we have*

$$\text{area}(X(r_2)) \leq \frac{r_2^2}{r_1^2} \text{area}(X(r_1)).$$

Proof. A point $x \in \mathcal{D}$ lies in $X(r)$ if and only if $d(x, x_i) \leq r$ for x_i the closest point of X to x . Let us partition $\mathcal{D}$ into the Voronoi cells $V_i = \{x \in \mathcal{D} : d(x, x_i) = \min_j d(x, x_j)\}$. (We may ignore the boundaries.) Then, for any r , we have $\text{area}(X(r)) = \sum_i \text{area}(X(r) \cap V_i)$. But V_i is convex; thus if $X(r_2) \cap V_i$ is a certain region A , then $X(r_1) \cap V_i$ certainly contains the region obtained by shrinking A by a factor r_2/r_1 around the point x_i . Hence, $\text{area}(X(r_1) \cap V_i) \geq r_1^2/r_2^2 \text{area}(X(r_2) \cap V_i)$, and the lemma follows.

Of course, a corresponding result holds in any dimension, with exactly the same proof. Also, the result holds for an arbitrary (infinite) set X .

Proof (of Theorem 1). If x_i is joined to the earlier point x_j , we call $x_i x_j$ the edge from x_i . We consider edges with lengths in three ranges: writing γ for $\alpha^{1/2} n^{-1/4} = o(1/\log n)$, we call an edge of length ℓ *short* if $\ell < 1$, *long* if $\ell > 1 + \gamma$, and *medium* if $1 \leq \ell \leq 1 + \gamma$.

The key observation is that if the edge $x_i x_j$ from x_i is short, then x_i has degree 1 in the final graph $T(n, \alpha)$. To see this, note that no later point x_k can possibly join to x_i , since $W(x_i) = W(x_j) + 1$, while $d(x_k, x_j) < d(x_k, x_i) + 1$, so x_k would join to x_j in preference to x_i . To complete the proof we shall show that the number of medium and long edges is small.

Suppose that the edge $x_i x_j$ from x_i is medium. Writing w for $W(x_j)$, at time $i - 1$ there is no point with weight w within distance 1 of x_i , but there is such a point within distance $1 + \gamma$. Turning this around, let $X = \{x_j : W(x_j) = w, 0 \leq j \leq i - 1\}$. Then x_i lies in $X(1 + \gamma)$, but not in the interior of $X(1)$. By Lemma 3, $\text{area}(X(1 + \gamma)) \leq (1 + \gamma)^2 \text{area}(X(1))$. Hence, given $x_0, \dots, x_{i-1}$, the probability that x_i lies in $X(1 + \gamma) \setminus X(1)$ is at most $\frac{(1+\gamma)^2 - 1}{(1+\gamma)^2} \leq 2\gamma$. It follows from Lemma 2 that there are at most $\log n$ values of w to consider, so the probability that for a given i the edge $x_i x_j$ is medium is at most $2\gamma \log n = o(1)$. It follows that **whp** there are at most $2\gamma n \log n = 2\alpha^{1/2} n^{3/4} \log n = o(n)$ medium edges in the final tree.

We now consider long edges, i.e., edges of length at least $1 + \gamma$. The key observation is that when the edge from x_i is long, this edge provides a useful shortcut in future: new points near x_i have a better connection route than if x_i were deleted. To formalize this, given the final set of points $x_0, \dots, x_n$ and their weights, for $1 \leq i \leq n$ let us define a function $c_i : \mathcal{D} \rightarrow \mathbf{R}$ by $c_i(x) = \min_{j < i} \{W(x_j) + d(x, x_j)\}$. Note that c_i only depends on the locations of $x_0, \dots, x_{i-1}$, and that $c_i(x)$ is the ‘cost’ of connecting a potential new point at x to the existing tree on $x_0, \dots, x_{i-1}$. In particular, x_i joins to the x_j attaining the minimum defining $c_i(x_i)$, and receives weight $W(x_j) + 1$. Suppose that $x_i x_j$ is long, i.e., has length at least $1 + \gamma$, and let $w = W(x_j)$. Then we have $c_i(x_i) = w + d(x_i, x_j) \geq w + 1 + \gamma$, but $c_{i+1}(x_i) = w + 1$. Hence

$$c_{i+1}(x_i) \leq c_i(x_i) - \gamma.$$

Our strategy is to consider the quantities $I_i = \int_{\mathcal{D}} c_i(x)$, $1 \leq i \leq n$. We shall show that I_i is positive, and decreases with i . Also, we shall show that **whp** I_{i_0} is not too large for some $i_0 = o(n)$, and that if the edge from i is long, then

$I_i - I_{i+1}$ is not too small; together these observations will give a bound on the number of long edges.

It is immediate from the definition that $c_i(x)$ and hence I_i are positive. Also, it is immediate that $c_{i+1}(x) \leq c_i(x)$ —the minimum is taken over a larger set. Hence $I_{i+1} \leq I_i$ for each i .

Set $i_0 = \lfloor (\alpha \log n)^2 \rfloor = o(n)$. At time i_0 the overall density of points is at least $(\log n)^2$. Hence, **whp**, for every $x \in \mathcal{D}$ there is a $j < i_0$ with $d(x, x_j) < 1$. Since $W(x_j) \leq 3 \log n$ from section 5, we have $c_{i_0}(x) \leq 1 + 3 \log n$. Thus, **whp**,

$$I_{i_0} \leq (1 + 3 \log n) \text{area}(\mathcal{D}) = O(\alpha^2 \log n).$$

Finally, suppose that the edge from x_i is long. As shown above, we then have $c_{i+1}(x_i) \leq c_i(x_i) - \gamma$. Now each $c_k(x)$ is the minimum of a set of Lipschitz functions with constant 1, and is hence Lipschitz with constant 1. Thus for y at distance $\ell \leq \gamma/2$ from x_i we have $c_{i+1}(y) \leq c_i(y) - \gamma + 2\ell$. Integrating, we see that

$$I_{i+1} \leq I_i - \frac{1}{4} \int_{\ell=0}^{\gamma/2} (\gamma - 2\ell) 2\pi \ell d\ell = I_i - \frac{\pi}{48} \gamma^3.$$

(The initial factor of $1/4$ allows for the fact that the little disk we are integrating over may not lie entirely within $\mathcal{D}$.)

Since I_i is decreasing and positive, from the two equations above we see that **whp** the number of x_i , $i \geq i_0$, from which we have long edges is at most $O(\alpha^2 \log n / \gamma^3)$. Thus, **whp** we have $i_0 + O(\alpha^2 \log n / \gamma^3) = O(\alpha^{1/2} n^{3/4} \log n)$ long edges.

Combining the cases above completes the proof: we have shown that in total there are $O(\alpha^{1/2} n^{3/4} \log n) = o(n)$ medium and long edges, and hence $n - o(n)$ short edges. But every short edge gives rise to a leaf in T , so almost all nodes are leaves.

The above result shows that for small α the degree sequence of $T(n, \alpha)$ is not a power law in the usual sense, which is that for fixed k there is a limiting proportion p_k of nodes with degree k , which falls off as some power of k . In particular, here $p_1 = 1$, while $p_k = 0$ for all $k \neq 1$.

7.2 Large Degrees

We now turn to the opposite end of the degree sequence, showing that there is a bunching of degrees near the maximum, in the sense that for $\alpha = n^a$, $0 < a < 1/2$, a positive power of n nodes have degree within a constant factor of the maximum. This is easy to see heuristically: up to time α^2 the process looks like the $\alpha = \infty$ case, and all degrees are at most $O(\log n)$. Beyond this time, $\Theta(\alpha^2)$ attractive points will have become established, each of which will attract the $\Theta(n/\alpha^2)$ later points that fall in its zone of attraction, which will have re-scaled area $O(1)$, out of a total re-scaled area of α^2 . Since no point can maintain a region of attraction much bigger than this for long, the maximum degree will also be of order $\Theta(n/\alpha^2)$.

As before, for simplicity we have allowed ourselves extra logarithmic factors when making this precise. In Theorem 2, which we now prove, the main case of interest is $\alpha = n^a$ for some constant a between 0 and $1/2$.

Proof (of Theorem 2). We start with the maximum degree, aiming to show that this is $O(n/\alpha^2)$. Let $t_0 = \alpha^2/(\log n)^2$. Arguing as in section 5 we see that **whp** at time t_0 the tree is essentially $T(t_0, \infty)$, and that all degrees are $O(\log n)$.

Fix a point x_i . To obtain the desired bound on the final degree of i we need only consider which x_j , $j > t_0$, join to x_i . Now at time t_0 the typical distance between points is $\log n$, and allowing for deviations no disk of radius $(\log n)^2$ is empty. (This is a rescaling of the final observation from section 4.) It follows that all later edges have length at most $2(\log n)^2$. Hence we need only consider a region R around x_i with radius $O((\log n)^2)$. We divide this into a ‘good region’, a disk of radius 1.1 around x_i , and a ‘bad region’, the rest of R . Note that $O(n/\alpha^2)$ points will fall into the good region, so we need only control the bad region. This is easy: the bad region can be covered by $O((\log n)^4)$ disks of radius 0.01. Within any such disk at most one point x_j , $j > i$, can join to i ; a second point $x_{j'}$ landing in the same disk would rather join to x_j at distance < 0.01 than to x_i at distance at least 1.1, since the weight of x_j is only one larger than that of x_i . Hence the expected degree of x_i is at most

$$O(\log n) + O(n/\alpha^2) + O((\log n)^4) = O(n/\alpha^2).$$

Since the main term is at least $\Theta((\log n)^2)$ it is easy to check that large deviations are very unlikely, and hence that the maximum degree is $O(n/\alpha^2)$, as claimed.

Establishing the existence of ‘attractive’ points which remain attractive is not quite so easy, as the situation is not really as simple as the heuristic description suggests. However, with the flexibility allowed by logarithmic factors we can proceed as follows. Let us consider time $t_1 = \alpha^2/\omega$, where $\omega = (\log n)^7$. Set $S = \{x_0, \dots, x_{t_1}\}$, noting that typical distances between nearest points of S are of order $\omega^{1/2}$. In fact, as S approximates a Poisson process with density ω^{-1} , one can check that **whp** every disk of radius $0.9\sqrt{\omega \log n}$ contains a point of S . (To see this, observe that S has very small probability of missing a given disk of radius $0.85\sqrt{\omega \log n}$.) For the moment we shall condition on $x_0, \dots, x_{t_1}$, assuming that this property holds, and noting the consequence that all edges added after time t_1 have length at most $0.9\sqrt{\omega \log n} + 3 \log n \leq (\log n)^4 - 1$; the nearest old point to any new point is within $0.9\sqrt{\omega \log n}$, and can have weight at most $3 \log n$ more than the point actually joined to.

Let us say that a point of S is *isolated* if it is at distance at least 2 from every other point of S . Let us say that a point $x_i \in S$ of weight w is *good* if no other point $x_j \in S$ with smaller weight lies within distance $3(\log n)^5$ of x_i . Isolated good points are useful for the following reason: we claim that every later point x_k , $k > t_1$, within distance 1 of an isolated good point x_i will join to x_i . To see this, note that we have $x_k = x_{a_0} \rightarrow x_{a_1} \rightarrow x_{a_2} \rightarrow \dots \rightarrow x_{a_{\ell-1}} \rightarrow x_{a_\ell}$ for some sequence $k = a_0 > a_1 > \dots > a_{\ell-1} > a_\ell$, with $a_{\ell-1} > t_1$, $a_\ell \leq t_1$. Suppose that $x_k \rightarrow x_i$ does not hold, i.e., $a_1 \neq i$. Then, as x_i is within distance one of x_k , we have $W(x_{a_1}) \leq W(x_i)$, and if equality holds, then x_{a_1} must also be within

distance one of x_k . In the case of equality, since x_i is isolated it follows that $a_1 > t_1$, i.e., $\ell > 1$. Since $x \rightarrow y$ implies $W(x) = W(y) + 1$, it follows in either case that $W(x_{a_\ell}) < W(x_i)$. But then x_k is connected by a sequence of at most $3 \log n$ edges of length at most $(\log n)^4 - 1$ to a point $x_{a_\ell} \in S$ with smaller weight than x_i , contradicting that x_i is good, and establishing the claim.

Thus an isolated good point attracts all points after t_1 within distance 1, and will have final degree at least cn/α^2 **whp**. In fact, using only the Chernoff bounds, the deviation probability for one point is $o(n^{-1})$, so **whp** every isolated good point has final degree at least cn/α^2 .

It remains to show that at time $t_1 = \alpha^2/(\log n)^7$ there are many isolated good points. We do this using a little trick. (We treat $3 \log n$ as an integer for notational convenience.)

Let $r_w = 3(\log n)^5(1 + 3 \log n - w)$, so $r_0 = O((\log n)^6)$, $r_{3 \log n} = 3(\log n)^5 \geq (\log n)^4$, and $r_w = r_{w-1} - 3(\log n)^5$. For $0 \leq w \leq 3 \log n$ let S_w be the set of points $x_i \in S$ with weight at most w , and let $T_w = S_w(r_w)$ be the set of all points in $\mathcal{D}$ within distance r_w of some point in S_w . Note that T_0 has area $O((\log n)^{12})$, which is much less than α^2 . On the other hand, $T_{3 \log n}$ is, **whp**, all of $\mathcal{D}$, since, as noted earlier, **whp** every point of $\mathcal{D}$ is within distance $(\log n)^4$ of some $x_i \in S$, which has weight at most $3 \log n$ by Lemma 2. Thus

$$\sum_w \text{area}(T_w \setminus T_{w-1}) \geq (1 - o(1))\alpha^2.$$

Suppose that $y \in T_w \setminus T_{w-1}$. Then there is some $x_i \in S$ with $W(x_i) \leq w$ and $d(y, x_i) \leq r_w$. On the other hand, there is no $x_j \in S$ with $W(x_j) \leq w - 1$ within distance $r_{w-1} = r_w + 3(\log n)^5$ of y . It follows that $W(x_i) = w$, and that x_i is good, so y is within distance r_w of a good x_i . As each such good x_i can only account for an area $\pi r_w^2 \leq \pi r_0^2 = O((\log n)^{12})$ of $T_w \setminus T_{w-1}$, it follows that **whp** the total number of good points in S is at least $g_0 = \Theta(\alpha^2(\log n)^{-12})$. On the other hand, since the density of points at time t_1 is $(\log n)^{-7}$, the probability that a given x_i , $i \leq t_1$, is not isolated is $\Theta((\log n)^{-7})$, and the expected number of non-isolated points in S is $\Theta(\alpha^2(\log n)^{-14})$. This is $o(g_0)$, so using Markov's inequality, **whp** almost all good points are isolated, completing the proof.

In fact, being a little more careful with the constants, we can show that both the maximum degree and the degrees of almost all isolated good points (those not too near the boundary of $\mathcal{D}$) are $(1 + o(1))\pi n/\alpha^2$. Thus there is a strong bunching of degrees near the maximum.

References

1. R. Albert and A.-L. Barabási, Statistical mechanics of complex networks, *Rev. Mod. Phys.* **74** (2002), 47–97.
2. A.-L. Barabási and R. Albert, Emergence of scaling in random networks, *Science* **286** (1999), 509–512.
3. B. Bollobás and O.M. Riordan, The diameter of a scale-free random graph, to appear in *Combinatorica*. (Preprint available from <http://www.dpmms.cam.ac.uk/~omr10/>.)

4. B. Bollobás and O. Riordan, Mathematical results on scale-free random graphs, in Handbook of Graphs and Networks, Stefan Bornholdt and Heinz Georg Schuster (eds.), Wiley-VCH, Weinheim (2002), 1–34.
5. J.M. Carlson and J. Doyle, Highly optimized tolerance: a mechanism for power laws in designed systems. *Phys. Rev. E* **60** (1999), 1412–1427.
6. S.N. Dorogovtsev and J.F.F. Mendes, Evolution of networks, *Adv. Phys.* **51** (2002), 1079.
7. A. Fabrikant, E. Koutsoupias and C.H. Papadimitriou, Heuristically optimized trade-offs: a new paradigm for power laws in the internet ICALP 2002, LNCS 2380, pp. 110–122.
8. M. Faloutsos, P. Faloutsos and C. Faloutsos, On power-law relationships of the internet topology, SIGCOMM 1999, *Comput. Commun. Rev.* **29** (1999), 251.
9. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins and E. Upfal, Stochastic models for the web graph, FOCS 2000.
10. H.M. Mahmoud and R.T. Smythe, A survey of recursive trees, *Th. of Probability and Math. Statistics* **51** (1995), 1–27.
11. M.D. Penrose, A strong law for the largest nearest-neighbour link between random points, *J. London Math. Soc.* (2) **60** (1999), 951–960.
12. M.D. Penrose, A strong law for the longest edge of the minimal spanning tree. *Ann. Probab.* **27** (1999), 246–260.
13. B. Pittel, Note on the heights of random recursive trees and random m -ary search trees, *Random Struct. Alg.* **5** (1994), 337–347.

Similarity Matrices for Pairs of Graphs

Vincent D. Blondel and Paul Van Dooren

Division of Applied Mathematics, Université catholique de Louvain, 4 avenue Georges
Lemaître, B-1348 Louvain-la-Neuve, Belgium, blondel@inma.ucl.ac.be,
<http://www.inma.ucl.ac.be/~blondel/>
vdooren@csam.ucl.ac.be

Abstract. We introduce a concept of similarity between vertices of directed graphs. Let G_A and G_B be two directed graphs with respectively n_A and n_B vertices. We define a $n_A \times n_B$ *similarity matrix* $\mathbf{S}$ whose real entry s_{ij} expresses how similar vertex i (in G_A) is to vertex j (in G_B): we say that s_{ij} is their *similarity score*. In the special case where $G_A = G_B = G$, the score s_{ij} is the similarity score between the vertices i and j of G and the square similarity matrix $\mathbf{S}$ is the *self-similarity matrix* of the graph G . We point out that Kleinberg's "hub and authority" method to identify web-pages relevant to a given query can be viewed as a special case of our definition in the case where one of the graphs has two vertices and a unique directed edge between them. In analogy to Kleinberg, we show that our similarity scores are given by the components of a dominant vector of a non-negative matrix and we propose a simple iterative method to compute them.

Remark: Due to space limitations we have not been able to include proofs of the results presented in this paper. Interested readers are referred to the full version of the paper [1], and to [2] for a description of an application of our similarity concept to the automatic extraction of synonyms in a dictionary. Both references are available from the first authors web-site.

1 Generalizing Hubs and Authorities

Efficient web search engines such as Google are often based on the idea of characterizing the most important vertices in a graph representing the connections or links between pages on the web. One such method, proposed by Kleinberg [10], identifies in a set of pages relevant to a query search those that are good *hubs* or good *authorities*. For example, for the query "automobile makers", the home-pages of Ford, Toyota and other car makers are good authorities, whereas web pages that list these home-pages are good hubs. Good hubs are those that point to good authorities, and good authorities are those that are pointed to by good hubs. From these implicit relations, Kleinberg derives an iterative method that assigns an "authority score" and a "hub score" to every vertex of a given graph. These scores can be obtained as the limit of a converging iterative process which we now describe.

Let G be a graph with edge set E and let h_j and a_j be the hub and authority scores of the vertex j . We let these scores be initialized by some positive values and then update them simultaneously for all vertices according to the following *mutually reinforcing relation* : the hub score of vertex j is set equal to the sum of the authority scores of all vertices pointed to by j and, similarly, the authority score of vertex j is set equal to the sum of the hub scores of all vertices pointing to j :

$$\begin{cases} h_j \leftarrow \sum_{i:(j,i) \in E} a_i \\ a_j \leftarrow \sum_{i:(i,j) \in E} h_i \end{cases}$$

Let B be the adjacency matrix of G and let h and a be the vectors of hub and authority scores. The above updating equations take the simple form

$$\begin{bmatrix} h \\ a \end{bmatrix}_{k+1} = \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix} \begin{bmatrix} h \\ a \end{bmatrix}_k, \quad k = 0, 1, \dots$$

which we denote in compact form by

$$x_{k+1} = M x_k, \quad k = 0, 1, \dots$$

where

$$x_k = \begin{bmatrix} h \\ a \end{bmatrix}_k, \quad M = \begin{bmatrix} 0 & B \\ B^T & 0 \end{bmatrix}.$$

We are only interested in the relative scores and we will therefore consider the *normalized* vector sequence

$$z_0 = x_0, \quad z_{k+1} = \frac{M z_k}{\|M z_k\|_2}, \quad k = 0, 1, \dots$$

Ideally, we would like to take the limit of the sequence z_k as a definition for the hub and authority scores. There are two difficulties with such a definition. Firstly, the sequence does not always converge. In fact, non-negative matrices M with the above block structure always have *two* real eigenvalue of largest magnitude and the resulting sequence z_k almost never converges. Notice however that the matrix M^2 is symmetric and non-negative definite and so, even though the sequence z_k may not converge, the even and odd sub-sequences do converge. Let us define

$$z_{\text{even}} = \lim_{k \rightarrow \infty} z_{2k} \quad \text{and} \quad z_{\text{odd}} = \lim_{k \rightarrow \infty} z_{2k+1}.$$

and let us consider both limits for the moment. The second difficulty is that the limit vectors z_{even} and z_{odd} do in general depend on the initial vector z_0 and there is no apparent natural choice for z_0 . In Theorem 2, we define the set of all limit vectors obtained when starting from a positive initial vector

$$Z = \{z_{\text{even}}(z_0), z_{\text{odd}}(z_0) : z_0 > 0\}.$$

and prove that the vector z_{even} obtained for $z_0 = \mathbf{1}$ is the vector of largest possible 1-norm among all vectors in Z (throughout this paper we denote by $\mathbf{1}$

the vector, or matrix, whose entries are all equal to 1; the appropriate dimension of $\mathbf{1}$ is always clear from the context). Because of this extremal property, we take the two sub-vectors of $z_{even}(\mathbf{1})$ as definitions for the hub and authority scores. In the case of the above matrix M , we have

$$M^2 = \begin{bmatrix} BB^T & 0 \\ 0 & B^T B \end{bmatrix}$$

and from this it follows that, if the dominant invariant subspaces associated to $B^T B$ and BB^T have dimension one, then the normalized hub and authority scores are simply given by the normalized dominant eigenvectors of $B^T B$ and BB^T , respectively. This is the definition used in [10] for the authority and hub scores of the vertices of G . The arbitrary choice of $z_0 = \mathbf{1}$ made in [10] is given here an extremal norm justification. Notice that when the invariant subspace has dimension one, then there is nothing special about the starting vector $\mathbf{1}$ since any other positive vector z_0 would give the same result.

We now generalize this construction. The authority score of the vertex j of G can be seen as a similarity score between j and the vertex *authority* in the graph

$$hub \longrightarrow authority$$

and, similarly, the hub score of j can be seen as a similarity score between j and the vertex *hub*. The mutually reinforcing updating iteration used above can be generalized to graphs that are different from the hub-authority structure graph. The idea of this generalization is quite simple; we illustrate it in this introduction on the path graph with three vertices and provide a general definition for arbitrary graphs in Section 3. Let G be a graph with edge set E and adjacency matrix B and consider the *structure graph*

$$1 \longrightarrow 2 \longrightarrow 3.$$

To the vertex j of G we associate three scores x_{j1}, x_{j2} and x_{j3} ; one for each vertex of the structure graph. We initialize these scores at some positive value and then update them according to the following mutually reinforcing relations

$$\begin{cases} x_{j1} \leftarrow \sum_{i:(j,i) \in E} x_{i2} \\ x_{j2} \leftarrow \sum_{i:(i,j) \in E} x_{i1} + \sum_{i:(j,i) \in E} x_{i3} \\ x_{j3} \leftarrow \sum_{i:(i,j) \in E} x_{i2} \end{cases}$$

or, in matrix form (we denote by x_i the column vector with entries x_{ji}),

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{k+1} = \begin{bmatrix} 0 & B & 0 \\ B^T & 0 & B \\ 0 & B^T & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_k, \quad k = 0, 1, \dots$$

which we again denote $x_{k+1} = Mx_k$. The situation is now identical to that of the previous example and all convergence arguments given there apply here as

well. The matrix M^2 is symmetric and non-negative definite, the normalized even and odd iterates converge, and the limit $z_{\text{even}}(\mathbf{1})$ is among all possible limits one that has largest possible 1-norm. We take the three components of this extremal limit $z_{\text{even}}(\mathbf{1})$ as definition for the similarity scores¹ s_1, s_2 and s_3 and define the similarity matrix by

$$\mathbf{S} = [s_1 \ s_2 \ s_3].$$

The rest of this paper is organized as follows. In Section 2, we describe some standard Perron-Frobenius results for non-negative matrices that will be useful in the rest of the paper. In Section 3, we give a precise definition of the similarity matrix together with different alternative definitions. The definition immediately translates into an approximation algorithm. In Section 4 we describe similarity matrices for the situation where one of the two graphs is a path graph; path graphs of lengths 2 and 3 are those that are discussed in this introduction. In Section 5, we consider the special case $G_A = G_B = G$ for which the score s_{ij} is the similarity between the vertices i and j in the graph G . Section 6 deals with graphs for which all vertices play the same rôle. We prove that, as expected, the similarity matrix in this case has rank one.

2 Graphs and Non-negative Matrices

With any directed graph $G = (V, E)$ one can associate a non-negative matrix via an indexation of its vertices. The so-called *adjacency matrix* of G is the matrix $B \in \mathbf{N}^{n \times n}$ whose entry d_{ij} equals the number of edges from vertex i to vertex j . Conversely, a square matrix B whose entries are non-negative integer numbers, defines a directed graph G with d_{ij} edges between i and j . Let B be the adjacency matrix of some graph G ; the entry $(B^k)_{ij}$ is equal to the number of paths of length k from vertex i to vertex j . From this it follows that a graph is strongly connected if and only if for every pair of indices i and j there is an integer k such that $(B^k)_{ij} > 0$. Matrices that satisfy this property are said to be *irreducible*.

The Perron-Frobenius theory [8] establishes interesting properties about the eigenvectors and eigenvalues for non-negative and irreducible matrices. Let us denote the spectral radius² of the matrix C by $\rho(C)$. The following results follow from [8].

Theorem 1. *Let C be a non-negative matrix. Then*

- (i) *the spectral radius $\rho(C)$ is an eigenvalue of C – called the Perron root – and there exists an associated non-negative vector $x \geq 0$ ($x \neq 0$) – called the Perron vector – such that $Cx = \rho x$*
- (ii) *if C is irreducible, then the algebraic multiplicity of the Perron root ρ is equal to one and there is a positive vector $x > 0$ such that $Cx = \rho x$*

¹ In Section 4, we prove that the “central similarity score” s_2 can be obtained more directly from B by computing the dominating eigenvector of the matrix $BB^T + B^T B$.

² The spectral radius of a matrix is the largest magnitude of its eigenvalues.

(iii) if C is symmetric, then the algebraic and geometric multiplicity of the Perron root ρ are equal and there is a non-negative basis $X \geq 0$ associated to the invariant subspace associated to ρ , such that $CX = \rho X$.

In the sequel, we shall also need the notion of orthonormal projection on subspaces. Let $\mathcal{V}$ be a linear subspace of $\mathbf{R}^n$ and let $v \in \mathbf{R}^n$. The *orthogonal projection* of v on $\mathcal{V}$ is the vector in $\mathcal{V}$ with smallest distance to v . The matrix representation of this projection is obtained as follows. Let $\{v_1, \dots, v_m\}$ be an orthogonal basis for $\mathcal{V}$ and arrange these column vectors in a matrix V . The projection of v on $\mathcal{V}$ is then given by $\Pi v = VV^T v$ and the matrix $\Pi = VV^T$ is the *orthogonal projector* on $\mathcal{V}$. From the previous theorem it follows that, if the matrix C is non-negative and symmetric, then the elements of the orthogonal projector Π on the vector space associated to the Perron root of C are all non-negative.

The next theorem will be used to justify our definition of similarity matrix between two graphs. The result describes the limit points of sequences associated with symmetric non-negative linear transformations.

Theorem 2. *Let M be a symmetric non-negative matrix of spectral radius ρ . Let $z_0 > 0$ and consider the sequence*

$$z_{k+1} = Mz_k / \|Mz_k\|_2, \quad k = 0, \dots$$

Then the subsequences z_{2k} and z_{2k+1} converge to the limits

$$z_{\text{even}}(z_0) = \lim_{k \rightarrow \infty} z_{2k} = \frac{\Pi z_0}{\|\Pi z_0\|_2} \quad \text{and} \quad z_{\text{odd}}(z_0) = \lim_{k \rightarrow \infty} z_{2k+1} = \frac{\Pi M z_0}{\|\Pi M z_0\|_2},$$

where Π is the orthogonal projector on the invariant subspace of M^2 associated to its Perron root ρ^2 . In addition to this, the set of all possible limits is given by:

$$Z = \{z_{\text{even}}(z_0), z_{\text{odd}}(z_0) : z_0 > 0\} = \{\Pi z / \|\Pi z\|_2 : z > 0\}$$

and the vector $z_{\text{even}}(\mathbf{1})$ is the unique vector of largest 1-norm in that set.

3 Similarity between Vertices in Graphs

We now introduce our definition of graph similarity for arbitrary graphs. Let G_A and G_B be two directed graphs with respectively n_A and n_B vertices. We think of G_A as a “structure graph” that plays the role of the graphs *hub* $\rightarrow$ *authority* and $1 \rightarrow 2 \rightarrow 3$ in the introductory examples. Let $\text{pre}(v)$ (respectively $\text{post}(v)$) denote the set of ancestors (respectively descendants) of the vertex v . We consider real scores x_{ij} for $i = 1, \dots, n_B$ and $j = 1, \dots, n_A$ and simultaneously update all scores according to the following updating equations

$$[x_{ij}]_{k+1} = \sum_{r \in \text{pre}(i), s \in \text{pre}(j)} [x_{rs}]_k + \sum_{r \in \text{post}(i), s \in \text{post}(j)} [x_{rs}]_k \quad (1)$$

These equations coincide with those given in the introduction. The equations can be written in more compact matrix form. Let X_k be the $n_B \times n_A$ matrix of entries $[x_{ij}]_k$. Then (1) takes the form

$$X_{k+1} = BX_k A^T + B^T X_k A, \quad k = 0, 1, \dots \quad (2)$$

where A and B are the adjacency matrices of G_A and G_B . In this updating equation, the entries of X_{k+1} depend linearly on those of X_k . We can make this dependance more explicit by using the matrix-to-vector operator that develops a matrix into a vector by taking its columns one by one. This operator, denoted vec , satisfies the elementary property $\text{vec}(CXD) = (D^T \otimes C) \text{vec}(X)$ in which $\otimes$ denotes the Kronecker tensorial product (for a proof of this property, see Lemma 4.3.1 in [9]). Applying this property to (2) we immediately obtain

$$x_{k+1} = (A \otimes B + A^T \otimes B^T) x_k \quad (3)$$

where $x_k = \text{vec}(X_k)$. This is the format used in the introduction. Combining this observation with Theorem 2 we deduce the following property for the normalized sequence Z_k .

Corollary 1. *Let G_A and G_B be two graphs with adjacency matrices A and B , select an initial positive matrix $Z_0 > 0$ and define*

$$Z_{k+1} = \frac{BZ_k A^T + B^T Z_k A}{\|BZ_k A^T + B^T Z_k A\|_2} \quad k = 0, 1, \dots$$

Then, the matrix subsequences Z_{2k} and Z_{2k+1} converge to Z_{even} and Z_{odd} . Moreover, among all the matrices in the set

$$\{Z_{\text{even}}(Z_0), Z_{\text{odd}}(Z_0) : Z_0 > 0\}$$

the matrix $Z_{\text{even}}(\mathbf{1})$ is the unique matrix of largest 1-norm.

In order to be consistent with the vector norm appearing in Theorem 2, the matrix norm $\|\cdot\|_2$ we use here is the square root of the sum of all squared entries (this norm is known as the Euclidean or Frobenius norm), and the 1-norm $\|\cdot\|_1$ is the sum of all entries magnitudes. In view of this result, the next definition is now justified.

Definition 1. *Let G_A and G_B be two graphs with adjacency matrices A and B . The similarity matrix between G_A and G_B is the matrix*

$$\mathbf{S} = \lim_{k \rightarrow +\infty} Z_{2k}$$

obtained for $Z_0 = \mathbf{1}$ and

$$Z_{k+1} = \frac{BZ_k A^T + B^T Z_k A}{\|BZ_k A^T + B^T Z_k A\|_2}, \quad k = 0, 1, \dots$$

A direct algorithmic transcription of the definition leads to an approximation algorithm. An example of a pair of graphs and their corresponding similarity matrix is given in Figure 3. Notice that it follows from the definition that the similarity matrix between G_B and G_A is the transpose of the similarity matrix between G_A and G_B . Similarity matrices can alternatively be defined as the projection of the matrix $\mathbf{1}$ on an invariant subspace associated to the graphs and for particular classes of adjacency matrices, one can compute the similarity matrix $\mathbf{S}$ directly from the dominant invariant subspaces of matrices of the size of A or B ; we provide explicit expressions for a few classes in the next sections. Similarity matrices can also be defined by their extremal property.

Corollary 2. *The similarity matrix of the graphs G_A and G_B of adjacency matrices A and B is the unique matrix of largest 1-norm among all matrices X that maximize the expression*

$$\frac{\|BXA^T + B^T XA\|_2}{\|X\|_2}. \quad (4)$$

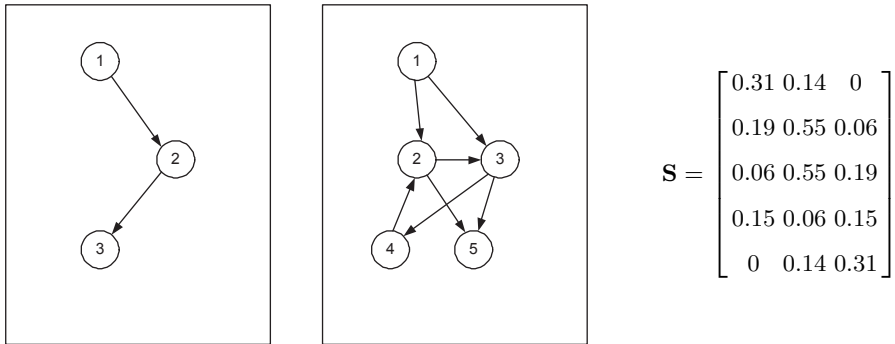


Fig. 1. Two graphs G_A and G_B and their corresponding similarity matrix $\mathbf{S}$. As an illustration, the similarity score between vertex 2 of graph G_A and vertex 3 of graph G_B is equal to 0.55.

4 Hubs, Authorities, Central Scores, and Path Graphs

As explained in the introduction, the hub and authority scores of a graph G_B can be expressed in terms of the adjacency matrix of G_B .

Theorem 3. *Let B be the adjacency matrix of the graph G_B . The normalized hub and authority scores of the vertices of G_B are given by the normalized dominant eigenvectors of the matrices $B^T B$ and BB^T , provided the corresponding Perron root is of multiplicity 1. Otherwise, it is the normalized projection of the vector $\mathbf{1}$ on the respective dominant invariant subspaces.*

The condition on the multiplicity of the Perron root is not superfluous. Indeed, even for strongly connected graphs, BB^T and B^TB may have multiple dominant roots: for cycle graph for example, both BB^T and B^TB are the identity matrix. Another interesting structure graph is the path graph of length three:

$$1 \longrightarrow 2 \longrightarrow 3$$

Similarly to the hub and authority scores, the resulting similarity score with vertex 2, a score that we call *central score*, can be given an explicit expression.

Theorem 4. *Let B be the adjacency matrix of the graph G_B . The normalized central scores of the vertices of G_B are given by the normalized dominant eigenvector of the matrix*

$$B^TB + BB^T,$$

provided the corresponding Perron root is of multiplicity 1. Otherwise, it is the normalized projection of the vector $\mathbf{1}$ on the dominant invariant subspace.

The above structure graphs are path graphs of length 2 and 3. For path graphs of arbitrary length ℓ we have:

Corollary 3. *Let B be the adjacency matrix of the graph G_B . Let G_A be the path graph of length ℓ :*

$$G_A : 1 \longrightarrow 2 \longrightarrow \cdots \longrightarrow \ell.$$

Then the odd and even columns of the similarity matrix $\mathbf{S}$ can be computed independently as the projection of $\mathbf{1}$ on the dominant invariant subspaces of EE^T and E^TE where

$$E = \begin{bmatrix} B & & & \\ B^T & \ddots & & \\ & \ddots & B & \\ & & B^T & B \end{bmatrix} \quad \text{or} \quad E = \begin{bmatrix} B & & & \\ B^T & \ddots & & \\ & \ddots & B & \\ & & B^T & \end{bmatrix}$$

for ℓ even and ℓ odd, respectively.

5 Self-Similarity Matrix of a Graph

When we compare two equal graphs $G_A = G_B = G$, the similarity matrix $\mathbf{S}$ is a square matrix whose entries are similarity scores between vertices of G ; this matrix is the *self-similarity matrix* of G . Various graphs and their corresponding self-similarity matrices are represented in Figure 2. In general, we expect vertices to have a high similarity score with themselves; that is, we expect the diagonal entries of self-similarity matrices to be large. We prove in the next theorem that the largest entry of a self-similarity matrix always appear on the diagonal and that, except for trivial cases, the diagonal elements of a self-similarity matrix are non-zero. As is shown with the last graph of Figure 2, it is however not true that diagonal elements dominate all elements on the same line and column.

Theorem 5. *The self-similarity matrix of a graph is positive semi-definite. In particular, the largest element of the matrix always appears on diagonal, and if a diagonal entry is equal to zero, then the corresponding line and column are equal to zero.*

For some classes of graphs, similarity matrices can be computed explicitly. We have for example:

Theorem 6. *The self-similarity matrix of the path graph of length ℓ is a diagonal matrix with diagonal elements equal to $\sin(j\pi/(\ell + 1))$, $j = 1, \dots, \ell$.*

When vertices of a graph are similar to each other, such as in cycle graphs, we expect to have a self-similarity matrix whose entries are all equal. This is indeed the case. Let us recall here that a graph is said to be *vertex-transitive* (or *vertex symmetric*) if all vertices play the same rôle in the graph. More formally, a graph G of adjacency matrix A is vertex-transitive if associated to any pair of vertices i, j , there is a permutation matrix T that satisfies $T(i) = j$ and $T^{-1}AT = A$.

Theorem 7. *All entries of the self-similarity matrix of a vertex-transitive graph are equal to $1/n$.*

6 Graphs Whose Vertices Are Symmetric to Each Other

We now analyze properties of the similarity matrix when one of the two graphs has all its vertices symmetric to each other, or has an adjacency matrix that is normal. We prove that in both cases the resulting similarity matrix has rank one.

Theorem 8. *Let G_A, G_B be two graphs and assume that G_A is vertex-transitive. Then the similarity matrix between G_A and G_B is a rank one matrix of the form*

$$\mathbf{S} = \alpha \mathbf{1}v^T$$

where $v = \Pi \mathbf{1}$ is the projection of $\mathbf{1}$ on the dominant invariant subspace of $(B + B^T)^2$ and α is the scaling factor $\alpha = 1/\|\mathbf{1}v^T\|$. In particular, if G_A and G_B are both vertex symmetric then the entries of their similarity matrix are all equal to $1/\sqrt{n_A n_B}$.

Cycle graphs have an adjacency matrix A that satisfies $AA^T = I$. This property corresponds to the fact that, in a cycle graph, all forward-backward paths from a vertex return to that vertex. More generally, we consider in the next theorem graphs that have an adjacency matrix A that is normal, i.e., such that $AA^T = A^T A$. In particular, graphs that have a symmetric adjacency matrix satisfy this property. We prove below that when one of the graphs has a normal adjacency matrix, then the similarity matrix has rank one and we provide an explicit expression for this matrix.

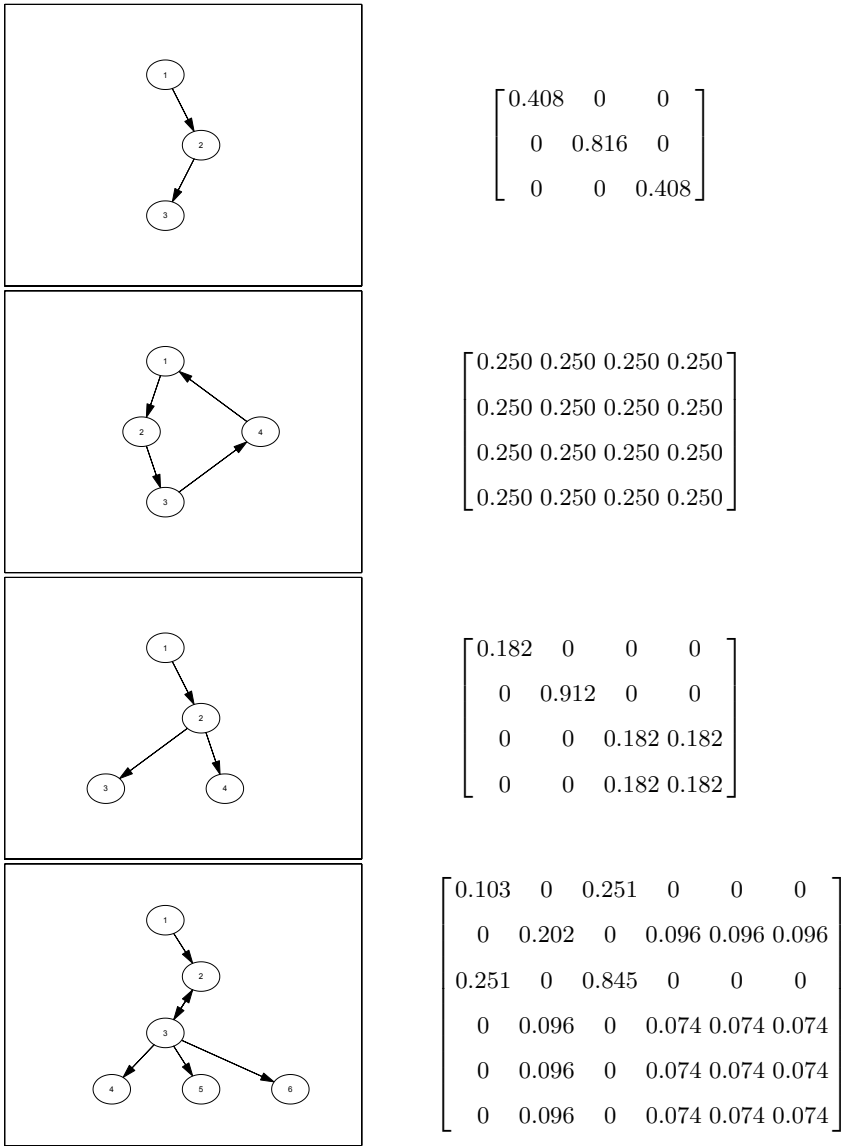


Fig. 2. Graphs and their corresponding self-similarity matrices.

Theorem 9. Let G_A and G_B be two graphs and assume that A is a normal matrix. Then the similarity matrix between G_A and G_B is a rank one matrix $S = uv^T$ where

$$u = \frac{(\Pi_{+\alpha} + \Pi_{-\alpha})\mathbf{1}}{\|(\Pi_{+\alpha} + \Pi_{-\alpha})\mathbf{1}\|_2}, \quad v = \frac{\Pi_\beta \mathbf{1}}{\|\Pi_\beta \mathbf{1}\|_2}.$$

In this expression α is the Perron root of A , $\Pi_{+\alpha}$, $\Pi_{-\alpha}$ are the projectors on its invariant subspaces corresponding to the eigenvalues $+\alpha$ and $-\alpha$, β is the Perron root of $(B + B^T)$, and Π_β is the projector on the invariant subspace of $(B + B^T)^2$ corresponding to the eigenvalue β^2 .

When one of the graphs G_A or G_B is vertex-transitive or has a normal adjacency matrix, the resulting similarity matrix $\mathbf{S}$ has rank one. Adjacency matrices of vertex-transitive graphs and normal matrices have the property that the projector $\Pi_{+\alpha}$ on the invariant subspace corresponding to the Perron root of A is also the projector on the subspace of A^T (and similarly for $-\alpha$). We conjecture here that the similarity matrix can only be of rank one if either A or B have this property.

7 Concluding Remarks

Investigations of properties and applications of the similarity matrix of graphs can be pursued in several directions. We outline here some possible research directions.

One natural extension of our concept is to consider networks rather than graphs; this amounts to consider adjacency matrices with arbitrary real entries and not just integers. The definitions and results presented in this paper use only the property that the adjacency matrices involved have non-negative entries, and so all results remain valid for networks with non-negative weights. The extension to networks makes a sensitivity analysis possible: How sensitive is the similarity matrix to the weights in the network? Experiments and qualitative arguments show that, for most networks, similarity scores are almost everywhere continuous functions of the network entries. Perhaps this can be analyzed for models for random graphs such as those that appear in [3]? These questions can probably also be related to the large literature on eigenvalues and eigenspaces of graphs; see, e.g., [4], [5] and [6].

More specific questions on the similarity matrix also arise. One open problem is to characterize the pairs of matrices that give rise to a rank one similarity matrix. The structure of these pairs is conjectured at the end of Section 6. Is this conjecture correct? A long-standing graph question also arise when trying to characterize the graphs whose similarity matrices have only positive entries. The positive entries of the similarity matrix between the graphs G_A and G_B can be obtained as follows. One construct the product graph, symmetrize it, and then identify in the resulting graph the connected component(s) of largest possible Perron root. The indices of the vertices in that graph correspond exactly to the nonzero entries in the similarity matrix of G_A and G_B . The entries of the similarity matrix will thus be all positive if and only if the product graph of G_A and G_B is weakly connected. The problem of characterizing all pairs of graphs that have a weakly connected product was introduced and analyzed in 1966 in [7]. The problem of efficiently characterizing all pairs of graphs that have a weakly connected product is a problem that is still open.

Another topic of interest is to investigate how the concepts proposed here can be used, possibly in modified form, for evaluating the similarity between two graphs, for clustering vertices or graphs, for pattern recognition in graphs or for data mining purposes.

Acknowledgment. Three of our students, Maureen Heymans, Anahí Gajardo and Pierre Senellart have provided inputs on several ideas developed in this paper. We are pleased to acknowledge the inputs of all these students. This paper presents research supported by NSF under Grant No. CCR 99-12415 and by the Belgian Programme on Inter-university Poles of Attraction, initiated by the Belgian State, Prime Minister's Office for Science, Technology and Culture.

References

1. Vincent D. Blondel, Paul Van Dooren, A measure of similarity between graph vertices. With applications to synonym extraction and web searching. Technical Report UCL 02-50, submitted to journal, 2002.
2. Vincent D. Blondel, Pierre P. Senellart, Automatic extraction of synonyms in a dictionary, Technical report 2001-89, Université catholique de Louvain, Louvain-la-Neuve, Belgium. Also: Proceedings of the SIAM Text Mining Workshop, Arlington (Virginia, USA) April 11, 2002.
3. B. Bollobas, *Random Graphs*, Academic Press, 1985.
4. Fan R. K. Chung, *Spectral Graph Theory*, American Mathematical Society, 1997.
5. Dragoš Cvetković, Peter Rowlinson, Slobodan Simić, *Eigenspaces of graphs*, Cambridge University Press, 1997.
6. Dragoš Cvetković, M. Doob, H. Sachs, *Spectra of graphs – Theory and applications* (Third edition), Johann Ambrosius Barth Verlag, 1995.
7. Frank Harary, C. Trauth, Connectedness of products of two directed graphs, *J. SIAM Appl. Math.*, 14, pp. 150–154, 1966.
8. R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, London, 1985.
9. R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*, Cambridge University Press, London, 1991.
10. Jon M. Kleinberg, Authoritative sources in a hyperlinked environment, *Journal of the ACM*, 46:5, pp. 604–632, 1999.
11. Pierre P. Senellart, Vincent D. Blondel, Automatic discovery of similar words. To appear in “A Comprehensive Survey of Text Mining”, Springer-Verlag, 2003.

Algorithmic Aspects of Bandwidth Trading

Randeep Bhatia¹, Julia Chuzhoy², Ari Freund², and Joseph (Seffi) Naor²

¹ Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974

`randeep@research.bell-labs.com`

² Computer Science Dept., Technion, Haifa 32000, Israel

`{cjulia, arief, naor}@cs.technion.ac.il`

Abstract. We study algorithmic problems that are motivated by bandwidth trading in next generation networks. Typically, bandwidth trading involves sellers (e.g., network operators) interested in selling bandwidth pipes that offer to buyers a guaranteed level of service for a specified time interval. The buyers (e.g., bandwidth brokers) are looking to procure bandwidth pipes to satisfy the reservation requests of end-users (e.g., Internet subscribers). Depending on what is available in the bandwidth exchange, the goal of a buyer is to either spend the least amount of money to satisfy all the reservations made by its customers, or to maximize its revenue from whatever reservations can be satisfied.

We model the above as a real-time non-preemptive scheduling problem in which machine types correspond to bandwidth pipes and jobs correspond to the end-user reservation requests. Each job specifies a time interval during which it must be processed and a set of machine types on which it can be executed. If necessary, multiple machines of a given type may be allocated, but each must be paid for. Finally, each job has a revenue associated with it, which is realized if the job is scheduled on some machine.

There are two versions of the problem that we consider. In the cost minimization version, the goal is to minimize the total cost incurred for scheduling all jobs, and in the revenue maximization version the goal is to maximize the revenue of the jobs that are scheduled for processing on a given set of machines. We consider several variants of the problems that arise in practical scenarios, and provide constant factor approximations.

Keywords: Scheduling, bandwidth trading, approximation algorithms, primal-dual schema.

1 Introduction

We study algorithmic problems involving *bandwidth trading* in next generation networks. As network operators are building new high-speed networks, they look for new ways to sell or lease their plentiful bandwidth. At the same time there are emerging potential buyers of bandwidth, such as virtual network carriers, who would like to be able to expand capacity easily and rapidly to meet the ever-changing demands of their customers. Similarly, many companies are looking for ways to be able to reserve bandwidth for on-off events such as video-conferences.

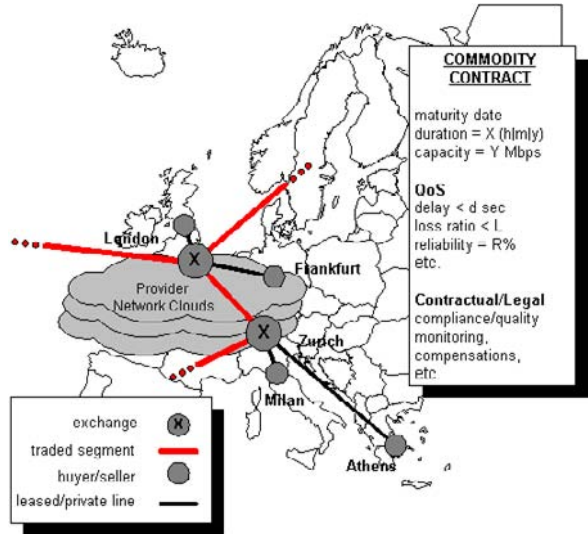
Finally, there are network subscribers who would like to be able to buy bandwidth for the duration of a webcast, pay-per-movie on the web, etc. In this paper we consider some of the algorithmic problems arising in this context.

1.1 Bandwidth Trading in Practice

Our work is motivated by the emerging business model being discussed in the networking community, which we briefly describe here. (More details can be found, for example, at [19].) Although bandwidth exchange/trading is not new, the traditional methodology is often marred by long periods of binding contracts and slow provisioning time. With the recent advances in network technologies, however, there has been a tremendous leap forward both in network capacity and provisioning times. It is now possible to quickly provision end-to-end protocol-independent light paths with a specified Service Level Agreement (SLA) that takes into account QoS, bandwidth, restoration level, etc., in order to rapidly meet the changing bandwidth demands of network users. In addition, many identical low bandwidth data streams can be multiplexed over a single light path in the core network, thus enabling a core network operator to sell bandwidth at smaller granularities.

Driven to meet demands for high-speed data-centric applications, various up-start network carriers have been rolling out networks with vast amounts of excess capacity. With all this capacity up for grabs, a new generation of resellers, wholesalers, and on-line bandwidth brokers are poised to resell it to customers. Leading the pack in the bandwidth commodity effort are carriers such as Williams and a host of real-time on-line trading centers pioneered by the likes of Band-X and RateXchange.

Typically, bandwidth trading involves a *bandwidth exchange* which includes a marketplace for suppliers and buyers of bandwidth and a set of *pooling points* which are used for actually providing the bandwidth upon settlement. Physically, a pooling point may be a fiber interconnection and switching site in a particular geographical location, with co-located points of presence for buyers (e.g. ISPs) and suppliers (e.g. long haul carriers). In the pooling point, the buyer's network interfaces with the supplier's high-speed optical network, and data passing between the two is converted from electrical packets to optical signal and vice versa. It is assumed that a bandwidth exchange trades well-defined band-



width contracts [6,12]. Each contract refers to a *bandwidth segment* between two pooling points, where a bandwidth segment is an abstraction of one or more high-capacity networks providing connectivity between the two pooling points. Each bandwidth contract describes the duration for which connectivity will be made available as well as a Service Level Agreement (SLA) that takes into account QoS, bandwidth, restoration level, etc., The inset, taken from the web site of IBM Zürich Research Lab (<http://www.zurich.ibm.com/bandwidth/concepts.html>—the section titled QoS), summarizes the situation by showing an example of a bandwidth segment being offered between two pooling points connected to buyers' networks.

Optical technologies play a central role in next generation networks. A single strand of optical fiber is now capable of carrying a large number of high bandwidth data streams, each of which can be individually managed. Typically, a low bandwidth data stream is dedicated to a single end-user at any given time, but may be shared over time by multiple end-users, where the switch from one user to another is provisioned almost instantaneously. A core optical network operator is therefore able to trade a large number of identical bandwidth contracts (with same attributes), each corresponding to a low bandwidth data stream, all of which are multiplexed over a single high bandwidth light path. For practical purposes, one can therefore assume that an unlimited number of “copies” of each contract are available.

Finally, buyers are themselves service providers to their clients—the end users. End users generate bandwidth requests which are called *forward reservations*. Each forward reservation specifies the two endpoints (which translate into two pooling points) between which the bandwidth reservation is required, the time interval for which it is required, any other attributes of the connection (QoS, restoration level, etc.), and the revenue obtained by honoring the reservation. Buyers in the bandwidth exchange, who may be ISPs, bandwidth brokers, etc., are looking to procure bandwidth contracts to satisfy at the cheapest cost the forward reservation requests made by their clients (e.g., network subscribers, companies, or virtual network operators). A single procured bandwidth contract can be used to serve a set of “non-overlapping” forward reservation requests. Depending on what is available in the bandwidth exchange, the goal of a buyer is to either spend the least amount of money to buy enough bandwidth contracts to satisfy all the reservations, or, if the reservations cannot be all satisfied, to maximize its revenue from whatever reservations are possible.

Combining Contracts. Given a bandwidth exchange, a *contract graph* [6,12,19] is defined to be a graph whose nodes are the pooling points and edges represent the traded contracts. Several point-to-point segments (on a path in the graph) can be assembled to connect any two geographical locations. This leads to a new (path) contract whose attributes depend on the choice of the path in the contract graph. We stress that the new contract is *indivisible*. For example, consider three pooling points A , B , and C . Suppose that an (A, B) contract and a (B, C) contract are combined into an (A, C) contract. Then, the (A, C) contract cannot be used to also route traffic from A to B or from B to C , since this would require optical-to-electrical followed by an electrical-to-optical conversion

at point B . Such conversions introduce substantial delays at the intermediate points and deteriorate end-to-end QoS, thus defeating the purpose of high-speed optical routing.

In general, we can assume that if a pair of pooling points have a point-to-point contract between them, then that is the cheapest way to connect the two points (for those attributes), since we consider a highly liquid bandwidth market in which arbitrage opportunities [7] are instantaneously removed. (A geographic arbitrage arises, for example, if the price of an indivisible point-to-point contract between New York and London is more than the price of a path contract with the same attributes that goes via Los Angeles.)

1.2 Wavelength Assignment in Optical Line Systems

Another motivation for the problems we consider comes from wavelength assignment in optical line systems [20] such as those involving DWDMs. An optical line system is a collection of nodes called Mesh Optical Add and Drop Multiplexers (MOADM) arranged in a line, with adjacent nodes connected by optical fibers. A demand enters the line system at one node and exits at some other node, and is routed on the same wavelength on the fibers connecting all the intermediate nodes. The set of wavelengths available on each fiber connecting two adjacent MOADM (say node i and $i+1$) may differ from fiber to fiber, and it is a function of the fiber characteristics, and also of the wavelengths which have been used up by previously provisioned demands. Given a set of demands, the problem is to assign wavelengths to them so that no two demands use the same wavelength on the same fiber.

We note that any optical line system as described above can be viewed as a set of windows, $\mathcal{I}$, where each $I \in \mathcal{I}$ is an interval corresponding to a single wavelength which is available between the two end-points of I . Given a set of demands D (i.e., D is a set of intervals), the wavelength assignment problem corresponds to packing the intervals belonging to D into $\mathcal{I}$ such that intervals packed into a window $I \in \mathcal{I}$ do not overlap.

1.3 Model Description, Notation, and Terminology

We model bandwidth trading as a real-time scheduling problem. As explained above, a bandwidth contract can only be used for routing traffic between its two end points. Therefore, we only need to consider the bandwidth trading problem for a single pair of pooling points. We view the bandwidth contracts as a set of machines of different types, where identical bandwidth contracts (with the same attributes) correspond to machines of the same type. Each machine type has a *cost* per machine. We can assume that an unlimited number of machines are available from each type, since the available bandwidth on the light paths in the core network is greater by many orders of magnitude than the end-user bandwidth requirement for any single data stream. The jobs correspond to reservation requests made by end users. Each job needs to be processed during a specified time interval, and it can only be processed on a machine of one of several specified types. Each job has a value associated with it corresponding to

the revenue obtained for processing the job. At most one job can be scheduled on any machine at any given time (since no two overlapping reservation requests can be served by the same bandwidth contract).

In the *cost minimization* version of the problem, the goal is to find a set of machines of minimum cost so as to be able to schedule all the jobs on the machines. In this version of the problem we ignore the job revenues. In the *revenue maximization* version, the goal is to maximize the total revenue by selecting a subset of the jobs that can be scheduled using a given set of machines. In this version of the problem we ignore the machine costs.

Formally, we have a set of m machine types $\mathcal{T} = \{T_1, \dots, T_m\}$. A *cost*, or *weight*, $w(T_i) \geq 0$, is incurred for allocating a machine of type T_i . There are n jobs belonging to an input set of jobs J , where each job j is associated with the following parameters: a revenue $w(j) \geq 0$, a set $S(j) \subseteq \mathcal{T}$ of machine types on which this job can be processed, and a time interval $I(j)$ during which the job is to be processed. We sometimes refer to a job and its interval interchangeably. At most one job can be processed on a given machine at any given moment.

The Problems. The general version of the cost minimization problem, where the sets $S(j)$ of machine types are arbitrary, is essentially equivalent (approximation-wise) to *set cover*. (Hardness can be shown by a simple reduction in which set elements become non-overlapping jobs and each set becomes a machine capable of processing only the jobs corresponding to the set's elements. Logarithmic approximability was shown by Jansen [11].) In practice, however, the definition of the sets $S(j)$ is usually based on properties of the machines and thus has a computationally more convenient structure. We consider two variants arising naturally in bandwidth trading and other real world applications.

Cost minimization with machine time intervals. Machine types are defined by time intervals. Each type T_i is associated with a time interval $I(T_i)$ during which machines of this type are available. A job can be processed by every machine that is available throughout its processing interval. Thus the sets $S(j)$ are defined by: $S(j) = \{T_i \in \mathcal{T} \mid I(j) \subseteq I(T_i)\}$. In the *unweighted* case all types have unit cost ($w(T_i) = 1$ for all i), and in the *weighted* case costs are arbitrary.

Cost minimization with machine strengths. There is a linear order of *strength* defined on the machine types, $T_1 \prec T_2 \prec \dots \prec T_m$, such that a job that may be processed by a machine of a given type may also be processed on every stronger machine, i.e., for all jobs j , $S(j)$ has the form $\{T_{i_j}, T_{i_j+1}, \dots, T_m\}$. We also assume that the stronger a machine is, the higher its cost (otherwise there is no point in using weaker machines). The linear order models a situation in which the SLA's are contained in each other in terms of the capabilities they specify.

We comment that no bounded approximation factors are possible for these problems (unless $P = NP$) if only a limited number of machines is available from each type. This follows by observing that it is NP-hard even to decide whether all the jobs can be scheduled on all available machines (by a simple reduction from the circular arc coloring problem which is NP-hard [9]).

We also consider the revenue maximization version.

Revenue maximization. We are given a collection $\mathcal{M}$ of machines (presumably, already paid for) and we wish to select a maximum revenue subset of the jobs that can be scheduled on these machines. Every job j specifies an arbitrary set, $S(j) \subseteq \mathcal{M}$, of machines on which it can be scheduled.

1.4 Our Contribution

The problems we consider are NP-hard. We present the first polynomial-time constant-factor approximation algorithms for both versions of the problem (cost minimization and revenue maximization).

In Section 2 we consider the cost minimization problem with machine time intervals. We describe a 3-approximation algorithm for the weighted case and a 2-approximation algorithm for the unweighted case. We remark that our 3-approximation algorithm for the weighted case can be extended to a *prize collecting* version of the problem, where it is not necessary to schedule all the jobs but there is a fine to be paid for each unscheduled job. We defer details to the full paper.

Our algorithm for the weighted case is based on a linear programming relaxation and it is a novel variant of the (combinatorial) primal-dual schema [18]. It is quite unique in that it copes with the difficulty posed by a constraint matrix containing both positive and negative entries. It is an interesting fact that the primal-dual schema is often incapable of dealing with both positive and negative coefficients. Our algorithm is also unconventional in that it departs from the common dual-ascent, reverse-delete paradigm. Rather than generating a minimal solution via a reverse-delete stage, we iteratively improve our schedule by a selective rescheduling of jobs which obviates some of the machines in our schedule. In addition, each iteration in our dual ascent stage increases several (but not all) of the dual variables. This contrasts with algorithms such as Goemans and Williamson's *network design* algorithm [10], where *all* dual variables are increased uniformly in each iteration, or Bar-Yehuda and Even's *vertex cover* algorithm [5], where a single dual variable is increased in each iteration.

In Section 3, we show a 2-approximation algorithm for the cost minimization problem with machine strengths. Both this algorithm and the algorithm for the unweighted version of the cost minimization problem with machine time intervals employ simple combinatorial lower bounds on the problems.

We conclude with the revenue maximization problem in Section 4. We present a $(1 - 1/e)$ -approximation algorithm for this problem. This result improves on the approximation factor of $1/2$ implicit in [2] for this problem.

1.5 Related Work

Kolen and Kroon [13,15] and Jansen [11] considered the same scheduling model as ours in the context of aircraft maintenance. When aircraft arrive at the airport, they must be inspected by ground engineers between their arrivals and departures. There are different types of aircraft, and different types of engineer

licenses, and there is an aircraft-type/engineer-license-type matrix specifying which license type permits an engineer to work on which aircraft type. In addition, the engineers work in shifts. The cost of assigning an engineer to an aircraft inspection job depends on the engineer's license type and the shift during which the inspection must be carried out. The goal is to enlist a minimum-cost multi-set of "engineer instances" to handle all aircraft inspections, where an engineer instance is defined by a $(\textit{license}, \textit{shift})$ pair. In our model, jobs correspond to inspections of aircraft, and machine types correspond to $(\textit{license}, \textit{shift})$ pairs.

Kolen and Kroon [13] study the computational complexity of this problem with respect to different aircraft/engineers matrices, when all the shifts are the same. In particular, their work implies that the cost minimization problem we consider in Section 3 is NP-hard. In [15], Kolen and Kroon study another version of this problem, where all the aircraft and license types are the same, and there are different time shifts. They show that the problem is NP-hard even for unit costs, implying that the problems we consider in Sections 2.1 and 2.2 are NP-hard as well. Jansen [11] gives an $O(\log n)$ -approximation algorithm for the general problem, with both aircraft/license types and time shifts. When all the shifts are the same and all aircraft types are identical, the problem reduces to optimal coloring of interval graphs, and has a polynomial time algorithm [1].

Maximizing the throughput (*revenue* in our terminology) in real-time scheduling was studied extensively in [2,3,17,4,8]. They focused on the case where for each job, more than one time interval in which it can be performed is specified, while machines are available continuously. As here, jobs are scheduled non-preemptively and at each point of time only one job can be scheduled on a given machine. This model captures many applications, e.g., scheduling a space mission, bandwidth allocation, and communication in a linear network. The results of [2] on maximizing the throughput of unrelated parallel machines imply an approximation factor of $1/2$ for our revenue maximization problem. This result was improved in [8] to $(1 - 1/e - \epsilon)$ (for any constant ϵ) for the unweighted version of the problem. The revenue maximization problem with machine time intervals was studied by Kolen and Kroon [14] (see also Kolen and Lenstra [16, pp. 1901-1903]). They solved the problem optimally with a dynamic programming algorithm whose running time is $O(n^m)$. This implies that the problem is polynomial-time solvable for a constant number of machines.

The wavelength assignment problem in optical line systems is studied in [20]. Their result implies that the resulting interval packing problem (which is a decision version of our revenue maximization problem) as described in Section 1.2 is NP-complete.

2 Cost Minimization with Machine Time Intervals

In this section we develop approximation algorithms for the special case of the cost minimization problem where each machine type T_i has a time interval $I(T_i)$ during which the machines of this type are available. The sets $S(j)$ of machine types allowable for job j are defined as follows: $S(j) = \{T_i \in \mathcal{T} \mid I(j) \subseteq I(T_i)\}$. We present a 3-approximation algorithm for the weighted case and a 2-approximation algorithm for the unweighted case.

2.1 The Weighted Case

Our algorithm for the weighted case is based on the primal-dual schema for approximation algorithms. The linear programming formulation of the problem contains two sets of variables: $\{x_i\}$ and $\{y_{ij}\}$. For each machine type T_i , variable x_i represents the number of machines allocated of type T_i , and for every pair of machine type T_i and job j such that $I(j) \subseteq I(T_i)$, variable y_{ij} indicates whether job j is assigned to a machine of type T_i . We also use the following notation: E is the set of endpoints of jobs and $J(t)$ is the set of jobs whose intervals contain time t . The linear program is:

$$\text{Min} \quad \sum_{i=1}^m w(T_i)x_i \quad \text{s.t.}$$

$$\sum_{i=1}^n y_{ij} \geq 1, \quad \forall j \in J; \quad (1)$$

$$x_i - \sum_{j \in J(t)} y_{ij} \geq 0, \quad \forall 1 \leq i \leq m, \forall t \in E \cap I(T_i); \quad (2)$$

$$x, y \geq 0. \quad (3)$$

(The sums in Constraints (1) and (2) should be understood to include only variables y_{ij} that are defined.) The dual variables are $\{\alpha_j\}$ and $\{\beta_i^t\}$, corresponding to Constraints (1) and (2), respectively. The dual program is:

$$\text{Max} \quad \sum_{j \in J} \alpha_j \quad \text{s.t.}$$

$$\sum_{t \in E \cap I(T_i)} \beta_i^t \leq w(T_i), \quad \forall 1 \leq i \leq m; \quad (4)$$

$$\alpha_j - \sum_{t \in E \cap I(j)} \beta_i^t \leq 0, \quad \forall 1 \leq i \leq m, \forall j \text{ s.t. } I(j) \subseteq I(T_i); \quad (5)$$

$$\alpha, \beta \geq 0. \quad (6)$$

Our algorithm proceeds in two phases. In the first phase it constructs a feasible schedule by iteratively allocating machines and scheduling jobs on them. In the second phase it improves the solution by considering the allocated machines in reverse order and (possibly) eliminates some of them, rescheduling jobs as necessary.

Phase 1: dual ascent. As mentioned, the first phase allocates machines and schedules jobs. Accordingly, at a given moment during this phase there are *scheduled* and *unscheduled* jobs, and *allocated* and *un-allocated* machines and machine types. Initially all jobs are unscheduled, all machines and machine types are un-allocated, and all dual variables are set to 0.

The k th iteration in Phase 1 proceeds as follows. Let $t_k \in E$ be such that a maximum number of unscheduled jobs contain t_k , and let n_k denote the number of these jobs. Let $\mathcal{T}_k$ be the set of all un-allocated machine types whose intervals

contain t_k . We increase $\beta_i^{t_k}$ for all i such that $T_i \in \mathcal{T}_k$ uniformly at the same rate until some constraint of type (4) becomes tight, i.e., we increase each of the β s in question by $\delta_k = \min\{w(T_i) - \sum_{t \in E \cap I(T_i)} \beta_i^t \mid T_i \in \mathcal{T}_k\}$. All the machine types that become tight are considered allocated from now on. For each currently unscheduled job j whose interval is contained in the interval one of these newly allocated machine types, we allocate a separate machine of the appropriate type, say T_i , schedule j on it, and set $\alpha_j = \sum_{t \in E \cap I(j)} \beta_i^t$.

We claim that the dual solution thus constructed is feasible. Clearly, the algorithm satisfies all constraints of type (4) at all times. To see that the solution satisfies all constraints of type (5) as well, consider any such constraint $\alpha_j - \sum_{t \in E \cap I(j)} \beta_i^t$. Suppose job j was scheduled in the k th iteration. Following the k th iteration, α_j remains unchanged and the sum of β s can only increase, so it suffices to show that the constraint is satisfied at the end of the k th iteration. Let $T_{i'}$ be the type of the machine on which job j was scheduled. If $i = i'$, the constraint is satisfied by equality. Otherwise, machine type T_i could not have been allocated prior to the k th iteration (for otherwise job j would have already been scheduled by the time the k th iteration commenced), and thus, for all $t \in E \cap I(j)$, the values of β_i^t and $\beta_{i'}^t$ must have increased identically during the first k iterations. Thus, $\sum_{t \in E \cap I(j)} \beta_i^t = \sum_{t \in E \cap I(j)} \beta_{i'}^t$ at the end of the k th iteration, and the claim follows.

Phase 2: reverse reschedule & delete. Let $\mathcal{M}$ be the set of machines allocated in the first phase. Later on we describe a reverse reschedule & delete procedure that returns a feasible schedule using a subset $\mathcal{M}' \subseteq \mathcal{M}$ of machines which has the property that for all k , the number of machines in $\mathcal{M}'$ of types from $\mathcal{T}_k$ is at most $3n_k$. We note that in standard primal-dual algorithms the second phase is a simple *reverse delete* phase, whose purpose is to yield a minimal solution. The approximation guarantee then follows from an upper bound on minimal solutions. In our case, we do not know how to find a minimal solution. In fact, even determining whether all jobs can be scheduled on a given set of machines is NP-hard. We therefore do not attempt to find a minimal solution, but instead, gradually discard some of the machines in a very special manner designed to achieve the above property.

Analysis. Let (α, β) be the dual solution constructed in Phase 1 and let (x, y) be the primal solution corresponding to the schedule generated in Phase 2. To show that this schedule is 3-approximate, it suffices to prove that $\sum_{i=1}^m w(T_i)x_i \leq 3 \sum_{j \in J} \alpha_j$. This inequality from the next two claims.

$$\text{Claim.} \quad \sum_{i \in T} w(T_i)x_i \leq 3 \sum_k n_k \delta_k.$$

Proof. For each allocated machine type T_i , $w(T_i)$ equals the sum of δ_k taken over all k such that machine type T_i was unallocated at the beginning of the k th iteration and $t_k \in I(T_i)$. Thus, $\sum_{i \in T} w(T_i)x_i = \sum_k \delta_k m_k$, where m_k is the number of machines of types in $\mathcal{T}_k$ used by the final schedule. The claim then follows, since $m_k \leq 3n_k$. $\square$

$$\text{Claim.} \quad \sum_{j \in J} \alpha_j = \sum_k n_k \delta_k.$$

Proof. For each job j , α_j is the sum of all δ_k such that job j was still unscheduled at the beginning of the k th iteration and $t_k \in I(j)$. For each iteration k , the number of jobs that were unscheduled at the beginning of the k th iteration and contain t_k is exactly n_k . $\square$

The reverse reschedule & delete procedure. Let $\mathcal{M}$ be the set of machines used in the schedule constructed in the first phase, and let $\mathcal{M}_k \subseteq \mathcal{M}$ be the subset of machines of types in $\mathcal{T}_k$. The purpose of the reverse reschedule & delete procedure is to prune each machine set $\mathcal{M}_k$, leaving only $3n_k$ (or less) of its members allocated, yet manage to feasibly schedule all of the jobs on the surviving machines. To achieve this we consider the sets $\mathcal{M}_k$ in reverse order (decreasing k), and prune each in turn.

The pruning procedure for $\mathcal{M}_k$ is the following. If $|\mathcal{M}_k| \leq 3n_k$, we do nothing. Otherwise, consider the jobs currently assigned to machines in $\mathcal{M}_k$. They are of three possible types: *left jobs*, which lie entirely to the left of t_k ; *right jobs*, which lie entirely to the right of t_k ; and *middle jobs*, which cross t_k .

The middle jobs are easiest. The number of middle jobs is exactly n_k (by definition), so they are currently scheduled on n_k different machines. We retain these machines, denoting them $\mathcal{M}_{\text{mid}}$, and the scheduling of *all* jobs currently assigned to them (these may include some right jobs or left jobs in addition to all middle jobs).

The remaining left jobs are scheduled in the following manner. First note that $|\mathcal{M}_k \setminus \mathcal{M}_{\text{mid}}| \geq 2n_k$, since $|\mathcal{M}_k| > 3n_k$. Denote by $\mathcal{M}_{\text{left}}$ the set of n_k machines in $\mathcal{M}_k \setminus \mathcal{M}_{\text{mid}}$ with leftmost left endpoints. Observe that the intervals of these machines all contain t_k by definition (as do the intervals of all machines in $\mathcal{M}_k$). Let t be the rightmost endpoint among the left endpoints of machines in $\mathcal{M}_{\text{left}}$. All left jobs whose left endpoints are to the left of t must be currently scheduled on machines in $\mathcal{M}_{\text{left}}$, so we leave them intact. We proceed to reschedule all remaining left jobs greedily in order of increasing left endpoint. Specifically, for each job j we select any machine in $\mathcal{M}_{\text{left}}$ on which we have not already rescheduled a job that conflicts with j , and schedule j on it. To see that this is always possible, observe that all n_k machines are available between t and t_k , and thus if a job cannot be scheduled, then its left endpoint must be contained in n_k other jobs that were scheduled on machines in $\mathcal{M}_k$. These $n_k + 1$ jobs were therefore unscheduled at the beginning of the k th iteration (since we are pruning the sets $\mathcal{M}_k$ in reverse order), but this contradicts the definition of n_k , as these jobs all intersect at one time point.

The remaining right jobs are scheduled in a symmetric manner on the n_k machines in $\mathcal{M} \setminus \mathcal{M}_{\text{mid}}$ with rightmost right endpoints. Some (or all) of these machines may belong to $\mathcal{M}_{\text{left}}$, and therefore may already have left jobs scheduled on them, but that is not a problem because the intervals of left jobs and right jobs do not intersect.

2.2 The Unweighted Case

We present a 2-approximation algorithm for this case. Let L be the set of left endpoints of job intervals. For each point of time $t \in L$, let n_t be the number

of jobs whose intervals contain t . The algorithm consists of two stages. In the first stage it solves the optimization problem of allocating a minimum number of machines such that for all $t \in L$, at least n_t of the allocated machines are available at time t . In the second stage it schedules the jobs using at most twice the number of machines allocated in the first stage.

Stage 1. Scan the points of time in L in left-to-right order. For each point $t \in L$, Let n'_t be the number of machines that are available at time t and have already been allocated. If $n'_t < n_t$, allocate another $n_t - n'_t$ machines of type T_t , where T_t is the machine type with the rightmost right endpoint among all machines available at time t .

Proposition 1. *The solution found in Stage 1 is optimal.*

Proof. We say that a time point $t \in L$ is *covered* by a set of machines $\mathcal{M}$ if $\mathcal{M}$ contains at least n_t machines that are available at time t . Let t_i be the time point considered in the i th iteration of Stage 1, and let $\mathcal{M}_i$ be the set of machines allocated in the first i iterations of Stage 1. We prove by induction that for all i , there exists an optimal solution that contains $\mathcal{M}_i$. For $i = 0$, $\mathcal{M}_i = \emptyset$ and the claim holds trivially. Consider $i > 0$. By the induction hypothesis there exists an optimal solution $\mathcal{M}^*$ such that $\mathcal{M}_{i-1} \subseteq \mathcal{M}^*$. If no new machines are allocated in the i th iteration, then $\mathcal{M}_i = \mathcal{M}_{i-1} \subseteq \mathcal{M}^*$. Otherwise, there are at least $n_{t_i} - n'_{t_i}$ machines in $\mathcal{M}^* \setminus \mathcal{M}_{i-1}$ that are available at time t_i . We remove any $n_{t_i} - n'_{t_i}$ of them from $\mathcal{M}^*$ and replace them by the newly allocated machines $\mathcal{M}_i \setminus \mathcal{M}_{i-1}$. This cannot affect the feasibility because all time points $t_j < t_i$ remain covered by $\mathcal{M}_{i-1}$, and the choice of $\mathcal{M}_i \setminus \mathcal{M}_{i-1}$ as machines with the rightmost right endpoints that are available at time t_i guarantees that they are all available at all times $t_j \geq t_i$ at which any of the machines they replace are available. $\square$

Remark 1. A different approach to the solution of the optimization problem of Stage 1 is through the natural integer linear program for this problem. It is easy to see that the constraint matrix defining the linear program is totally unimodular (TUM) and thus the optimal solution to the linear program is always integral.

Stage 2. Let $\mathcal{M}$ be the set of machines allocated in Stage 1. Order the jobs by their left endpoint (from left to right) and schedule them in this order on machines in $\mathcal{M}$. Select for each job any machine on which no previously scheduled jobs intersect with the present job. The machine selected must also satisfy the condition that its time interval contains the job's left endpoint (though not necessarily the job's entire interval). The resultant schedule might, of course, be infeasible, due to jobs extending beyond the right endpoints of the machines on which they are scheduled, but at most one job per machine may do so. Fix the schedule by allocating new machines, one for each of these jobs. At most $|\mathcal{M}|$ new machines are added to the schedule.

Theorem 1. *Stage 2 returns a 2-approximate solution.*

Proof. The initial (infeasible) schedule constructed in Stage 2 contains all jobs, for if Stage 2 cannot schedule some job, then there are at least k other jobs containing its left endpoint $t_i \in L$, where k is the number of machines in $\mathcal{M}$ available at time t_i . This implies $n_{t_i} > k$, contradicting the fact that at each point of time $t \in L$, at least n_t machines are allocated in Stage 1. Thus, the final schedule constructed in Stage 2 is feasible and it uses at most $2|\mathcal{M}|$ machines. Since $|\mathcal{M}|$ is clearly a lower bound on the optimum, the solution is 2-approximate. $\square$

3 Cost Minimization with Machine Strengths

In this section we present a 2-approximation algorithm for the special case of the cost minimization problem where there is a linear order of *strength* on the machine types $T_1 \prec T_2 \prec \dots \prec T_m$, such that a job that may be processed by a machine of a given type may also be processed on every stronger machine. In other words, $S(j)$ has the form $\{T_{i_j}, T_{i_j+1}, \dots, T_m\}$ for all $j \in J$. We also assume that the stronger a machine, the higher its cost, i.e., $w(T_i) < w(T_{i+1})$ (otherwise there is no point in ever using weaker machines). We say that job j exists at time t if $t \in I(j)$.

For $1 \leq i \leq m$, let n_i be the maximum cardinality of a set of jobs that all exist simultaneously at some time point and all require machines of type T_i or stronger. Clearly, every feasible schedule requires at least n_i machines of type T_i or stronger, for all i . Thus, the cost of an optimal schedule is at least as high as the minimum cost of a set of machines with the property that for all i , the set contains at least n_i machines of type T_i or stronger. Define $n_{m+1} = 0$. Consider a set of machines $\mathcal{M}$ consisting of $n_i - n_{i+1}$ machines of type T_i , for all $1 \leq i \leq m$ (note that $n_i \geq n_{i+1}$ for all i , and the number of machines allocated in $\mathcal{M}$ of type T_i or stronger is n_i). Then $\mathcal{M}$ has the above property, and because stronger machines cost more than weaker ones, $\mathcal{M}$ is a minimum cost set with this property. Thus the cost of $\mathcal{M}$ is a lower bound on the cost of optimal solution. We show how to schedule all jobs on a set of machines containing at most two copies of each machine in $\mathcal{M}$. This schedule is therefore 2-approximate.

Let $M_1, \dots, M_k$ (where $k = n_1$) be the machines in $\mathcal{M}$ ordered from weakest to strongest. Construct an *initial* infeasible schedule as follows. Consider the machines in order from M_1 to M_k . For each M_i , construct a schedule containing a subset of the jobs as follows. First, schedule on M_i all of the currently unscheduled jobs that can be processed on it. Ignore job overlap when constructing this schedule. Then, iterate: as long as there is a job j scheduled on M_i that is fully contained in the union of other jobs scheduled on M_i , un-schedule job j . Although the schedule thus constructed for M_i may contain overlapping jobs, it has the redeeming property that the interval graph it induces is 2-colorable, as it is an easy fact that if three intervals intersect, at least one of them must be contained in the union of the other two. Having constructed the initial schedule (on all machines), color the induced interval graph on each machine with two colors and create a feasible schedule by using two copies of $\mathcal{M}$, one for each color class.

It remains to show that the initial schedule contains all jobs. Restricting our attention to this schedule, we say that a time point t is *covered* on machine M_i if there is job containing t scheduled on M_i . By construction, the set of points covered on M_i is precisely the union of all jobs that could be processed on M_i and were still unscheduled when the algorithm reached M_i . It follows that if a time point t is not covered on M_i , then every job that contains it either cannot be processed on M_i , or is scheduled on some machine $M_{i'}$, $i' < i$. Thus, suppose the algorithm fails to schedule some job j . Let t be any time point in j . Then t must be covered on the strongest machine, i.e., M_k , since it is contained in an unscheduled job (namely j) that can be processed on it. Let i be minimal such that t is covered on machines $M_i, M_{i+1}, \dots, M_k$. Let J' be the set of jobs scheduled on these machines that contain t . Assuming $i > 1$, point t is not covered on M_{i-1} by definition. Thus, by our previous observation, all of the jobs in $J' \cup \{j\}$ cannot be processed on M_{i-1} , and one (or two) of them are scheduled on M_i , so M_i is strictly stronger than M_{i-1} . Let T_{i-1} be the type of machine M_{i-1} . Then, all the jobs from $J' \cup \{j\}$ require machines of type at least as strong as T_i . Thus, $|J' \cup \{j\}| \leq n_i$. On the other hand, the number of available machines of types T_i or stronger is $< |J' \cup \{j\}| \leq n_i$, in contradiction with the fact the $\mathcal{M}$ contains n_i such machines. In the case $i = 1$ we get a contradiction directly: $n_1 \geq |J' \cup \{j\}| \geq k + 1 > n_1$.

Theorem 2. *Our algorithm returns a 2-approximate solution.*

4 Revenue Maximization

In the revenue maximization problem, we are given a set of machines $\mathcal{M} = \{M_1, \dots, M_m\}$ (presumably, already paid for) and a set of jobs J . Since the set of machines is fixed here, we identify machines with machine types. For each job $j \in J$, there is a time interval $I(j)$ during which it should be processed and a non-negative profit (or *weight*) $w(j)$ associated with it. Every job j specifies an arbitrary set, $S(j) \subseteq \mathcal{M}$, of machines on which it can be scheduled. The goal is to find a feasible schedule of a subset of the jobs on the machines that maximizes the total profit of the jobs scheduled. We present a $(1 - 1/e)$ -approximation algorithm for this problem. Our approach is to cast the problem as an integer problem and solve its linear programming (LP) relaxation. We then obtain an integral solution by randomly rounding the optimal fractional solution found for the LP relaxation.

The Linear Program: For each job j and for each machine M_i , there is a variable x_{ij} that indicates whether job j is scheduled on machine M_i .

$$\begin{aligned} \text{Max} \quad & \sum_{j \in J} \sum_{M_i \in S(j)} w(j)x_{ij} \quad \text{s.t.} \\ \sum_{i=1}^m x_{ij} & \leq 1, \quad \forall j \in J; \end{aligned} \tag{7}$$

$$\sum_{j: t \in I(j)} x_{ij} \leq 1, \quad \forall i \in \{1, \dots, m\}, \forall t; \tag{8}$$

$$x \geq 0. \tag{9}$$

Constraints (7) guarantee that each job is scheduled at most once. Constraints (8) guarantee that each machine executes at most one job at each time point.

Randomized Rounding: Let x be an optimal fractional solution. Choose N to be the smallest integer such that $N \cdot x_{ij}$ is integral for all i, j . We perform the randomized rounding on each machine separately. For each machine M_i , perform the following steps:

1. Construct an interval graph $\mathcal{I}$ as follows. For each job j , add $N \cdot x_{ij}$ copies of the time interval $I(j)$ to $\mathcal{I}$. Note that at each time point, the sum of the fractions of the jobs that are executed on machine M_i is at most 1. Thus the size of the maximum clique in the interval graph $\mathcal{I}$ is at most N .
2. Color $\mathcal{I}$ with N colors. Each color class induces a feasible schedule on machine M_i .
3. Choose one of the color classes uniformly at random. Schedule on M_i all the jobs that have time intervals belonging to this color class. If a job is scheduled on more than one machine, arbitrarily unassign it from all but one machine.

We remark that there is no need to build the interval graph $\mathcal{I}$ explicitly, i.e., to replicate intervals. In fact, a coloring satisfying the above can be computed in strongly polynomial time.

We now estimate the expected revenue of the schedule thus generated. For each job j , let $x_j = \sum_{i=1}^m x_{ij}$. For a job j , the probability of its being scheduled on a particular machine M_i is exactly x_{ij} . Therefore, the probability that it is not assigned to M_i is $1 - x_{ij}$. Thus, the probability that it is not assigned to any machine is

$$\prod_{i=1}^m (1 - x_{ij}) \leq \prod_{i=1}^m \left(1 - \frac{x_j}{m}\right) = \left(1 - \frac{x_j}{m}\right)^m < e^{-x_j}.$$

The probability that job j appears in the final schedule is therefore not less than $1 - e^{-x_j} \geq (1 - 1/e)x_j$, where the inequality follows from the fact that the real function $1 - e^{-x} - (1 - 1/e)x$ is non-negative in the range $0 \leq x \leq 1$ (as can be seen easily by differentiation). Thus the expected revenue is at least $(1 - 1/e) \sum_j w(j)x_j$. Using standard techniques, we can derandomize our algorithm without decreasing the approximation factor. The next theorem follows from the discussion above.

Theorem 3. *The algorithm yields a $(1 - 1/e)$ -approximate solution.*

Remark 2. A similar algorithm can be used to obtain a $(1 - 1/e)$ -approximation for the more general problem where each job j has a release date r_j , a deadline d_j and a processing time p_j , and $d_j - r_j < 2p_j$ for all j .

Acknowledgment. We thank Frits Spieksma for pointing out reference [13].

References

1. E. M. ARKIN AND E. B. SILVERBERG, Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, Vol. 18, pp. 1–8, 1987.
2. A. BAR-NOY, S. GUHA, J. NAOR, AND B. SCHIEBER, Approximating the throughput of multiple machines in real-time scheduling, *SIAM Journal on Computing*, Vol. 31 (2001), pp. 331–352.
3. R. BAR-YEHUDA, A. BAR-NOY, A. FREUND, J. NAOR, AND B. SCHIEBER, A unified approach To approximating resource allocation and scheduling, *Proc. 32nd Annual ACM Symposium on Theory of Computing*, pp. 735–744, 2000.
4. P. BERMAN AND B. DASGUPTA, Multi-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization*, Vol. 4, pp. 307–323, 2000.
5. R. BAR-YEHUDA AND S. EVEN, A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, Vol. 2, pp. 198–203, 1981.
6. G. CHELIOTIS, Bandwidth Trading in the Real World: Findings and Implications for Commodities Brokerage. *3rd Berlin Internet Economics Workshop*, 26–27 May 2000, Berlin.
7. S. CHIU AND J. P. CRAMETZ, Surprising pricing relationships. *Bandwidth Special Report, RISK. ENERGY & POWER RISK MANAGEMENT* pages 12–14, July 2000. (<http://www.riskwaters.com/bandwidth>)
8. J. CHUZHUY, R. OSTROVSKY, Y. RABANI, Approximation algorithms for the job interval selection problem and related scheduling problems *Proc. 42nd Annual Symposium of Foundations of Computer Science*, pp. 348–356, 2001.
9. M.R. GAREY, D.S. JOHNSON, G.L. MILLER AND C.H. PAPADIMITRIOU, The complexity of coloring circular arcs and chords. *SIAM Journal on Algebraic and Discrete Methods*, Vol. 1, pp. 216–227, 1980.
10. M. X. GOEMANS AND D. P. WILLIAMSON, A general approximation technique for constrained forest problems. *SIAM J. on Computing*, Vol. 24, pp. 296–317, 1995.
11. K. JANSEN An approximation algorithm for the license and shift class design problem *European Journal of Operational Research* 73 pp. 127–131, 1994.
12. C. KENYON AND G. CHELIOTIS, Stochastic Models for Telecom Commoditiy Prices *Computer Networks* 36(5–6):533–555, Theme Issue on Network Economics, Elsevier Science, 2001.
13. A. W. J. KOLEN AND L. G. KROON, On the computational complexity of (maximum) class scheduling, *European Journal of Operational Research*, Vol. 54, pp. 23–38, 1991.
14. A. W. J. KOLEN AND L. G. KROON, On the computational complexity of (maximum) shift scheduling, *European Journal of Operational Research*, Vol. 64, pp. 138–151, 1993.
15. A. W. J. KOLEN AND L. G. KROON, An analysis of shift class design problems, *European Journal of Operational Research*, Vol. 79, pp. 417–430, 1994.
16. A. W. J. KOLEN AND J. K. LENSTRA, Combinatorics in operations research, *Handbook of Combinatorics*, Eds: R. L. Graham, M. Grotschel, and L. Lovasz, North Holland, 1995.
17. F. C. R. SPIEKSMAN. On the approximability of an interval scheduling problem. *Journal of Scheduling*, Vol. 2 pp. 215–227, 1999.

18. V. V. VAZIRANI. Approximation Algorithms. Springer-Verlag, 2001.
19. <http://www.zurich.ibm.com/bandwidth/concepts.html>
20. P. WINKLER AND L. ZHANG, Wavelength Assignment and Generalized Interval Graph Coloring, *Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003.

CTL⁺ Is Complete for Double Exponential Time

Jan Johannsen and Martin Lange

Institut für Informatik
Ludwig-Maximilians-Universität München
Munich, Germany
{jjohanns,mlange}@informatik.uni-muenchen.de

Abstract. We show that the satisfiability problem for CTL⁺, the branching time logic that allows boolean combinations of path formulas inside a path quantifier but no nesting of them, is 2-EXPTIME-hard. The construction is inspired by Vardi and Stockmeyer's 2-EXPTIME-hardness proof of CTL^{*}'s satisfiability problem. As a consequence, there is no subexponential reduction from CTL⁺ to CTL which preserves satisfiability.

1 Introduction

In the early 80s, a family of branching time logics was defined by Emerson and Halpern [3,4]. This included the commonly known logics CTL and CTL^{*} as well as the less known logic CTL⁺.

CTL formulas can only speak about states of a transition system, while CTL^{*} allows properties of paths and states to be expressed. CTL⁺ is the fragment of CTL^{*} which does not allow temporal operators to be nested. It subsumes CTL syntactically.

Emerson and Halpern [3] already showed that every CTL⁺ formula is equivalent to a CTL formula. The translation, however, yields formulas of exponential length. Recently, Wilke [10] and Adler and Immerman [1] have shown that this is unavoidable, i.e. that there are CTL⁺ formulas of size n such that every equivalent CTL formula is of size $\Omega(n!)$.

This gap becomes apparent for example when the complexity of the model checking problem for these logics is considered. For CTL the problem is PTIME-complete, even in linear time, while the CTL⁺ model checking problem is Δ_2 -complete in the polynomial time hierarchy [8].

Kupferman and Grumberg [7] have shown that one can relax the syntactic restrictions CTL imposes on branching time formulas without having to give up linear time model checking. They define a logic CTL², which allows two temporal operators in the scope of a path quantifier - either nested or a boolean combination thereof. Syntactically, CTL⁺ and CTL² are incomparable although semantically CTL² strictly subsumes CTL and therefore CTL⁺ as well. To the best of our knowledge, no complexity bounds on CTL²'s satisfiability problem are given.

In contrast, CTL^{*} which is known to be strictly more expressive than CTL, CTL⁺ and even CTL², has a PSPACE-complete model checking problem [6].

Concerning the satisfiability checking problem, CTL is EXPTIME-complete while CTL^* is 2-EXPTIME-complete. Inclusion in 2-EXPTIME was proved by Emerson and Jutla [5] after it had been shown to be contained in various deterministic and nondeterministic complexity classes between 2-EXPTIME and 4-EXPTIME. 2-EXPTIME-hardness was shown by Vardi and Stockmeyer [9] using a reduction from the word problem for an alternating exponential space bounded Turing Machine.

We use the basic ideas of their construction in order to prove 2-EXPTIME-hardness of CTL^+ 's satisfiability checking problem. For instance, we also encode the computation tree of an alternating exponential space bounded Turing Machine on an input word by a tree model for a CTL^+ formula that describes the machine's behaviour. However, in order to overcome CTL^+ 's weaknesses in expressivity compared to CTL^* we need to make amendments to the models and the resulting formulas. Note that CTL^+ is, for example, not able to speak about the penultimate state on a finite path which is a crucial point in Vardi and Stockmeyer's reduction.

To overcome this problem we use a special type of alternating Turing Machine which is easily seen to be equivalent to a common one in terms of space complexity. This Turing Machine has states of three different types: those in which the tape head is deterministically moved, as well as existentially and universally branching states in which the symbol under the tape head is replaced and no movement takes place.

For this sort of alternating Turing Machine it becomes possible to describe the machine's behaviour by a CTL^+ formula. The distinction of Turing Machine states does not require formulas that speak about more than two consecutive states on a path of a transition system.

There are other CTL^* formulas in Vardi and Stockmeyer's paper which cannot easily be transformed into CTL^+ because of CTL^+ 's restriction regarding the nesting of path operators. E.g. the natural way of expressing that some event E happens at most once along a path uses two nested *until* formulas ("it is not the case that E happens at some point and at another point later on"). Formulas of this kind occur in properties like "there is exactly one tape head per configuration". To make the reduction work for CTL^+ too, we use additional atomic propositions in a model for the resulting CTL^+ formula.

Completeness follows from the fact that the satisfiability checking problem for CTL^* is in 2-EXPTIME, but also because CTL^+ can be translated into CTL at the cost of an exponential blow-up. This does not only – to the best of our knowledge – provide the first complexity-theoretical completeness result for the CTL^+ satisfiability problem. It also shows the curious fact that concerning expressiveness CTL and CTL^+ fall into the same class different from CTL^* . Concerning the model checking problem the three logics were shown to be complete for three (probably) different classes. But regarding satisfiability, CTL^+ and CTL^* are complete for the same class which is different from the complexity of CTL satisfiability.

Finally, we present a consequence of CTL^+ 's 2-EXPTIME-hardness. Wilke was the first to prove an exponential lower bound on the size of CTL formulas that arise under an equivalence preserving translation from CTL^+ [10]. This

was improved by Adler and Immerman, who showed that there is indeed an $n!$ lower bound [1]. The 2-EXPTIME-hardness of the CTL⁺ satisfiability problem strengthens Wilke's result in a different way: there is no subexponential reduction from CTL⁺ to CTL that preserves satisfiability.

2 Preliminaries

The logic CTL⁺. Let $\mathcal{P}$ be a finite set of propositional constants including **tt** and **ff**. A labelled transition system is a triple $\mathcal{T} = (\mathcal{S}, \rightarrow, L)$ s.t. $(\mathcal{S}, \rightarrow)$ is a directed graph, and $L : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ labels the elements of $\mathcal{S}$, called *states*, with **tt** $\in L(s)$, **ff** $\notin L(s)$ for all $s \in \mathcal{S}$. $\mathcal{T}$ is called *total* if for all $s \in \mathcal{S}$ there is an $s' \in \mathcal{S}$ s.t. $s \rightarrow s'$.

A path in a total transition system $\mathcal{T}$ is an infinite sequence $\pi = s_0 s_1 \dots$ of states s.t. $s_i \rightarrow s_{i+1}$ for all $i \in \mathbb{N}$. With π^i we denote the suffix of π starting with the i -th state.

Formulas of CTL⁺ are given by the following grammar.

$$\begin{aligned} \varphi &::= q \mid \varphi \vee \varphi \mid \neg \varphi \mid \mathbf{E}\psi \\ \psi &::= q \mid \psi \vee \psi \mid \neg \psi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \end{aligned}$$

where q ranges over $\mathcal{P}$. The φ are often called *state formulas* while the ψ are *path formulas*. Only state formulas are CTL⁺ formulas. Path formulas can only occur as subformulas of these.

We will use the standard abbreviations $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi := \neg\varphi \vee \psi$, $\mathbf{A}\varphi := \neg\mathbf{E}\neg\varphi$, $\mathbf{F}\varphi := \mathbf{ttU}\varphi$ and $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$. Furthermore, we will use a special until formula $\mathbf{F}_{\psi}\varphi := \neg\psi\mathbf{U}(\psi \wedge \varphi)$ which says that eventually φ holds in the first moment when ψ holds, too.

Formulas of CTL⁺ are interpreted over paths $\pi = s_0 s_1 \dots$ of a total transition system $\mathcal{T} = (\mathcal{S}, \rightarrow, L)$.

$$\begin{aligned} \pi \models q &\quad \text{iff } q \in L(s_0) \\ \pi \models \varphi \vee \psi &\quad \text{iff } \pi \models \varphi \text{ or } \pi \models \psi \\ \pi \models \neg \varphi &\quad \text{iff } \pi \not\models \varphi \\ \pi \models \mathbf{E}\varphi &\quad \text{iff } \exists \pi', \text{ s.t. } \pi' = s_0 \dots \text{ and } \pi' \models \varphi \\ \pi \models \mathbf{X}\varphi &\quad \text{iff } \pi^1 \models \varphi \\ \pi \models \varphi \mathbf{U}\psi &\quad \text{iff } \exists k \in \mathbb{N} \text{ s.t. } \pi^k \models \psi \text{ and } \forall i < k : \pi^i \models \varphi \end{aligned}$$

Since the truth value of a state formula φ in a path $\pi = s_0 s_1 \dots$ only depends on s_0 , it is possible to write $s \models \varphi$ for a state s of a transition system and such a formula φ . A state formula φ is called *satisfiable* if there is a transition system $\mathcal{T}$ with a state s , s.t. $s \models \varphi$.

Alternating Turing Machines. We use the following model of alternating Turing Machine, which differs slightly from the standard model [2], but is easily seen to be equivalent w.r.t. space complexity. An alternating Turing Machine $\mathcal{M}$ is of the form $\mathcal{M} = (Q, \Sigma, q_0, q_a, q_r, \delta)$, where Q is the set of states, Σ is the alphabet, which contains a blank symbol $\square \in \Sigma$, and $q_0, q_a, q_r \in Q$.

The set Q of states is partitioned into $Q = Q_{\exists} \cup Q_{\forall} \cup Q_m \cup \{q_a, q_r\}$, where we write Q_b for $Q_{\exists} \cup Q_{\forall}$, these are the *branching* states. The transition relation δ is of the form

$$\delta \subseteq (Q_b \times \Sigma \times Q \times \Sigma) \cup (Q_m \times \Sigma \times Q \times \{L, R\}) .$$

In a branching state $q \in Q_b$, the machine can act nondeterministically and writes on the tape, i.e., for each $a \in \Sigma$, there can be several transitions $(q, a, q', b) \in \delta$ for $q' \in Q$ and $b \in \Sigma$, meaning that the machine overwrites the a in the current tape cell with b , the machine enters state q' , and the head does not move.

In a state $q \in Q_m$, the machine acts deterministically and moves its head, i.e., for each $a \in \Sigma$, there is exactly one transition $(q, a, q', D) \in \delta$, for $q' \in Q$ and $D \in \{L, R\}$, meaning that the head moves to the left (L) or right (R), and the machine enters state q' . For $q \in \{q_a, q_r\}$, there are no transitions in δ , and the machine halts.

We assume that the machine only halts when the state is q_a or q_r . A halting configuration is accepting iff the state is q_a . For the other configurations, the acceptance behaviour depends on the kind of state:

If the state is in Q_m , then the configuration is accepting iff its unique successor is accepting. If the state is in $Q_{\exists}$, then the configuration is accepting iff at least one of its successors is accepting. If the state is in $Q_{\forall}$, then the configuration is accepting iff all of its successors are accepting. The whole computation accepts if the initial configuration is accepting.

Double exponential time. The complexity class of double exponential time is defined as

$$2\text{-EXPTIME} = \bigcup_{k \in \mathbb{N}} \text{DTIME}(2^{2^{k \cdot n}})$$

where $\text{DTIME}(f(n))$ is the class of all languages which are accepted by a deterministic Turing Machine in time $f(n)$ where n is the length of the input word at hand.

It is well-known [2] that 2-EXPTIME coincides with

$$\text{AEXPSPACE} = \bigcup_{k \in \mathbb{N}} \text{ASPACE}(2^{k \cdot n})$$

the class of all languages accepted by an alternating Turing Machine using space which is at most exponential in the size of the input word.

3 The Reduction

Theorem 1. *Satisfiability of CTL^+ is 2-EXPTIME-hard.*

Proof. Suppose $\mathcal{M} = (Q, \Sigma, q_0, q_a, q_r, \delta)$ is an alternating exponential space bounded Turing Machine. Let $w = a_0 \dots a_{n-1} \in \Sigma^*$ be an input for $\mathcal{M}$. W.l.o.g. we assume the space needed by $\mathcal{M}$ on input w to be bounded by $2^{kn} - 1$ for some $k \geq 1$. Let $N := 2^{kn} - 1$. Furthermore we assume that every computation ends

in a configuration with the head on the rightmost tape cell while the machine is in either of the states q_a or q_r .

In the following we will construct a CTL⁺ formula $\varphi_{\mathcal{M},w}$ s.t. $w \in L(\mathcal{M})$ iff $\varphi_{\mathcal{M},w}$ is satisfiable. Informally, an accepting computation of $\mathcal{M}$ on w will serve as a model for $\varphi_{\mathcal{M},w}$.

Like Vardi and Stockmeyer [9], we encode a configuration of $\mathcal{M}$ as a sequence of $2^{k \cdot n} - 1$ states in a possible model for $\varphi_{\mathcal{M},w}$. Successive configurations of the Turing Machine are modelled by concatenating these sequences, where we add one dummy state with index 0 between each pair of adjacent configurations.

The underlying set of propositions is $\mathcal{P} = Q \cup \Sigma \cup \{c_0, \dots, c_{k \cdot n - 1}\} \cup \{x, z, e\}$.

- $q \in Q$ is true in a state of the model iff the head of the Turing Machine is on the corresponding tape cell in the corresponding configuration while the machine is in state q . The formula $h := \bigvee_{q \in Q} q$ says that the machine is in some state, i.e. the head is on that cell.
- $a \in \Sigma$ is true iff a is the symbol on the corresponding tape cell.
- $c_{k \cdot n - 1}, \dots, c_0$ represent a counter in binary representation. The counter value in a state of the model is 0 at the dummy states and the number of the corresponding tape cell otherwise.
- x is used to denote that the corresponding configuration is accepting.
- z is used to mark the part of a tree model which corresponds to the computation. In order to be able to speak about a certain state somewhere on a path we let every state of the encoding have a successor which carries exactly the same amount of information except that it is labelled with $\neg z$. Thus, such a state can be seen as not belonging directly to the encoding of the computation tree but being a clone of a state in this tree.
- e indicates that the state at hand belongs to an “even” configuration, i.e. one with an even index in a sequence $C_0, C_1, \dots$ of configurations of the computation.

For every fixed m we can write a formula χ_m which says that the counter value is m in the current state, e.g.

$$\chi_0 := \bigwedge_{i=0}^{k \cdot n - 1} \neg c_i, \quad \chi_1 := c_0 \wedge \bigwedge_{i=1}^{k \cdot n - 1} \neg c_i \quad \text{and} \quad \chi_N := \bigwedge_{i=0}^{k \cdot n - 1} c_i$$

for the dummy ($m = 0$), the leftmost ($m = 1$) and rightmost ($m = N$) position in a configuration.

In order to describe $\mathcal{M}$'s behaviour on w we need to express several properties. The formula φ_0 says that there is always exactly one symbol on a tape cell, and $\mathcal{M}$ is never in two different states at the same time.

$$\begin{aligned} \varphi_0 := \mathbf{AG} (& (\neg \chi_0 \rightarrow \bigvee_{a \in \Sigma} a) \wedge (\chi_0 \rightarrow \neg h \wedge \bigwedge_{a \in \Sigma} \neg a) \wedge \\ & \bigwedge_{a, b \in \Sigma, b \neq a} \neg(a \wedge b) \wedge \bigwedge_{q, q' \in Q, q \neq q'} \neg(q \wedge q')) \end{aligned}$$

We can say that the counter value is not changed in the transition to the next state on a given path. This is used to clone states as indicated above. The value of e does not change in this case.

$$\psi_{rem} := (e \leftrightarrow Xe) \wedge \bigwedge_{j=0}^{k \cdot n - 1} (c_j \leftrightarrow Xc_j)$$

We can also say that the counter value is increased by 1 modulo $2^{k \cdot n}$. Then, a switch from e to $\neg e$ or vice versa occurs iff the counter is increased from $2^{k \cdot n} - 1$ to 0.

$$\begin{aligned} \psi_{inc} := & ((e \leftrightarrow X\neg e) \wedge \chi_N \wedge X\chi_0) \vee \\ & (e \leftrightarrow Xe) \wedge \bigvee_{j=0}^{k \cdot n - 1} (\neg c_j \wedge Xc_j \wedge \bigwedge_{i>j} (c_i \leftrightarrow Xc_i) \wedge \bigwedge_{i<j} (c_i \wedge X\neg c_i)) \end{aligned}$$

The entire computation of $\mathcal{M}$ forms a tree. Each state is labelled with a symbol of Σ . Moreover, z holds on every state on the computation, and every state has at least one successor from which on z never holds. Furthermore, the subtree under this state reflects the labelling of its root's predecessor which still satisfies z . This idea is taken from Vardi and Stockmeyer's proof [9] and used to be able to speak about finite prefixes of infinite paths.

On all paths q_a or q_r is eventually reached and all following states do not satisfy z . The counter is only increased (modulo $2^{k \cdot n}$) in states satisfying z .

$$\begin{aligned} \psi_{eq} &:= \psi_{rem} \wedge \bigwedge_{q \in Q} q \leftrightarrow Xq \wedge \bigwedge_{a \in \Sigma} a \leftrightarrow Xa \\ \varphi_1 &:= \mathbf{AF} \neg z \wedge \\ &\quad \mathbf{AG} ((z \wedge \neg q_a \wedge \neg q_r) \rightarrow (\mathbf{EX} z \wedge \mathbf{EX} \neg z) \wedge \\ &\quad \neg z \rightarrow \mathbf{A} (X\neg z \wedge \psi_{eq}) \wedge \\ &\quad (q_a \vee q_r) \rightarrow \mathbf{AX} \neg z \wedge \chi_N) \wedge \\ &\quad \mathbf{AGA} ((z \wedge Xz \leftrightarrow \psi_{inc}) \wedge (z \wedge X\neg z \leftrightarrow \psi_{eq})) \end{aligned}$$

There is at most one tape head in every configuration. (The fact that there is at least one will be guaranteed by φ_5 later on.) This is achieved by saying that there is no bit c_i which distinguishes two possible occurrences of an h in one configuration. To guarantee that one speaks about the same configuration for two such occurrences of h , we demand that the value of e never changes in between.

$$\begin{aligned} \varphi_2 := & \mathbf{AGA} (\chi_0 \rightarrow (e \rightarrow \neg (\bigvee_{i=0}^{k \cdot n - 1} eU(e \wedge h \wedge c_i) \wedge eU(e \wedge h \wedge \neg c_i)) \wedge \\ & \neg e \rightarrow \neg (\bigvee_{i=0}^{k \cdot n - 1} \neg eU(\neg e \wedge h \wedge c_i) \wedge \neg eU(\neg e \wedge h \wedge \neg c_i)))) \end{aligned}$$

The computation is accepting. Every q_a is marked with an x but no q_r is. Moreover, an x occurs together with an existential state only if there is a path along

z s.t. x holds together with the first occurrence of h . For universal or moving states all z -paths must satisfy x in their first occurrence of h .

$$\begin{aligned} \varphi_3 := & x \wedge \mathbf{AG}((q_a \rightarrow x) \wedge (q_r \rightarrow \neg x) \wedge \\ & \bigwedge_{q \in Q_\exists} q \rightarrow (x \leftrightarrow \mathbf{EX}((z \wedge \neg h)\mathbf{U}(z \wedge h \wedge x))) \wedge \\ & \bigwedge_{q \in Q_\forall \cup Q_m} q \rightarrow (x \leftrightarrow \mathbf{AXA}(z\mathbf{U}(z \wedge h) \rightarrow \mathbf{F}_h x))) \end{aligned}$$

At the beginning, the tape contains $a_1 \dots a_n \square \dots \square$, the input word followed by $2^{k \cdot n} - n$ blank symbols. $\mathcal{M}$ is in state q_0 and the head is on the first symbol of w .

$$\begin{aligned} \varphi_4 := & z \wedge e \wedge \chi_0 \wedge \\ & \mathbf{EX}(z \wedge q_0 \wedge a_1 \wedge \\ & \mathbf{EX}(z \wedge a_2 \wedge \\ & \dots \wedge \\ & \mathbf{EX}(z \wedge a_n \wedge \\ & \mathbf{EXE}(z \wedge \square)\mathbf{U}(z \wedge \chi_0))) \dots) \end{aligned}$$

Now we have to say that two adjacent configurations comply with $\mathcal{M}$'s transition rules. In order to do so we need the following statements about a path. The counter value is 0 exactly once before $\neg z$ holds.

$$\begin{aligned} \psi_1 := & e \rightarrow z\mathbf{U}(z \wedge \neg e \wedge \chi_0) \wedge \neg e \rightarrow z\mathbf{U}(z \wedge e \wedge \chi_0) \wedge \\ & \neg(z\mathbf{U}(e \wedge \chi_0) \wedge z\mathbf{U}(\neg e \wedge \chi_0)) \end{aligned}$$

We need three formulas saying that the counter value in the first state not satisfying z is the same as the value of the first state on the path, resp. increased or decreased by 1. We explicitly forbid to increase a maximal value, resp. decrease a minimal one, i.e. do not calculate modulo $2^{k \cdot n}$, because these formulas are used to describe the tape head's moves. Note that it cannot go left at the right end of the tape and vice-versa.

$$\begin{aligned} \psi_0 := & \bigwedge_{i=0}^{k \cdot n - 1} c_i \leftrightarrow \mathbf{F}_{\neg z} c_i \\ \psi_{+1} := & \neg \chi_N \wedge \bigvee_{j=0}^{k \cdot n - 1} (\neg c_j \wedge \mathbf{F}_{\neg z} c_j) \wedge \bigwedge_{i>j} (c_i \leftrightarrow \mathbf{F}_{\neg z} c_i) \wedge \bigwedge_{i<j} (c_i \wedge \mathbf{F}_{\neg z} c_i) \\ \psi_{-1} := & \neg \chi_1 \wedge \bigvee_{j=0}^{k \cdot n - 1} (c_j \wedge \mathbf{F}_{\neg z} \neg c_j) \wedge \bigwedge_{i>j} (c_i \leftrightarrow \mathbf{F}_{\neg z} c_i) \wedge \bigwedge_{i<j} (\neg c_i \wedge \mathbf{F}_{\neg z} c_i) \end{aligned}$$

Finally, we have to describe the machine's transition behaviour δ . On every state the following holds.

- If it is labelled with a $q \in Q_b$ then the actual symbol is replaced in every next configuration at the same position.

- If it is not labelled with a $q \in Q_b$, in particular no q at all, then the corresponding state of the next configuration carries the same symbol from Σ .
- If it is labelled with a $q \in Q_m$ then every next or previous state to the corresponding one in the next configuration is labelled with the machine state that is given by the transition relation.

Note that the second and third case do not exclude each other.

$$\begin{aligned}
\varphi_5 := & \text{AG} \left(\bigwedge_{q \in Q_b, a \in \Sigma} (q \wedge a \rightarrow \bigwedge_{(q,a,q',b) \in \delta} \text{E}(\psi_1 \wedge \psi_{\text{=}} \wedge \text{F}_{\neg z}(q' \wedge b)) \right) \wedge \\
& \text{A}(\psi_1 \wedge \psi_{\text{=}} \rightarrow \text{F}_{\neg z} \bigvee_{(q,a,q',b) \in \delta} (q' \wedge b)) \big) \\
& \wedge \bigwedge_{a \in \Sigma} \neg \left(\bigvee_{q \in Q_b} q \right) \wedge a \rightarrow \text{A}(\psi_1 \wedge \psi_{\text{=}} \rightarrow \text{F}_{\neg z} a) \big) \\
& \wedge \bigwedge_{(q,a,q',L) \in \delta} q \wedge a \rightarrow \text{A}(\psi_1 \wedge \psi_{-1} \rightarrow \text{F}_{\neg z} q') \big) \\
& \wedge \bigwedge_{(q,a,q',R) \in \delta} q \wedge a \rightarrow \text{A}(\psi_1 \wedge \psi_{+1} \rightarrow \text{F}_{\neg z} q') \big)
\end{aligned}$$

Altogether, the machine's behaviour is described by the formula

$$\varphi_{\mathcal{M},w} := \varphi_0 \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5$$

Then, the part of a model for $\varphi_{\mathcal{M},w}$ that is marked with z corresponds to a successful computation tree of $\mathcal{M}$ on w . Conversely, such a tree can easily be extended to a model for $\varphi_{\mathcal{M},w}$.

Thus, $\mathcal{M}$ accepts w iff there exists a successful computation tree for $\mathcal{M}$ on w iff there exists a model for $\varphi_{\mathcal{M},w}$ iff $\varphi_{\mathcal{M},w}$ is satisfiable.

Finally, the size of $\varphi_{\mathcal{M},w}$ is quadratic in $|\Sigma|$ and $|Q|$ and linear in $|w|$ and $|\delta|$. $\square$

Corollary 1. *There is no reduction $r : \text{CTL}^+ \rightarrow \text{CTL}$ s.t. for all $\varphi \in \text{CTL}^+$:*

- φ is satisfiable iff $r(\varphi)$ is satisfiable, and
- $|r(\varphi)| \leq f(|\varphi|)$ for some $f : \mathbb{N} \rightarrow \mathbb{N}$ with $f(n^2) = o(2^n)$.

Proof. Suppose there is a reduction from CTL^+ to CTL that preserves satisfiability and produces formulas of subexponential length $f(n)$. Then this reduction in conjunction with a satisfiability checker for CTL can be used to decide satisfiability of CTL^+ in asymptotically less time than $O(2^{f(n)})$. As a consequence of Theorem 1, every language in 2-EXPTIME can be decided in time $O(2^{f(n^2)})$ since it can be reduced to CTL^+ in quadratic time, and satisfiability for CTL can be decided in time $O(2^n)$. But according to the asymptotic restriction on f and the Time Hierarchy Theorem, there is a language in 2-EXPTIME which is not decidable in time $O(2^{f(n^2)})$. To see this note that

$$f(n^2) = o(2^n) \text{ iff } f(n^2) + \log f(n^2) = o(2^n) \text{ iff } 2^{f(n^2)} \cdot f(n^2) = o(2^{2^n})$$

$\square$

References

1. M. Adler and N. Immerman. An $n!$ lower bound on formula size. In *Proc. 16th Symp. on Logic in Computer Science, LICS'01*, pages 197–208, Boston, MA, USA, June 2001. IEEE Computer Society.
2. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, January 1981.
3. E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.
4. E. A. Emerson and J. Y. Halpern. “Sometimes” and “not never” revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, January 1986.
5. E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, February 2000.
6. E. A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987.
7. O. Kupferman and O. Grumberg. Buy one, get one free!!! *Journal of Logic and Computation*, 6(4):523–539, August 1996.
8. F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking CTL⁺ and FCTL is hard. In *Proc. 4th Conf. Foundations of Software Science and Computation Structures, FOSSACS'01*, volume 2030 of LNCS, pages 318–331, Genova, Italy, April 2001. Springer.
9. M. Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc. 17th Symp. on Theory of Computing, STOC'85*, pages 240–251, Baltimore, USA, May 1985. ACM.
10. T. Wilke. CTL⁺ is exponentially more succinct than CTL. In *Proc. 19th Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'99*, volume 1738 of LNCS, pages 110–121. Springer, 1999.

Hierarchical and Recursive State Machines with Context-Dependent Properties^{*}

Salvatore La Torre, Margherita Napoli, Mimmo Parente, and
Gennaro Parlato

Dipartimento di Informatica e Applicazioni
Università degli Studi di Salerno

Abstract. Hierarchical and recursive state machines are suitable abstract models for many software systems. In this paper we extend a model recently introduced in literature, by allowing atomic propositions to label all the kinds of vertices and not only basic nodes. We call the obtained models *context-dependent hierarchical/recursive state machines*. We study on such models cycle detection, reachability and LTL model-checking. Despite of a more succinct representation, we prove that LTL model-checking can be done in time linear in the size of the model and exponential in the size of the formula, as for standard LTL model-checking. Reachability and cycle detection become NP-complete, and if we place some restrictions on the representation of the target states, we can decide them in time linear in the size of the formula and the size of the model.

Keywords: Model Checking, Automata, Temporal Logic.

1 Introduction

Due to their complexity, the verification of the *correctness* of many modern digital systems is infeasible without suitable automated techniques. Formal verification has been very successful and recent results have led to the implementation of powerful design tools (see [CK96]). In this area one of the most successful techniques has been *model checking* [CE81]: a high-level specification is expressed by a formula of a logic and this is checked for fulfillment on an abstract model (state machine) of the system. Though model checking is linear in the size of the model, it is computationally hard since the model generally grows exponentially with the number of variables used to describe a state of the system (*state-space explosion*). As a consequence, an important part of the research on model checking has been concerned with handling this problem.

Complex systems are usually composed of relatively simple modules in a hierarchical manner. Hierarchical structures are also typical of object-oriented

^{*} This research was partially supported by the MIUR in the framework of the project “Metodi Formali per la Sicurezza e il Tempo” (MEFISTO) and MIUR grant 60% 2002.

paradigms [BJR97,RBP⁺91,SGW94]. We consider systems modeled as *hierarchical finite state machines*, that is, finite state machines where a vertex can either expand to another hierarchical state machine or be a basic vertex (in the former case we call the vertex a *supernode*, in the latter simply a *node*).

The model we consider in this paper generalizes instead the model studied in [AY01]. There the authors consider the model checking on Hierarchical State Machines (HSM) where only the nodes are labeled with atomic propositions (*AP*). We relax this constraint and thus we allow to associate atomic propositions also with vertices that expand to a machine. Expanding a supernode v to a machine M , all vertices of M *inherit* the atomic propositions of v (*context*), so that different vertices expanding to M can place M into different contexts. For this reason, we call such a model a *hierarchical state machine with context-dependent properties* (in the following denoted by **Context-dependent Hierarchical State Machine**). The semantics of a **CHSM** is given by the corresponding natural flat model which is a Kripke structure.

By allowing this more general labeling, for a given system it is possible to obtain very succinct abstract models. In the following example, we show that the gain of succinctness can be exponential compared to the models used in [AY01]. Consider a digital clock with hours, minutes, and seconds. We can construct a hierarchical finite state machine $\mathcal{M}$ composed of three machines M_1 , M_2 , and M_3 such that the supernodes of M_3 expands to M_2 and the supernodes of M_2 expands to M_1 . Machine M_1 is a chain of nodes. Machines M_2 and M_3 are chains of supernodes except for the initial and the output vertices that are nodes. In M_3 each supernode corresponds to a hour and they are linked accordingly to increasing time. Analogously, M_2 models minutes and M_1 seconds. A flat model for the digital clock has at least $24 \cdot 60 \cdot 60 = 86,400$ vertices, while the above hierarchical model has only $24 + 60 + 60 + 6 = 150$ vertices (6 are simply initial and output nodes). Assume that we are interested in checking properties that refer to a precise time expressed in hours, minutes and seconds. Clearly, it is not sufficient to label only the nodes (we would be able to capture only that an event happens at a certain second, but we would have no clue of the actual hour and minute). In the model defined in [AY01], at least 86,400 nodes are needed, that is, there would be no gain with respect to a minimal flat model. In our model, we are able to label each supernode in M_3 with atomic propositions encoding the corresponding hour. Analogously we can use atomic propositions to encode minutes and seconds on M_2 and M_1 , respectively. This way, each state of the corresponding flat model is labeled with the encoding of a hour, a minute and a second in a day and vertices are linked by increasing time.

A simple way of analyzing hierarchical systems is first to flatten them into equivalent non-hierarchical systems and then apply existing verification techniques on finite state systems. The drawback of such an approach is that the size of the flat system can be exponential in the hierarchical depth. In many recent papers, it has been shown that it is possible to reduce the complexity growth caused by handling large systems, by performing verification in a hierarchical manner [AGM00,AG00,BLA⁺99,AY01]. We follow this approach and study on

CHSMs standard decision problems which are related to system verification, such as reachability, cycle detection, and model checking. In this paper, we also consider **Context-dependent Recursive State Machines (CRSM)** which generalize **CHSMs** by allowing recursive expansions and we study on them the verification-related problems listed above. Recursive generalizations of the hierarchical model presented in [AY01] are studied in [AEY01,BGR01]. Recursive machines can be used to model the control flow of programs with recursive calls and thus are suitable for abstracting the behavior of reactive software systems.

Results. Given a transition system, a state s and a set of *target states* T , (usually expressed by a propositional boolean formula), the *reachability problem* is the problem of determining whether a state of T can be reached from s on a run of the system. In practice, this problem is relevant in the verification of systems, for example it is related to the verification of *safety requirements*: we want to check whether all the reachable states of the system belong to a given “safe” region (*invariant checking problem*).

We prove that reachability on **CRSMs** is NP-complete, and NP-hardness still holds if we restrict to **CHSMs**. We then give an algorithm to decide reachability on **CRSMs** that runs in time linear in the size of the model and exponential in the size of the formula. Finally, given a **CHSM** $\mathcal{M}$, we show effective sufficient conditions for solving reachability in time linear in both the size of the formula and the size of the model. Let us remark that these conditions are satisfied when we consider an instance of the reachability problem where the model is given by a Hierarchical State Machine (HSM) as defined in [AY01].

The *cycle detection problem* is the problem of verifying whether a given state can be reached repeatedly. Cycle detection is the basic problem for the verification of *liveness properties*: “some good thing will eventually happen”.

We also consider the model checking of LTL formulas on **CRSMs**. Given a set of atomic propositions AP , a *linear temporal logic* (LTL) formula is built up in the usual way from atomic propositions, the boolean connectives, the temporal operators *next* and *until*. An LTL formula is interpreted over an infinite sequence over 2^{AP} . A **CRSM** satisfies a formula φ if every run in the corresponding flat model satisfies φ . Given an LTL formula φ and a **CRSM** $\mathcal{M}$, the *model checking problem* for $\mathcal{M}$ and φ is the problem to determine whether $\mathcal{M}$ satisfies φ . We give a decision algorithm that runs in $O(|\mathcal{M}| \cdot 8^{|\varphi|})$ time for **CHSMs** and an algorithm in $O(|\mathcal{M}| \cdot 16^{|\varphi|})$ time for **CRSMs**. Our algorithms do not need to flatten the system and mainly consist of reducing the model checking problem to the emptiness problem of recursive Büchi automata [AEY01].

The rest of the paper is organized as follows. In the next section definitions and notation are given. The NP-completeness of the cycle detection and of the reachability problems is shown in section 3 (actually the proofs for the cycle detection problems are omitted in this version, due to the lack of space). In section 4 we give the linear time algorithms for **CHSMs** and **CRSMs**. In Section 5, we discuss the model checking of LTL formulas. We conclude with few remarks in Section 6.

2 Context-Dependent State Machines

In this section we introduce the definitions and the notation we will use in the rest of the paper. We consider *Kripke structures*, that is, state-transition graphs where each state is labeled by a subset of a finite set of atomic propositions (AP). A *Context-dependent Recursive State Machine* (**CRSM**) over AP is a tuple $\mathcal{M} = (M_1, \dots, M_k)$ of Kripke structures with:

- a set of vertices N , split into disjoint sets $N_1, \dots, N_k$; a set $IN = \{in_1, \dots, in_k\}$ of initial vertices, where $in_i \in N_i$, and a set of output vertices OUT split into $OUT_1, \dots, OUT_k$, with $OUT_i \subseteq N_i$;
- a mapping $expand : N \rightarrow \{0, 1, \dots, k\}$ such that $expand(u) = 0$, for each $u \in IN \cup OUT$. We define the closure of $expand$, $expand^+ : N \rightarrow 2^{\{0, 1, \dots, k\}}$, as: $h \in expand^+(u)$ if either $h = expand(u)$ or $u' \in N_{expand(u)}$ exists such that $h \in expand^+(u')$.
- the sets of edges E_i , for $1 \leq i \leq k$, such that each edge in E_i is either a pair (u, v) , with $u, v \in N_i$ and $expand(u) = 0$, or a triple $((u, z), v)$ with $z \in OUT_{expand(u)}$, and $u, v \in N_i$;
- a mapping $true : N \rightarrow 2^{AP}$, such that $true(u) \cap true(v) = \emptyset$, for $v \in N_h, u \notin N_h$ and $h \in expand^+(u)$.

Informally, a **CRSM** is a collection of graphs which can call each other recursively. Each graph has an initial vertex and some output vertices. The mapping $expand$ gives the recursive-call structure. If $expand(u) = j > 0$, then the vertex u expands to the graph M_j and u is called a *supernode*; when $expand(u) = 0$ the vertex u is called a *node*. The mapping $true$ labels each vertex with a set of atomic propositions holding at that vertex. The *starting* node of a **CRSM** $\mathcal{M} = (M_1, \dots, M_k)$ is the initial node in_k of M_k .

The Semantics of CRSMs. Every **CRSM** $\mathcal{M}$ corresponds to a flat model $\mathcal{M}^F$ which is a directed graph with (possibly infinite) vertices (*states*) labeled with atomic propositions. Informally speaking, the flat machine $\mathcal{M}^F$ is obtained starting from M_k and iteratively replacing every supernode u in it with the graph $M_{expand(u)}$. The flat machine $\mathcal{M}^F$ is defined as follows. A state of $\mathcal{M}^F$ is a tuple $X = [u_1, \dots, u_m]$ where $u_1 \in N_k$, $u_{j+1} \in N_{expand(u_j)}$ for $j = 1, \dots, m-1$, and $expand(u_m) = 0$. State X is labeled by a set of atomic proposition $true(X)$, consisting of the union of $true(u_j)$, for $j = 1, \dots, m$. State $[in_k]$ is the initial state of $\mathcal{M}^F$. The set of transitions E is defined as follows. Let $X = [u_1, \dots, u_m]$ be a state with $u_m \in N_h$ and $u_{m-1} \in N_j$. Then, $(X, X') \in E$ provided that one of the following cases holds:

1. $(u_m, u') \in E_h$, $u' \in N_h$, and if $expand(u') = 0$ then $X' = [u_1, \dots, u_{m-1}, u']$, otherwise $X' = [u_1, \dots, u_{m-1}, u', in_l]$ for $l = expand(u')$.
2. $u_m \in OUT_h$, $((u_{m-1}, u_m), u') \in E_j$, $u' \in N_j$, and if $expand(u') = 0$ then $X' = [u_1, \dots, u_{m-2}, u']$, otherwise $X' = [u_1, \dots, u_{m-2}, u', in_l]$ for $l = expand(u')$.

Let $[u_1, \dots, u_n]$ be a state of $\mathcal{M}^F$, a *prefix* of $[u_1, \dots, u_n]$ is $u_1, \dots, u_i$ for $i \leq n$.

A *Context-dependent Hierarchic State Machine* (**CHSM**) is a **CRSM** such that $\text{expand}(u) < i$, for every $u \in N_i$. A **CHSM** is a collection of graphs which are organized to form a hierarchy and expand gives the hierarchical structure. The graph M_k is clearly the *top-level* graph of the hierarchy, i.e., no vertices expand to it and, as for **CRSMs**, its initial node in_k is the starting node of the **CHSM**.

3 Reachability and Cycle Detection Problems: Computational Complexity

In this section we discuss the computational complexity of the reachability and cycle detection problems for **CRSMs** and **CHSMs**. Given a **CRSM** $\mathcal{M} = (M_1, \dots, M_k)$ and a propositional boolean formula φ , the *reachability problem* is the problem of deciding if a path in $\mathcal{M}^F$ exists from $[\text{in}_k]$ to a state X on which φ is satisfied. Analogously, the *cycle detection problem* is the problem of deciding if a cycle in $\mathcal{M}^F$ exists containing a reachable state X on which φ is satisfied.

We prove that for **CRSMs** and **CHSMs** these decision problems are NP-complete by showing NP-hardness for **CHSMs** and giving nondeterministic polynomial-time algorithms for **CRSMs**.

Lemma 1. *Reachability and cycle detection for CHSMs are NP-hard.*

Proof We give a reduction in linear time with respect to the size of the formula from the satisfiability problem SAT. Given a boolean formula φ over the variables $x_1, \dots, x_m$, we construct a **CHSM** $\mathcal{M} = (M_1, M_2, \dots, M_m)$ over $AP = \{P_1, P_2, \dots, P_m\}$, as follows. Each graph M_i has four vertices $\text{in}_i, p_i, \text{not}p_i, \text{out}_i$ forming a chain. Each vertex p_i is labeled by $\{P_i\}$ whereas the vertices $\text{not}p_i, \text{in}_i$ and out_i are labeled by the empty set. Since an atomic proposition P_i does not label vertices in graphs other than M_i , this labeling implicitly corresponds to assigning $\neg P_i$ to $\text{not}p_i$. Vertices p_i and $\text{not}p_i$, for $i > 1$, are supernodes which expand into M_{i-1} , and p_1 and $\text{not}p_1$ are instead nodes.

Thus there are 2^m states of $\mathcal{M}^F$ of type $[u_1, \dots, u_m]$ such that $u_{m-i+1} \in \{p_i, \text{not}p_i\}$ for $i = 1, \dots, m$, and it is easy to verify that all these states are reachable from $[\text{in}_m]$. Clearly, given a truth assignment ν of $x_1, \dots, x_m$, a state X of $\mathcal{M}^F$ exists such that ν assigns TRUE to x_i if and only if p_i occurs in X and, in turns, if and only if $P_i \in \text{true}(X)$. Thus a reachable state X of $\mathcal{M}^F$ exists whose labeling corresponds to a truth assignment fulfilling φ if and only if φ is satisfiable.

By definition of the cycle detection problem, checking for the existence of a cycle containing a state on which φ is satisfied requires to check for reachability first. Thus, NP-hardness is inherited from reachability. $\square$

To prove membership to NP of the reachability on **CRSMs**, we need to consider a notion of connectivity of vertices in a **CRSM**. We say that a vertex $u \in N$ is *connected* if a reachable state $[u_1, \dots, u_m]$ of $\mathcal{M}^F$ exists, where $u = u_i$ for some $i = 1, \dots, m$. Observe that the starting node in_k is clearly connected

and a vertex $u \in N_j$ is connected if and only if in_j is connected and a path π in M_j from in_j to u exists, such that if π goes through an edge $((v, z), v') \in E_j$ then z is a connected vertex (recall that $z \in \text{OUT}_{\text{expand}(v)}$). From this the following proposition holds.

Proposition 1. *A state $[u_1, \dots, u_m]$ of $\mathcal{M}^F$ is reachable if and only if all the vertices u_i , for $i = 1, \dots, m$, are connected.*

The above observation suggests also an algorithm to determine in linear time the connected vertices. We omit the proof of this result which is given by a rather simple modification of a depth-first search on a graph (see also [AEY01]).

Proposition 2. *Given a CRSM $\mathcal{M}$, the set of connected vertices of $\mathcal{M}$ can be determined in $O(|\mathcal{M}|)$.*

To prove membership to NP of the reachability on CRSMs, we need to prove the following technical lemma. Notice that this lemma is not needed for CHSMs, where the number of supernodes that compose a state of $\mathcal{M}^F$ is bounded from above by the number of component graphs.

Lemma 2. *Given a CRSM $\mathcal{M}$, for each state $X = [u_1, \dots, u_m]$ such that $m > n^2 + 1$, where n is the number of supernodes of $\mathcal{M}$, a state $X' = [u'_1, \dots, u'_{m'}]$ exists such that $m' < m$ and $\text{true}(X) = \text{true}(X')$. Moreover, if X is reachable then also X' is reachable.*

Proof Consider a sequence $v_1, \dots, v_h \in N$. We say that a sub-sequence $v_i \dots v_j$, $1 \leq i < j$, is a *cycle* if $v_i = v_j$. Moreover, we say that a cycle $v_i \dots v_j$, is *erasable* if $\{v_{i+1}, \dots, v_j\} \subseteq \{v_1, \dots, v_i\}$. It is easy to verify that for a sequence $u_1 \dots u_m$ such that $X = [u_1, \dots, u_m]$ is a state of $\mathcal{M}^F$ and $u_i \dots u_j$ is a cycle, we have that $X' = [u_1, \dots, u_i, u_{j+1}, \dots, u_m]$ is a state of $\mathcal{M}^F$ and if $u_i \dots u_j$ is also erasable then $\text{true}(X) = \text{true}(X')$. Moreover, by Proposition 1, if X is reachable then also X' is reachable.

To conclude the proof we only need to show that for each state $X = [u_1, \dots, u_m]$ such that $m > n^2 + 1$, where n is the number of supernodes in $\mathcal{M}$, $u_1 \dots u_m$ contains an erasable cycle. Notice that $m > n^2 + 1$ implies that a supernode u exists, occurring at least $(n + 1)$ times in $u_1 \dots u_m$. Suppose $u_1 \dots u_m = \alpha_0 u \alpha_1 u \dots \alpha_n u \beta$, where each α_i does not contain occurrences of u . A cycle $u \alpha_i u$ is not erasable only if it contains a supernode that is not in $\alpha_0 u \dots \alpha_{i-1} u$. By a simple count, if $\alpha_0 u \dots \alpha_{n-1} u$ does not contain erasable cycles, then all supernodes occur in it. Thus, $u \alpha_n u$ is erasable. $\square$

Now, we can prove membership to NP of the reachability and the cycle detection problems on CRSMs.

Lemma 3. *Reachability and cycle detection for CRSMs are decidable in non-deterministic polynomial-time.*

Proof Consider the instance of the reachability problem given by a CRSM $\mathcal{M}$ and a propositional boolean formula φ . By Proposition 2 we can determine in

$O(|\mathcal{M}|)$ time the set of the connected vertices, and then, given a state X of $\mathcal{M}^F$, by Proposition 1 we can check if X is reachable in $O(|\mathcal{M}| + |X|)$ time. Verifying the fulfillment of φ on X takes $O(|\varphi| + |X|)$ time. Moreover, by Lemma 2 we need only to consider states $X = [u_1, \dots, u_m]$ for $m \leq n^2 + 1$, where n is the number of supernodes of $\mathcal{M}$. Thus, we can conclude that the reachability problem on **CHSMs** is in NP. $\square$

By Lemmas 1 and 3 we have the following theorem.

Theorem 1. *Reachability and cycle detection for **CRSMs** (**CHSMs**) are NP-complete.*

4 Efficient Solutions to Reachability and Cycle Detection Problems

In this section, we give a linear time algorithm that solves reachability and cycle detection problems for **CHSMs** which are related to target sets by a particular condition (specified later). As a corollary we get three consequences: first the results regarding reachability and cycle detection for the model considered in [AY01] are obtained as particular cases, second we characterize a class of formulas guaranteeing that the algorithm works correctly and finally we show that the algorithm works also for DNF formulas, thus obtaining a general solution for any formula with a tight worst case running time of $O(|\mathcal{M}| \cdot 2^{|\varphi|})$.

Finally, we give a linear time reduction from the reachability problem on **CRSMs** for DNF formulas to the corresponding problem on **CHSMs**, thus the above general solution still holds for **CRSMs**.

Consider now **CHSMs**. Clearly a propositional formula φ can be evaluated in a state X of $\mathcal{M}^F$ by instantiating to TRUE the variables corresponding to the atomic propositions in $true(X)$ and to FALSE all the others. Now we wish to evaluate φ without constructing the graph $\mathcal{M}^F$, to this aim we use a greedy approach in a top-down fashion on the hierarchy: at each supernode we instantiate as many variables as possible. By traversing the hierarchy in a top-down fashion, once a node is reached, φ can only *partially* evaluated. On a supernode u of a **CHSM** all the variables instantiated to TRUE correspond to the atomic propositions in $true(u)$. Determining the variables to instantiate to FALSE is not so immediate. We define $AP(h)$ as the union of the sets labeling either the vertices in N_h or those having an ancestor in N_h , that is, $AP(h) = \bigcup_{v \in N_h} (true(v) \cup AP(expand(v)))$ where $AP(0) = \emptyset$. Moreover, for $u \in N_h$, we define the set $false(u)$ as $AP(h) \setminus (true(u) \cup AP(expand(u)))$. This set contains the atomic propositions that can be instantiated to FALSE at u , since a proposition $p \in false(u)$ if and only if $p \notin true(X)$, for every state X of $\mathcal{M}^F$ having the supernode u as a component. It is easy to see that the sets $false(u)$, $u \in N$, can be preprocessed in time $O(|\mathcal{M}|)$, by visiting $\mathcal{M}$ in a bottom-up way.

For a propositional boolean formula φ we denote by $Eval(\varphi, u)$ the formula obtained by instantiating φ with $true(u)$ and $false(u)$. We generalize this notation to sequences of vertices defining $Eval(\varphi, u_1, \dots, u_i)$ as

Algorithm Reachability ($\mathcal{M}, \varphi$) return(Reach (M_k, φ)).
Function Reach (M_h, φ) VISITED[h] $\leftarrow$ MARK ; foreach $u \in N_h$ do $\varphi' = \text{Eval}(\varphi, u)$; if ($\varphi' == \text{TRUE}$) then return TRUE ; if ($\varphi' == \text{FALSE}$) then continue ; if (($\text{expand}(u) > 0$) AND ($\text{VISITED}[\text{expand}(u)] \neq \text{MARK}$)) then if Reach ($M_{\text{expand}(u)}, \varphi'$) then return TRUE ; endfor return FALSE ;

Fig. 1. Algorithm Reachability.

$\text{Eval}(\text{Eval}(\varphi, u_1), u_2, \dots, u_i)$. Finally, we will denote by $AP(\varphi)$ the set of atomic propositions corresponding to φ variables.

We consider a condition relating a **CHSM** $\mathcal{M}$ and a target set specified by a formula φ asserting that "when two supernodes expand to the same graph, then any partial evaluation of φ ending on them coincides". Formally, the condition is as follows:

Condition 1 Let $x_1, \dots, x_i$ and $y_1, \dots, y_j$ be two prefixes of $\mathcal{M}^F$ states such that $\text{expand}(x_i) = \text{expand}(y_j)$. If neither $\text{Eval}(\varphi, x_1, \dots, x_i)$ nor $\text{Eval}(\varphi, y_1, \dots, y_j)$ is one of the constants $\{\text{TRUE}, \text{FALSE}\}$, then $\text{Eval}(\varphi, x_1, \dots, x_i) = \text{Eval}(\varphi, y_1, \dots, y_j)$.

When reachability and cycle detection become tractable.

Theorem 2. The reachability and cycle detection problems on a **CHSM** $\mathcal{M}$ and a formula φ satisfying Condition 1 are decidable in time $O(|\mathcal{M}| \cdot |\varphi|)$.

Proof Consider a **CHSM** $\mathcal{M} = (M_1, \dots, M_k)$ and without loss of generality assume that all the vertices of $\mathcal{M}$ are connected (see Proposition 2). Algorithm **Reachability**($\mathcal{M}, \varphi$) (Figure 1), returns **TRUE** if and only if φ is evaluated to **TRUE** on a reachable state of $\mathcal{M}^F$. The function **Reach** uses a global array VISITED (initially unmarked in all the positions) to mark the visited graphs M_h . For each node u of M_h , φ is evaluated on it according to $\text{true}(u)$ and $\text{false}(u)$, call φ' the returned formula. If φ' evaluates **TRUE** on u , then **Reach** stops returning **TRUE**. (and the main algorithm stops too returning **TRUE**). If φ' evaluates **FALSE**, another vertex of M_h which has not yet been explored is processed. In case u is a supernode and $M_{\text{expand}(u)}$ has never been visited, then the function is called on the graph $M_{\text{expand}(u)}$ and φ' . Now note that Condition 1 assures that it is not necessary to visit a graph M_h more than once, thus the overall complexity of the algorithm is linear in $|\mathcal{M}|$ and $|\varphi|$ and clearly returns **TRUE** if and only if a node X in $\mathcal{M}^F$ exists on which φ is **TRUE**. $\square$

It is easy to see that given any formula φ and a Hierarchical State Machine (HSM) introduced in [AY01] (where only nodes are labeled with the mapping *true*, see the introduction), Condition 1 always holds, thus the linear time solutions for the reachability and cycle detection problems for HSM given in that paper are here obtained as particular cases.

Now we present a characterization of formulas for which Theorem 2 holds. A propositional boolean formula φ is said to be in *$\mathcal{M}$ -normal form* if $\varphi = \varphi_1 \wedge \dots \wedge \varphi_m$ and for every φ_i and for every vertex u of $\mathcal{M}$ it holds that either $AP(\varphi_i) \cap (\text{true}(u) \cup \text{false}(u)) = \emptyset$ or $AP(\varphi_i) \cap (\text{true}(u) \cup \text{false}(u)) = AP(\varphi_i)$. It is easy to see that also in this case Condition 1 holds.

Theorem 2 can be generalized for a finite disjunction of formulas satisfying Condition 1. Since a conjunction of literals is in *$\mathcal{M}$ -normal form*, for all possible $\mathcal{M}$, then this generalization can be applied to DNF formulas. Thus, as any formula φ can always be transformed in a DNF formula, we have an algorithm for reachability and cycle detection problems whose worst case running time is $O(|\mathcal{M}| \cdot \text{DNF}(\varphi))$, where $\text{DNF}(\varphi)$ is the cost of the transformation of φ in Disjunctive Normal Form. All this yields a tight upper bound of $O(|\mathcal{M}| \cdot 2^{|\varphi|})$.

Reachability and cycle detection are also tractable on **CRSMs** if we restrict to formulas in disjunctive normal form as shown in the following theorem.

Theorem 3. *Reachability and cycle detection problems for a **CRSM** $\mathcal{M}$ and a formula φ in DNF are decidable in time $O(|\mathcal{M}| \cdot |\varphi|)$.*

Proof Consider a **CRSM** $\mathcal{M}$ and a DNF formula $\varphi = \psi_1 \vee \dots \vee \psi_m$ where each ψ_i is a conjunction of literals. Our algorithm consists of reducing in $O(|\varphi| \cdot |\mathcal{M}|)$ time the reachability problem for $\mathcal{M}$ and ψ_i to the reachability problem for a **CHSM** $\bar{\mathcal{M}}$ and ψ_i , where size of $\bar{\mathcal{M}}$ is $O(|\mathcal{M}|)$. Then the result follows from Theorem 2.

Consider a disjunct clause ψ of φ . We simplify $\mathcal{M}$ using the following two steps.

1. for each graph M_i , delete all the existing edges and insert an edge from in_i to any other connected vertex of M_i ;
2. if u is not an initial node and $\text{true}(u)$ contains an atomic proposition corresponding to a variable which is negated in ψ , then delete u from M_i .

This transformation can be performed in $O(|\psi| \cdot |\mathcal{M}|)$ time and preserves the reachability of the states of $\mathcal{M}^{\mathcal{F}}$ satisfying ψ , thanks to Proposition 2.

Now, define a supernode $u \in N_i$ as *recursively expansible* if $i \in \text{expand}^+(u)$ and a graph M_i as *recursively expansible* if it contains at least a recursively expansible supernode. We define the equivalence relation $\approx$ on the indices of recursively expansible graphs: $i \approx j$ if and only if vertices $u \in N_i$ and $v \in N_j$ exist such that $i \in \text{expand}^+(v)$ and $j \in \text{expand}^+(u)$. We want to define a **CHSM** $\bar{\mathcal{M}} = (\bar{M}_1, \bar{M}_2, \dots, \bar{M}_{k'})$ such that $\bar{\mathcal{M}}$ has a component graph for each equivalence class of the relation $\approx$. Let $f : \{1, \dots, k\} \rightarrow \{1, \dots, k'\}$ be the function that maps each i to j such that i is in the equivalence class corresponding to $\bar{M}_j$.

For a graph M_i which is not recursively expansible (i.e., $[i] = \{i\}$), we define $\bar{M}_{f(i)}$ as M_i except for the mapping $expand$, since $expand_{\bar{\mathcal{M}}}(u) = f(expand_{\mathcal{M}}(u))$. For a recursively expansible graph M_i we define $\bar{M}_{f(i)}$ as follows. All vertices $u \in N_j$ which are not recursively expansible, with $j \approx i$, are vertices of $\bar{M}_{f(i)}$, the edges between them in M_i are edges of $\bar{M}_{f(i)}$ as well and $OUT_{f(i)} = \bigcup_{j, j \approx i} OUT_j$. Moreover, we add a new initial node $in_{f(i)}$ and insert edges from $in_{f(i)}$ to all vertices in_j , $j \approx i$. For each supernode u of $\bar{M}_{f(i)}$ we define $expand_{\bar{\mathcal{M}}}(u) = f(expand_{\mathcal{M}}(u))$. Let $S_{\mathcal{M}}(i)$ be the set of all recursively expansible vertices belonging to all graphs M_j such that $j \approx i$. We define $true_{\bar{\mathcal{M}}}(in_{f(i)})$ as $true_{\bar{\mathcal{M}}}(in_j)$ for an arbitrary $j \approx i$, and for each vertex u of $\bar{M}_{f(i)}$, $true_{\bar{\mathcal{M}}}(u)$ as $\bigcup_{v \in S_{\mathcal{M}}(i)} true_{\mathcal{M}}(v) \cup true_{\mathcal{M}}(u)$ (note that no atomic proposition added in this way to the label of u corresponds to a variable which is negated in ψ).

Now observe that, by the above part 2 of the above simplification, if X is a state of $\mathcal{M}^F$ satisfying ψ and Y is a state of $\bar{\mathcal{M}}^F$ such that $true_{\mathcal{M}}(X) \subseteq true_{\bar{\mathcal{M}}}(Y)$ and $true_{\bar{\mathcal{M}}}(Y) \setminus true_{\mathcal{M}}(X)$ does not contain an atomic proposition corresponding to a variable which is negated in ψ , then Y satisfies ψ as well. Since the initial simplification also preserves reachability, we have that if a reachable state of $\mathcal{M}^F$ fulfilling ψ exists, then a state of $\bar{\mathcal{M}}^F$ fulfilling ψ also exists. Since by construction, states of $\bar{\mathcal{M}}^F$ corresponds to states of $\mathcal{M}^F$, the vice-versa also holds. $\square$

As a consequence of Theorem 3 and the arguments for **CHSMs** and DNF formulas, the following theorem holds.

Theorem 4. *The reachability and cycle detection problems on a **CRSM** $\mathcal{M}$ and a propositional boolean formula φ are decidable in $O(|\mathcal{M}| \cdot 2^{|\varphi|})$ time.*

5 LTL Model Checking

Here we consider the verification problem of linear-time requirements, expressed by LTL-formulas [Pnu77]. We follow the automata theoretic approach to solving model checking [VW86]: given an LTL formula φ and a Kripke structure M , it is possible to reduce model checking to the emptiness problem of Büchi automata. To use this approach, we extend the Cartesian product between Kripke structures.

Given a transition graph with states labeled by subsets of atomic propositions and a state s , a *trace* is an infinite sequence $\alpha_1 \alpha_2 \dots \alpha_i \dots$ of labels of states occurring in a path starting from s . Moreover, given a **CRSM** $\mathcal{M}$, we define the language $\mathcal{L}(\mathcal{M})$ as the set of the traces of $\mathcal{M}^F$ starting from its initial state. A Büchi automaton $A = (Q, q_1, \Delta, L, T)$ is a Kripke structure (Q, Δ, L) together with a set of accepting states T and a starting state q_1 . The language $\mathcal{L}(A)$ accepted by A is the set of the traces corresponding to paths visiting infinitely often a state of T .

Let $\mathcal{M} = (M_1, \dots, M_k)$ be a **CRSM** and $A = (Q, q_1, \Delta, L, T)$, for $Q = \{q_1, \dots, q_m\}$, be a Büchi automaton. Let $1 \leq i \leq k$, $1 \leq j \leq m$, and P be such

that $P \subseteq AP$ and $P \cup \text{true}_{\mathcal{M}}(in_i) = L(q_j)$, we define the graphs $M_{(i,j,P)}$ as follows. Each $M_{(i,j,P)}$ contains vertices $[u, q, j, P]$ such that (u, q) belongs to the standard Cartesian product of M_i and A , and the labeling of q coincides with the labeling of u augmented with the atomic propositions that u inherits from its ancestors in a given context. The inherited set of atomic propositions is given by P . The property $P \cup \text{true}_{\mathcal{M}}(in_i) = L(q_j)$ assures that we consider only graphs $M_{(i,j,P)}$ whose initial vertex is compatible with the automaton state. Formally, we have:

- The set $N_{(i,j,P)}$ of the vertices of $M_{(i,j,P)}$ contains quadruples $[u, q, j, P]$, where $u \in N_i$, $q \in Q$, and
 - either $\text{expand}_{\mathcal{M}}(u) = 0$ and $L(q) = \text{true}_{\mathcal{M}}(u) \cup P$
 - or $\text{expand}_{\mathcal{M}}(u) = h > 0$ and $L(q) = \text{true}_{\mathcal{M}}(u) \cup \text{true}_{\mathcal{M}}(in_h) \cup P$.
- The initial vertex of $M_{(i,j,P)}$ is $[in_i, q_j, j, P]$ and the output nodes are $[u, q, j, P]$ for $u \in OUT_i$ and $q \in Q$;
- $M_{(i,j,P)}$ contains the following edges:
 - $([u, q', j, P], [v, q'', j, P])$, with $(q', q'') \in \Delta$ and $(u, v) \in E_i$,
 - $([u, q_{j'}, j, P], [z, q', j', P \cup \text{true}_{\mathcal{M}}(u)]), [v, q'', j, P])$, with $(q', q'') \in \Delta$, $((u, z), v) \in E_i$, and $L(q) = \text{true}_{\mathcal{M}}(u) \cup \text{true}_{\mathcal{M}}(in_h) \cup P$ for $\text{expand}_{\mathcal{M}}(u) = h$.

From the above definition we observe that if u is a supernode then the labeling of q has to match also with the labeling of $in_{\text{expand}_{\mathcal{M}}(u)}$ since $[u, q, j, P]$ is a supernode of $\mathcal{M}'$ and one has to assure the correctness, with respect to the labeling, of its expansion. Note that when only the value of j varies, we have graphs which differ one each other only for the choice of the the initial vertex $[in_i, q_j, j, P]$. Moreover, the edges in $M_{(i,j,P)}$ are given by coupling the transitions (q', q'') of A with both kinds of edges (u, v) and $((u, z), v)$ in E_i . For $h = \text{expand}_{\mathcal{M}}(u)$, we have edges $([u, q, j, P], [z, q', h, P \cup \text{true}_{\mathcal{M}}(u)]), [v, q'', j, P])$ for every $q \in Q$ such that $L(q) = \text{true}_{\mathcal{M}}(u) \cup \text{true}_{\mathcal{M}}(in_h) \cup P$. Thus, there might be as many as $|Q|$ edges, for every pair of edges $((u, z), v)$ and (q', q'') .

We can now define $\mathcal{M}' = \mathcal{M} \otimes A$ as a **CRSM** constituted by some of the graphs $M_{(i,j,P)}$, and defined inductively as follows:

- $M_{(k,1,\emptyset)}$ is the graph containing the starting node of $\mathcal{M}'$;
- Let $M_{(i,j,P)}$ be a graph of $\mathcal{M}'$, and $[u, q_t, j, P]$ be a vertex of $M_{(i,j,P)}$.
 - If $\text{expand}_{\mathcal{M}}(u) = 0$ then $\text{expand}_{\mathcal{M}'}([u, q_t, j, P]) = 0$;
 - If $\text{expand}_{\mathcal{M}}(u) = h > 0$, and $P' = P \cup \text{true}_{\mathcal{M}}(u)$ then $M_{(h,t,P')}$ is a graph of $\mathcal{M}'$ and $\text{expand}_{\mathcal{M}'}([u, q_t, j, P]) = \langle h, t, P' \rangle$ where $\langle h, t, P' \rangle$ denotes the index of $M_{(h,t,P')}$;
- $\text{true}_{\mathcal{M}'}([u, q, j, P]) = \text{true}_{\mathcal{M}}(u)$, for every $[u, q, j, P]$.

Observe that $\mathcal{M}' = \mathcal{M} \otimes A$ is a **CRSM** and if $\mathcal{M}$ is a **CHSM**, then $\mathcal{M}'$ is a **CHSM**, as well. To determine the size of $\mathcal{M}'$, first consider the size of each graph $M_{(i,j,P)}$. The number of the edges is bounded by the product of the number of edges in M_i and the number of transitions in A multiplied at most by m , since we have at most $|Q|$ edges for any $(q', q'') \in \Delta$ and $((u, z), v) \in E_i$. Thus, an upper bound to the size of $M_{(i,j,P)}$ is given by $(m \cdot |E_i| \cdot |A|)$. The size of $\mathcal{M}'$ can be obtained now by counting the number of its component graphs.

Lemma 4. *Given a **CRSM** $\mathcal{M}$, $\mathcal{M}' = \mathcal{M} \otimes A$ is a **CRSM** that can be constructed in $O(m^2 \cdot |\mathcal{M}| \cdot |A| \cdot |2^{AP}|)$ time. Moreover, if $\mathcal{M}$ is a **CHSM**, then $\mathcal{M}'$ is a **CHSM** that can be constructed in $O(m^2 \cdot |\mathcal{M}| \cdot |A|)$ time.*

Proof First recall that a graph $M_{(i,j,P)}$ of $\mathcal{M}'$ has the property that $P \cup \text{true}_{\mathcal{M}}(\text{in}_i) = L(q_j)$. Therefore, P is the union of two disjoint sets P_1 and P_2 , such that P_1 is the set of the atomic propositions of $L(q_j)$ that do not belong to $\text{true}_{\mathcal{M}}(\text{in}_i)$, and $P_2 = P \cap \text{true}_{\mathcal{M}}(\text{in}_i)$ is a subset of $\text{true}_{\mathcal{M}}(\text{in}_i)$. Thus, for fixed values of i and j , P_1 is fixed and the number of different graphs $M_{(i,j,P)}$ is bounded above by the number of different subsets of $\text{true}_{\mathcal{M}}(\text{in}_i)$. Therefore, the size of $\mathcal{M}'$ is bounded above by $\sum_{j=1}^m \sum_{i=1}^k (2^{|AP|} \cdot m \cdot |M_i| \cdot |A|)$.

Now, let $\mathcal{M}$ be a **CHSM**. Given a graph $M_{(i,j,P)}$ of $\mathcal{M}'$, P is defined as the set of the propositions that the vertices of M_i inherit. Thus, $P \cap \text{true}_{\mathcal{M}}(u) = \emptyset$, for every vertex u of M_i and then $P \cap \text{true}_{\mathcal{M}}(\text{in}_i) = \emptyset$. Hence, in this case, P_2 is empty and then at most one graph $M_{(i,j,P)}$ exists for fixed values of i and j . Therefore, the size of $\mathcal{M}'$ is bounded above by $\sum_{j=1}^m \sum_{i=1}^k (m \cdot |M_i| \cdot |A|)$. $\square$

The **CRSM** $\mathcal{M} \otimes A$ can be used to check for the emptiness of the language given by the intersection of $\mathcal{L}(\mathcal{M})$ and $\mathcal{L}(A)$, as shown in the following lemma.

Lemma 5. *There exists an algorithm checking whether $\mathcal{L}(\mathcal{M}) \cap \mathcal{L}(A) = \emptyset$ in time linear in the size of $\mathcal{M}' = \mathcal{M} \otimes A$.*

Proof First, observe that if we consider as set of final states the vertices $[u, q, h, P]$ such that $q \in T$, the **CRSM** $\mathcal{M}'$ is a recursive Büchi automaton. Moreover, the set of the traces of $\mathcal{M}'^F$ is the same as the set of traces of the Cartesian product of $\mathcal{M}^F$ and A . Thus $\mathcal{L}(\mathcal{M}) \cap \mathcal{L}(A) \neq \emptyset$ if and only if $\mathcal{L}(\mathcal{M}') \neq \emptyset$. From [AEY01], for recursive Büchi automata with a single initial node for each graph, non-emptiness can be checked in linear time. $\square$

As a consequence of the above lemmas, we obtain an algorithm to solving the LTL model checking for **CRSMs**. Following the automata theoretic approach, one can construct a Büchi automaton $A_{\neg\varphi}$ of size $O(2^{|\varphi|})$ accepting the set $\mathcal{L}(A_{\neg\varphi})$ of the sequences which do not satisfy φ , and then φ is satisfied on all paths of $\mathcal{M}$ if and only if $\mathcal{L}(\mathcal{M}) \cap \mathcal{L}(A_{\neg\varphi})$ is empty. From Lemma 4, one can now construct $\mathcal{M} \otimes A_{\neg\varphi}$, whose size is $O(m^2 \cdot |\mathcal{M}| \cdot |A_{\neg\varphi}| \cdot 2^{|AP|}) = O(|\mathcal{M}| \cdot 16^{|\varphi|})$ (since $m = |A_{\neg\varphi}| = O(2^{|\varphi|})$ and $2^{|AP|} \leq 2^{|\varphi|}$). Moreover, this size reduces to $O(m^2 \cdot |\mathcal{M}| \cdot |A_{\neg\varphi}|) = O(|\mathcal{M}| \cdot 8^{|\varphi|})$, when $\mathcal{M}$ is a **CHSM**. Hence, by Lemma 5 we obtain the main result of this section.

Theorem 5. *The LTL model checking on a **CRSM** $\mathcal{M}$ and a formula φ can be solved in $O(|\mathcal{M}| \cdot 16^{|\varphi|})$ time. Moreover, if $\mathcal{M}$ is a **CHSM** the problem can be solved in $O(|\mathcal{M}| \cdot 8^{|\varphi|})$ time.*

6 Discussion

We have proposed new abstract models for sequential state machines: the context-dependent hierarchical and recursive state machines. On these models we have studied reachability, cycle detection and the more general problem

of model checking with respect to linear-time specifications. An interesting feature of **CHSMs** is that they allow very succinct representations of systems, and this comes substantially at no cost if compared to analogous hierarchical models studied in the literature. Moreover, we prove that for some particular formulas we improve the complexity of previous approaches.

Several extensions of the introduced models can be considered.

Our models are sequential. If we add concurrency to **CHSMs**, the computational complexity of the considered decision problems grows significantly (we recall that reachability in communicating hierarchical state machines is EXPSPACE-complete [AKY99]). While for **CRSMs** with concurrency, reachability becomes undecidable since sequential **CRSMs** are as expressive as pushdown automata [AEY01,BGR01].

We have only considered models where a single entry node is allowed for each component machine. We can relax this limitation allowing multiple entry points. The semantics of this extension naturally follows from the semantics given for the single entry case. In the hierarchic setting, we can translate a multiple-entry **CHSM** $\mathcal{M}$ into an equivalent single-entry **CHSM** $\mathcal{M}'$ of size at most cubic in the size of $\mathcal{M}$. In fact, each component machine of $\mathcal{M}$ can be replaced in $\mathcal{M}'$ by multiple copies, each copy corresponding to an entry point and having as unique entry point the entry point itself. Expansions are redirected to the proper components in order to match the expansions in $\mathcal{M}$. Thus, supernodes may need to be replaced by multiple copies each pointing to the proper machine in $\mathcal{M}'$. If we apply this construction to a multiple-entry **CRSM**, the obtained single-entry **CRSM** does not satisfy the property $true(u) \cap true(v) = \emptyset$, for $v \in N_h, u \notin N_h$ and $h \in expand^+(u)$ (see definition of **CRSM**). This is a consequence of the fact that if a machine of the multiple-entry **CRSM** can directly or indirectly call itself, then there are two copies of this machine that may call each other recursively. We recall that the above property is sufficient to ensure that Condition 1 holds for conjunctions of literals, and thus is crucial to obtain the results given in Section 4. However, it is possible to prove that Theorem 3 also holds for multiple-entry **CRSMs**. We leave the details of this proof to the full paper.

For modeling purposes it is useful to have variables over a finite domain that can be passed from a component to another. We can extend our models to handle input, output and local variables. Consider a component machine M with h_e entry nodes, h_x exit nodes, and h_t internal vertices. If M is equipped also with k_i input boolean variables, k_o output boolean variables, and k_l local boolean variables, we can model by a machine having $2^{k_i} \cdot h_e$ entry nodes, $2^{k_o} \cdot h_x$ exit nodes, and $2^{k_i+k_l+k_o} \cdot h_t$ internal vertices.

References

- [AEY01] R. Alur, K. Etessami, and M. Yannakakis. Analysis of recursive state machines. In *Proc. of the 13th International Conference on Computer Aided Verification, CAV'01*, LNCS 2102, pages 207–220. Springer, 2001.

- [AG00] R. Alur and R. Grosu. Modular refinement of hierarchic reactive machines. In *Proc. of the 27th Annual ACM Symposium on Principles of Programming Languages*, pages 390–402, 2000.
- [AGM00] R. Alur, R. Grosu, and M. McDougall. Efficient reachability analysis of hierarchical reactive machines. In *Computer Aided Verification, 12th International Conference*, LNCS 1855, pages 280–295. Springer, 2000.
- [AKY99] R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. In *Proc. of the 26-th International Colloquium on Automata, Languages and Programming, ICALP'99*, LNCS 1644, pages 169–178. Springer-Verlag, 1999.
- [AY01] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(3):273–303, 2001.
- [BGR01] M. Benedikt, P. Godefroid, and T. W. Reps. Model checking of unrestricted hierarchical state machines. In *Proc. of the 28th International Colloquium Automata, Languages and Programming, ICALP'01*, LNCS 2076, pages 652–666. Springer, 2001.
- [BJR97] G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide*. Addison Wesley, 1997.
- [BLA⁺99] G. Behrmann, K.G. Larsen, H.R. Andersen, H. Hulgaard, and J. Lind-Nielsen. Verification of hierarchical state/event systems using reusability and compositonality. In *Proc. of the Tools and Algorithms for the Construction and Analysis of Systems, TACAS'99*, LNCS 1579, pages 163–177. Springer, 1999.
- [CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. of Workshop on Logic of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.
- [CK96] E.M. Clarke and R.P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.
- [Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–77, 1977.
- [RBP⁺91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented Modeling and Design*. Prentice-Hall, 1991.
- [SGW94] B. Selic, G. Gullekson, and P.T. Ward. *Real-time object oriented modeling and design*. J. Wiley, 1994.
- [VW86] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:182–211, 1986.

Oracle Circuits for Branching-Time Model Checking

Philippe Schnoebelen

Lab. Spécification & Vérification
ENS de Cachan & CNRS UMR 8643
61, av. Pdt. Wilson, 94235 Cachan Cedex France
`psh@lsv.ens-cachan.fr`

Abstract. A special class of oracle circuits with tree-vector form is introduced. It is shown that they can be evaluated in deterministic polynomial-time with a polylog number of adaptive queries to an NP oracle. This framework allows us to evaluate the precise computational complexity of model checking for some branching-time logics where it was known that the problem is NP-hard and coNP-hard.

1 Introduction

Many different *temporal logics* have been proposed in the computer science literature [5]. Their main use is in the field of reactive systems, where *model checking* allows automated verification of correctness [3].

Comparing and classifying the different temporal logics is an important task. This is usually done along several axis, most notably *expressive power* and *computational complexity*. Regarding computational complexity, several open questions remain [16]. In particular, for several branching-time temporal logics, the complexity of model checking is not known. Advances in this domain are welcome since it is important to understand what ideas underly “optimal” algorithms, and what special cases may benefit from specialized methods.

Model checking in the polynomial-time hierarchy. There is a family of branching-time temporal logics for which the complexity of model checking is not known precisely. These logics can be described as branching-time logics where the underlying path properties are in NP or coNP (we give several examples in section 2). This leads to a P^{NP} (that is, Δ_2^P) upper bound for the full logic.

For such logics, the question of finding matching lower bounds saw no progress until recently, when Laroussinie, Markey, and the author managed to prove that some of them (including $B^*(F)$, CTL^+ , and $FCTL$) have indeed a Δ_2^P -complete model checking problem [13,14].

However, for some remaining logics, the techniques used in [13,14] for proving Δ_2^P -hardness do not apply. The difficulty here is that, if these problems are *not* Δ_2^P -complete, we still lack methods for proving that a model checking problem has upper bounds higher than NP or coNP but lower than Δ_2^P .

Our contribution. In this paper we develop a framework that allows proving upper bounds below Δ_2^p and apply it to branching-time model checking problems. The approach is successful in that it allows us to prove model checking $B^*(X)$ is $\text{P}^{\text{NP}[O(\log^2 n)]}$ -complete, and model checking *Timed* $B(F)$ is $\text{P}^{\text{NP}[O(\log n)]}$ -complete.

Our framework is based on Boolean circuits with oracle queries (introduced in [21]). We identify two special classes of oracle circuits having *tree-vector* form (with special constraints on the oracle queries) for which we prove evaluation can be done in $\text{P}^{\text{NP}[O(\log n)]}$ and, respectively, $\text{P}^{\text{NP}[O(\log^2 n)]}$, i.e. they can be evaluated by a deterministic polynomial-time Turing machine that makes $O(\log n)$ (resp. $O(\log^2 n)$) adaptive queries to an NP-oracle (while Δ_2^p -complete problems require¹ polynomially-many adaptive queries).

Branching-time model checking problems lead naturally to tree-vector circuits, so that we obtain upper bounds directly by translations. The lower bounds are proved by ad-hoc reductions.

These results are important for several reasons:

1. The tree-vector oracle circuits may have more applications than just in model checking. In any case, they illuminate a structural feature of model checking where the formula is a modal expression *tree* evaluated over a *vector* of worlds.
2. The results help complete the picture in the classification of temporal logics. A logic like $B^*(X)$, the *full branching-time logic of “next”*, is perhaps not used in practice, but it is a fundamental fragment of CTL^* , for which we should be able to assess the complexity of model checking.
3. They provide examples of problems complete for $\text{P}^{\text{NP}[O(\log n)]}$ and $\text{P}^{\text{NP}[O(\log^2 n)]}$. Very few such examples are known. In particular, with the model checking of $B^*(X)$, we provide the first example (to the best of our knowledge) of a natural problem complete for $\text{P}^{\text{NP}[O(\log^2 n)]}$.

Related work. The best known framework for assessing the complexity of model checking problems is the automata-theoretic framework initiated by Vardi and Wolper [18]. By moving to tree-automata, this framework is able to deal with branching-time logics [12], where it has proved very successful. However, the tree-automata approach seems too coarse-grained for our problems where it seems we need a fine-grained look at the structure of the oracle calls.

Gottlob’s work on NP trees [9] was an inspiration. His result prompted us to check whether certain tree-vectors of queries could be normalized.

Plan of the paper. We recall the necessary background in Section 2. Then Section 3 is devoted to tree-vector oracle circuits and flattening algorithms for evaluating them. This lays the ground for our proof that model checking $B^*(X)$ is $\text{P}^{\text{NP}[O(\log^2 n)]}$ -complete (Section 4) and model checking *Timed* $B(F)$ is $\text{P}^{\text{NP}[O(\log n)]}$ -complete (Section 5). The proofs that have been omitted for lack of space appear in the full version.

¹ That is, assuming Δ_2^p does not collapse to $\text{P}^{\text{NP}[O(\log^2 n)]}$ and $\text{P}^{\text{NP}[O(\log^2 n)]}$! We shall often write such sentences that implicitly assume the separation conjectures most complexity theorists believe are true.

2 Branching-Time Logics with Model Checking in Δ_2^p

2.1 Complexity Classes below Δ_2^p

We assume familiarity with computational complexity. The main definitions we need concern classes in the polynomial-time hierarchy (see [10,15]).

Δ_2^p is the class P^{NP} of problems solvable by deterministic polynomial-time Turing machines that may query an NP oracle. Some relevant subclasses of Δ_2^p have been identified:

- $P^{NP[O(\log n)]}$ only allows $O(\log n)$ oracle queries instead of polynomially-many. For example, PARITY-SAT (the problem where one is asked whether the number of satisfiable Boolean formulae from some input set $f_1, \dots, f_n$ is odd or even) is $P^{NP[O(\log n)]}$ -complete [19].
- $P_{\parallel}^{NP}$ only allows one round of *parallel* queries: the polynomially-many queries may not be adaptive (i.e., depend on the outcomes of earlier oracle queries) but must first be all formulated *before* the oracle is consulted on all queries. Then the computation proceeds normally, using the polynomially-many oracle answers.

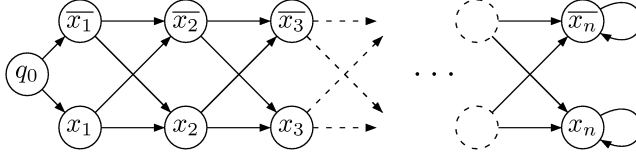
$P^{NP[O(\log n)]}$ and $P_{\parallel}^{NP}$ coincide (and they further coincide with $P_{\parallel O(1)}^{NP}$, where a fixed number of parallel rounds is allowed). Wagner showed that many different and natural ways of restricting Δ_2^p all lead to the same $P^{NP[O(\log n)]}$ class (e.g. $P^{NP[O(\log n)]}$ coincide with L^{NP}), for which he introduced the name Θ_2^p [20]. Further variants were introduced by Castro and Seara, who proved that, for all $k \in \mathbb{N}$, $P^{NP[O(\log^k n)]}$ coincide with $P_{\parallel O(\log^{k-1} n)}^{NP}$ (where a succession of $O(\log^{k-1})$ parallel querying rounds are allowed) [1].

2.2 Branching-Time Logics and NP-Hard Path Modalities

We assume familiarity with temporal logic model checking [5,3,16]. Several branching-time logics combine the path quantifiers E and A with linear-time modalities whose path existence problem is in NP. Here are five examples:

- *FCTL* [8], or “Fair *CTL*”, allows restricting to the fair paths of a Kripke structure, where the fair paths are defined by an arbitrary Boolean combination of $\tilde{F} \pm P_i$ s. The existence of a fair path is NP-complete [8].
- *TCTL* [11], or “Timed *CTL*”, allows adding timing subscripts to the usual modalities. In Timed KSs (i.e. Kripke structures where edges carry a discrete “duration” weight) the existence of a path of a given accumulated duration is NP-complete [14].
- *CTL*⁺ [6] allows arbitrary Boolean combinations (not nesting) of the U and X modalities under a path quantifier. Thus *CTL*⁺ is the branching-time extension of $L^1(U, X)$, the fragment of linear-time logic with modal depth one, for which the existence of a path is NP-complete [4].
- $B^*(F)$ and $B^*(X)$ are the branching-time extensions of $L(F)$ and $L(X)$ (resp.). $B^*(F)$ (called *BT** in [2]) is the *full branching-time logic of “eventually”*, while $B^*(X)$ is the *full branching-time logic of “next”*. The existence of a path satisfying an $L(F)$ or an $L(X)$ formula is NP-complete [17].

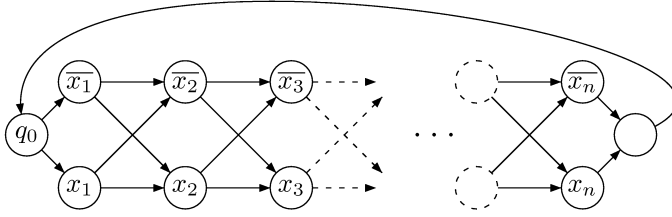
For these examples, NP-hardness is easy to prove by reduction from 3SAT. For example, consider an instance $\mathcal{I}$ of the form “ $(x_1 \vee \overline{x_2} \vee \overline{x_4}) \wedge (\overline{x_1} \vee \dots) \wedge \dots$ ”. With $\mathcal{I}$ we associate the following structure that applies to CTL^+ , $B^*(F)$, and $B^*(X)$:



$$\mathcal{I} \text{ is satisfiable iff } q_0 \models E \left[(F x_1 \vee F \overline{x_2} \vee F \overline{x_4}) \wedge (F \overline{x_1} \vee \dots) \wedge \dots \right] \quad (1)$$

$$\text{iff } q_0 \models E \left[(X x_1 \vee X X \overline{x_2} \vee X X X X \overline{x_4}) \wedge (X \overline{x_1} \vee \dots) \wedge \dots \right] \quad (2)$$

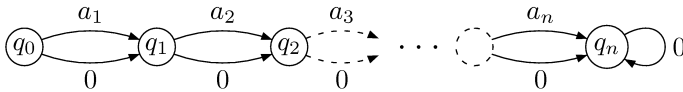
For $FCTL$ we use a slight variant:



Here $\mathcal{I}$ is satisfiable iff

$$q_0 \models E \left[\neg(\tilde{F} x_1 \wedge \tilde{F} \overline{x_1}) \wedge \neg(\tilde{F} x_2 \wedge \tilde{F} \overline{x_2}) \wedge \dots \wedge \neg(\tilde{F} x_n \wedge \tilde{F} \overline{x_n}) \right] \wedge (\tilde{F} x_1 \vee \tilde{F} \overline{x_2} \vee \tilde{F} \overline{x_4}) \wedge (\tilde{F} \overline{x_1} \vee \dots) \wedge \dots \quad (3)$$

For $TCTL$ we reduce from SUBSET-SUM. With an instance $\mathcal{I}$ of the form “can one add numbers taken from $\{a_1, \dots, a_n\}$ and obtain b ?” we associate the following Timed KS:



Obviously $\mathcal{I}$ is solvable iff $q_0 \models EF_{=b} q_n$.

2.3 Model Checking $B(L)$

Assume L is some linear-time logic, and write $B(L)$ for the associated branching-time logic. Emerson and Lei [8] observed that, from an algorithm for the existence

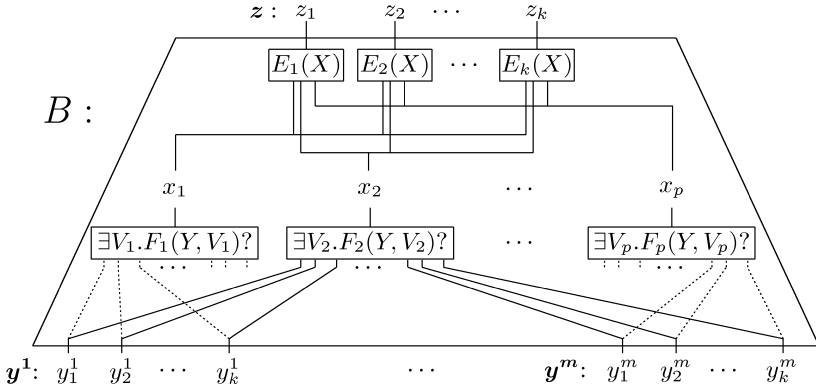


Fig. 1. General form of a “block” oracle circuit

of paths satisfying L properties, one easily derives a model checking algorithm for $B(L)$. Furthermore, this only needs a polynomial-time Turing reduction, so that if the existential problem for L belongs to some complexity class $\mathcal{C}$, then the model checking problem for $B(L)$ is in $P^{\mathcal{C}}$ [8].

Example 2.1. With path modalities having an NP-complete existential problem, $B^*(F)$, $B^*(X)$, CTL^+ , $ECTL^+$ (from [7]), $FCTL$, BTL_2 and $TCTL$ (over Timed KSs), all have a model checking in P^{NP} , the level called Δ_2^p in the polynomial-time hierarchy.² $\square$

Concerning the logics mentioned in Example 2.1, the only known lower bounds for their model checking problem were the obvious “NP-hard and coNP-hard” (or even DP-hard). However, all these logics have Θ_2^p -hard model checking (see Remark 5.3 below). Recently, Laroussinie, Markey and Schnoebelen showed Δ_2^p -hardness (hence Δ_2^p -completeness) for $FCTL$ and $B^+(F)$ in [13] (hence also for $B^*(F)$, CTL^+ , $ECTL^+$, and BTL_2), and for $TCTL$ over Timed KSs in [14].

The techniques from [13,14] were not able to cope with $B^*(X)$, or with *Timed* $B(F)$ (the fragment of $TCTL$ where only the F modality may carry timing subscripts). This raises the question of whether these logics have Δ_2^p -hard model checking, and how to prove that. The Δ_2^p upper-bound is indeed too coarse: in the rest of the paper, we prove that model checking $B^*(X)$ is $P^{NP[O(\log^2 n)]}$ -complete, and model checking *Timed* $B(F)$ is $P^{NP[O(\log n)]}$ -complete.

3 Oracle Boolean Circuits and TB(SAT)

We consider special oracle Boolean circuits called *blocks*. As illustrated in Fig. 1, a block is a circuit B computing an output vector z of k bits from a set $y^1, \dots, y^m$ of m input vectors, again with k bits each. Inside the block, p internal gates $x_1, \dots, x_p$ query a SAT oracle: x_i evaluates to 1 iff $F_i(Y, V_i)$ is satisfiable, where

² For CTL^+ and $B^*(F)$, membership in Δ_2^p was observed as early as [2, Theo. 6.2].

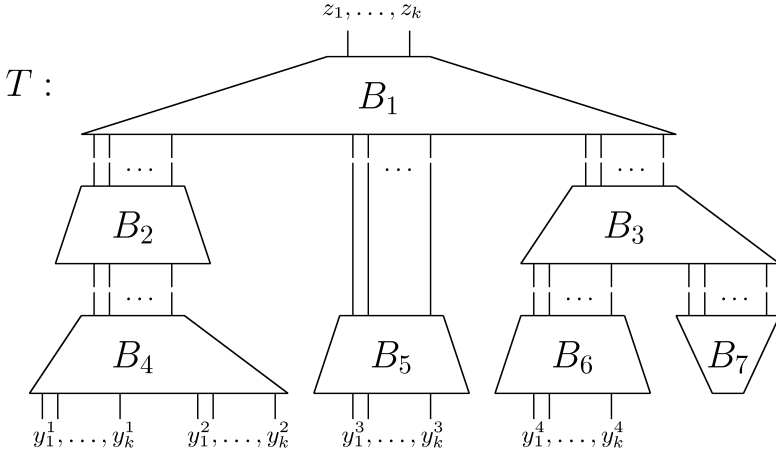


Fig. 2. A “tree of blocks” oracle circuit

F_i is a Boolean formula combining the km input bits $Y = \{y_l^j \mid j = 1, \dots, m, l = 1, \dots, k\}$ with some additional variables from some set V_i . Finally, the values of the output bits are computed from the x_i 's by means of classical Boolean circuits (no oracles): z_i is some $E_i(X)$ where $X = \{x_1, \dots, x_p\}$. We say m is the *degree* of the block, k is its *width*, and its size is the usual number of gates *augmented* by the sizes of the F_i formulae.

The obvious algorithm for computing the value of z for some km input bits is a typical instance of $P_{\parallel}^{\text{NP}}$: the p oracle queries are independent and can be asked in parallel. Building the queries and combining their answers to produce z is a simple polynomial-time computation.

Blocks are used to form more complex circuits: a *tree of blocks* is a circuit T obtained by connecting blocks having a same width k in a tree structure, as illustrated in Fig. 2 (where block B_7 has degree 0).

Every block in a tree has a level defined in the obvious way: in our example, $B_4, \dots, B_7$ are at level 1, B_2, B_3 at level 2, and B_1 , the *root*, at level 3. If the root of some tree is at level d , then the natural way of computing the value of the output z requires d rounds of parallel queries: in our example, the queries inside B_1 can only be formulated after the B_2 queries have been answered, and formulating these require that the B_4 queries have been answered before.

TB(SAT) is the decision problem where one is given a tree of blocks, Boolean values for its input bits, and is asked the value of (one of) the output bits. Compared to the more general problem of evaluating circuits with oracle queries (e.g. the Δ_2^p -complete **DAG(SAT)** problem of [9]), we impose the restriction of a tree-like structure, and compared to the more particular problem of evaluating Boolean trees with oracle queries (the Θ_2^p -complete **TREE(SAT)** problem of [9]),

we allow each node of the tree to transmit a vector of bits to its parent node. Thus **TB**(SAT) is a restriction of **DAG**(SAT) and a generalization of **TREE**(SAT).

Fact 3.1 **TB**(SAT) is Δ_2^P -complete.

3.1 Circuits with Simple Oracle Queries

In a block of width k and degree m , we say a query $\exists V.F(Y, V)?$ has *type* $1 \times M$ if it has the form $\exists l_1, \dots, l_m \exists V'. F(y_{l_1}^1, \dots, y_{l_m}^m, l_1, \dots, l_m, V')?$, i.e. F only uses one bit from each input vector (but it can be any bit and this is existentially quantified upon). Our formulation quantifies upon indexes $l_1, \dots, l_m$ in the $1, \dots, k$ range but such a l_j is a shorthand for e.g. k bits “ $l_j=1$ ”, $\dots$, “ $l_j=k$ ” among which one and only one will be true. These bits are part of V and this is why F depends on $l_1, \dots, l_m$ (and on V' , which is V without the l_j s). There is a similar notion of type $2 \times M$, type $3 \times M$, $\dots$, where F only uses 2 (resp. 3, $\dots$) bits from each input vector.

We say that a query has *type* 1×1 if it has the form $\exists j \exists l \exists V'. F(y_l^j, j, l, V')?$, i.e. F only uses one bit from one input vector (can be any bit from any vector and this is existentially quantified upon). Again, there is a similar notion of type 2×1 , type 3×1 , $\dots$, where we only use 2 (resp. 3, $\dots$) bits in total.

For a query type τ , we let **TB**(SAT) $_\tau$ denote the **TB**(SAT) problem restricted to trees of type τ (i.e. trees where all queries have type τ).

Before we see (in later sections) where such restricted queries appear, we show that they give rise to simpler oracle circuits:

Theorem 3.2. *For any $n > 0$*

1. **TB**(SAT) $_{n \times 1}$ is $P^{NP[O(\log n)]}$ -complete,
2. **TB**(SAT) $_{n \times M}$ is $P^{NP[O(\log^2 n)]}$ -complete.

We prove the upper bounds in the rest of this section. The lower bounds, Corollaries 4.6 and 5.6, are deduced from hardness results for model checking problems studied in the following sections.

3.2 Lowering **TB**(SAT) $_{1 \times M}$ Circuits

Assume block B is the parent of some B' inside a type $1 \times M$ tree T . Fig. 3 illustrates how one can merge B and B' into an equivalent single block B_{new} . Here B' is the leftmost child block of B , so that the input vector $\mathbf{y}^1$ of B will play a special role, but the construction could have been applied with any other child.

The new block copies the u_i query gates and the G_i circuits from B' without modifying them. $2k$ new query gates $x_i^{s,b}$ are introduced for each x_i in B : $x_i^{s,b}$ is like x_i but it assumes $l_1 = s$ and $y_s^1 = b$ in F_i . The x_i query gates from B are replaced by new (non-query) circuits picking the best $x_i^{s,b}$ for which w_s agrees with the assumed value for y_s^1 . The final B_{new} has type $1 \times M$ and degree $m' + m - 1$. $|B_{\text{new}}|$ is $O(|B'| + 2k|B|)$: B was *expanded* but B' is unchanged.

The purpose of this merge operation is to lower the level of trees: we say a tree is *low* if its root has level at most $\log(1 + \text{number of blocks in the tree})$. The tree in Fig. 2 has 7 blocks and root at level 3, so it is (just barely) low.

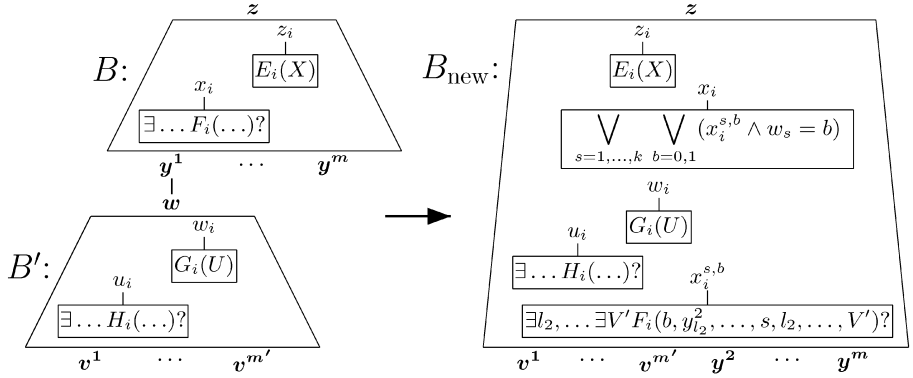


Fig. 3. Merging type $1 \times M$ blocks

Lemma 3.3. *There is a logspace reduction that transforms type $1 \times M$ trees of blocks into equivalent low trees.*

Proof. Consider a type $1 \times M$ tree T . We say a block in T is *bad* if it is at some level $d > 1$ in T and has exactly one child at level $d - 1$ (called its *bad child*). For example B_2 is the only bad node in Fig. 2. If T has bad nodes, we pick a bad B of lowest level and merge it with its bad child. We repeat this until T has no bad node: the final tree T_{new} is low since any non-leaf block at level d must have at least two children at level $d - 1$ hence at least $2^d - 2$ descendants.

Observe that, when we merge a bad B at level d with its bad child B' , the resulting B_{new} has level $d - 1$. Also, since we picked B lowest possible, B' was not bad, so B_{new} cannot be bad or have bad descendants. Thus B_{new} will never be bad again (though it can become a bad child) and will not be expanded a second time. Therefore T_{new} has size $O(k|T|)$ which is $O(|T|^2)$. $\square$

Observe that evaluating a low tree T only requires $O(\log|T|)$ rounds of parallel oracle queries. Therefore Lemma 3.3 provides a reduction from $\mathbf{TB}(\text{SAT})_{1 \times M}$ to $\mathbf{P}_{\parallel O(\log n)}^{\text{NP}}$, a $\mathbf{P}^{\text{NP}[O(\log^2 n)]}$ -complete problem [1].

Corollary 3.4. $\mathbf{TB}(\text{SAT})_{1 \times M}$ is in $\mathbf{P}^{\text{NP}[O(\log^2 n)]}$.

If now n is any fixed number, the obvious adaptation of the merging technique can lower trees of type $n \times M$. Here the new block B_{new} uses $(2k)^n$ new query gates but since n is fixed, the transformation is logspace and the resulting T_{new} has size $O(|T|^{n+1})$.

Corollary 3.5. For any $n \in \mathbb{N}$, $\mathbf{TB}(\text{SAT})_{n \times M}$ is in $\mathbf{P}^{\text{NP}[O(\log^2 n)]}$.

3.3 Flattening $\mathbf{TB}(\text{SAT})_{1 \times 1}$ Circuits

Lemma 3.6. *For any $n \in \mathbb{N}$, there is a logspace reduction that transforms type $n \times 1$ trees of blocks into equivalent blocks.*

Proof (Sketch). With type 1×1 trees, one can merge all children $B_1, \dots, B_m$ with their parent B without incurring any combinatorial explosion. A query gate x_i of the form $\exists j \exists l \exists V'. F_i(y_l^j, j, l, V')?$ will give rise to $2km$ new query gates

$$x_i^{r,s,b} := \exists V'. F_i(b, r, s, V')?$$

where r is the assumed value for j , s the assumed value for l and b the assumed value for y_s^r . x_i will now be computed via

$$x_i := \bigvee_{r=1, \dots, m} \bigvee_{s=1, \dots, k} \bigvee_{b=0,1} (x_i^{r,s,b} \wedge w_s^r = b).$$

We have $|B_{\text{new}}| = O(|B_1| + \dots + |B_m| + 2km|B|)$ so that a bottom-up repetitive application will transform a type 1×1 tree T into a single block of size $O(|T|^3)$.

For type $n \times 1$ trees, the obvious generalization introduces $(2km)^n$ new query gates when merging $B_1, \dots, B_m$ with their parent B , so that a tree T is flattened into a single block of size $O(|T|^{2n+1})$. $\square$

Lemma 3.6 reduces $\mathbf{TB}(\text{SAT})_{n \times 1}$ to $\mathbf{P}_{\parallel}^{\text{NP}}$, a $\mathbf{P}^{\text{NP}[O(\log n)]}$ -complete problem.

Corollary 3.7. *For any $n \in \text{Nat}$, $\mathbf{TB}(\text{SAT})_{n \times 1}$ is in $\mathbf{P}^{\text{NP}[O(\log n)]}$.*

4 Model Checking $B^*(\mathbf{X})$

In this section we show:

Theorem 4.1. *The model checking problem for $B^*(\mathbf{X})$ is $\mathbf{P}^{\text{NP}[O(\log^2 n)]}$ -complete.*

We start by introducing BX^* , a fragment of $B^*(\mathbf{X})$ where all occurrences of $\mathbf{X}$ are immediately over an atomic proposition, or an existential path quantifier (or an other $\mathbf{X}$). Formally, BX^* is given by the following abstract syntax:

$$\varphi ::= \mathbf{E}f(\mathbf{X}^{n_1}\varphi_1, \dots, \mathbf{X}^{n_k}\varphi_k) \mid P_1 \mid P_2 \mid \dots$$

where $f(\dots)$ is any Boolean formula.

Lemma 4.2. *There exists a logspace transformation of $B^*(\mathbf{X})$ formulae into equivalent BX^* formulae.*

Proof (Idea). Bury the $\mathbf{X}$'s using $\mathbf{X}(\varphi \wedge \psi) \equiv \mathbf{X}\varphi \wedge \mathbf{X}\psi$ and $\mathbf{X}(\neg\varphi) \equiv \neg\mathbf{X}\varphi$. $\square$

Lemma 4.3. *There exists a logspace transformation from model checking for BX^* into $\mathbf{TB}(\text{SAT})_{1 \times M}$.*

Proof. With a KS S and a BX^* formula φ we associate a tree of blocks where the width k is the number of states in S , and where there is a block B_ψ for every subformula ψ of φ (so that the structure of the tree mimics the structure of φ). The blocks are built in a way that ensures that the i th output bit of B_ψ is true iff q_i , the i th state in S , satisfies ψ . This only needs type $1 \times M$ blocks.

Assume ψ is some $\exists f(X^{n_1}\psi_1, \dots, X^{n_m}\psi_m)$ with $n_1 \leq n_2 \leq \dots \leq n_m$. Then, for $i = 1, \dots, k$, B_ψ computes whether $q_i \models \psi$ with a query gate x_i defined via

$$x_i := \exists l_1, \dots, l_m \left(f(y_{l_1}^1, \dots, y_{l_m}^m) \wedge \bigwedge_{j=1, \dots, m} \text{Path}(l_{j-1}, n_j - n_{j-1}, l_j) \right) ?$$

where $l_0 = i$, $n_0 = 0$ and $\text{Path}(l, n, l')$ (definition omitted) is a Boolean formula stating that S has an n -steps path from q_l to $q_{l'}$. $\square$

Corollary 4.4. *Model checking for $B^*(X)$ is in $P^{NP}[O(\log^2 n)]$.*

For Theorem 4.1, we need prove the corresponding lower bound:

Proposition 4.5. *Model checking for $B^*(X)$ is $P^{NP}[O(\log^2 n)]$ -hard.*

We have to omit the proof of Proposition 4.5 for lack of space. The complete 3-pages proof can be found in the full version of this paper.

Corollary 4.6. $\text{TB}(\text{SAT})_{1 \times M}$ is $P^{NP}[O(\log^2 n)]$ -hard.

5 Model Checking *Timed* $B(F)$

In this section we show:

Theorem 5.1. *The model checking problem for *Timed* $B(F)$ over *Timed* KSs is $P^{NP}[O(\log n)]$ -complete.*

This is obtained through the next two lemmas.

Lemma 5.2. *Model checking *Timed* $B(F)$ over *Timed* KSs is $P^{NP}[O(\log n)]$ -hard.*

Proof. By reduction from PARITY-SAT. Assume we are given a set $\mathcal{I}_0, \dots, \mathcal{I}_{n-1}$ of SUBSET-SUM instances: we saw in section 2.2 how to associate with these a Kripke structure S and simple *Timed* $B(F)$ formulae $\psi_0, \dots, \psi_{n-1}$ s.t. for every i , $\mathcal{I}_i$ is solvable iff $S \models \psi_i$. Assume w.l.o.g. that n is some power of 2: $n = 2^d$ and for every tuple $\langle b_1, \dots, b_k \rangle$ of $k \leq d$ bits define

$$\varphi_{\langle b_1, \dots, b_k \rangle} \stackrel{\text{def}}{=} \begin{cases} \neg \psi_{\sum_{j=1}^k b_j 2^{j-1}} & \text{if } k = d, \\ (\varphi_{\langle 0, b_1, \dots, b_k \rangle} \wedge \varphi_{\langle 1, b_1, \dots, b_k \rangle}) \vee (\neg \varphi_{\langle 0, b_1, \dots \rangle} \wedge \neg \varphi_{\langle 1, b_1, \dots \rangle}) & \text{otherwise.} \end{cases}$$

$S \models \varphi_{\langle b_1, \dots, b_k \rangle}$ iff there is an even number of solvable $\mathcal{I}_i$ s among those whose index i has $b_1, \dots, b_k$ as last k bits. Therefore the total number of solvable $\mathcal{I}_i$ is even iff $S \models \varphi_{\langle \rangle}$. Since $d = \log n$, $|\varphi_{\langle \rangle}|$ is in $O(n \sum_i |\psi_i|)$ and the reduction is logspace. $\square$

We note that $P^{NP}[O(\log n)]$ -hardness already occurs with a modal depth 1 formula.

Remark 5.3. Observe that this proof applies to all the logics we mentioned in section 2.2: it only requires that several SAT problems $f_1, \dots, f_n$ can be reduced to respective formulae $\psi_1, \dots, \psi_n$ other a same structure S . (This is always possible for logics having a reachability modality like EX or EF). $\square$

Lemma 5.4. *There exists a logspace transformation from model checking for $\text{Timed } B(F)$ over Timed KSs into $\mathbf{TB}(\text{SAT})_{1 \times 1}$.*

Proof (Sketch). We mimic the proof of Lemma 4.3: again we associate a block B_ψ for each subformula and k is the number of states of the Kripke structure S . Assume the edges $e_1, \dots, e_r$ of S carry weights $d_1, \dots, d_r$. Then, for ψ of the form $\text{EF}_{=c}\psi'$, block B_ψ will compute whether $q_i \models \psi$ by asking the query

$$x_i := \exists l \exists n_1, \dots, n_r \left(y_l^1 \wedge c = \sum_{j=1, \dots, r} n_j r_j \wedge \text{Path}'(i, n_1, \dots, n_r, l) \right) ?$$

where $\text{Path}'(i, n_1, \dots, n_r, l)$ (definition omitted) is a Boolean formula checking that there exists a path from q_i to q_l that uses exactly n_j times edge e_j for each $j = 1, \dots, r$ (Euler's circuit theorem makes the check easy). We refer to [14, Lemma 4.5] for more details (e.g. how are the n_i s polynomially bounded?) since here we only want to see that type 1×1 queries are sufficient for $\text{Timed } B(F)$.

$\text{AF}_{=c}\psi$ is dealt with similarly. $\square$

Corollary 5.5. *Model checking $\text{Timed } B(F)$ over Timed KSs is in $\text{P}^{\text{NP}[O(\log n)]}$.*

Corollary 5.6. $\mathbf{TB}(\text{SAT})_{1 \times 1}$ *is $\text{P}^{\text{NP}[O(\log n)]}$ -hard.*

6 Conclusion

We solved the model checking problems for $B^*(X)$ and $\text{Timed } B(F)$, two temporal logic problems where the precise computational complexity was left open.

For $B^*(X)$, the result is especially interesting because of the fundamental nature of this logic, but also because it provides the first example of a natural problem complete for $\text{P}^{\text{NP}[O(\log^2 n)]}$. Indeed, identifying the right complexity class for this problem was part of the difficulty.

Proving membership in $\text{P}^{\text{NP}[O(\log^2 n)]}$ required introducing a new family of oracle circuits. These circuits are characterized by their tree-vector form, and additional special logical conditions on the way an oracle query may depend on its inputs. The tree-vector form faithfully mimics branching-time model checking, while the special logical conditions originate from the modalities that appear in the path formulae. We expect our results on the evaluation of these circuits will be applied to other branching-time logics.

References

1. J. Castro and C. Seara. Complexity classes between Θ_k^p and Δ_k^p . *RAIRO Informatique Théorique et Applications*, 30(2):101–121, 1996.
2. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Programming Languages and Systems*, 8(2):244–263, 1986.
3. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
4. S. Demri and Ph. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1):84–103, 2002.

5. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B, chapter 16, pp 995–1072. Elsevier Science, 1990.
6. E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985.
7. E. A. Emerson and J. Y. Halpern. “Sometimes” and “Not Never” revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.
8. E. A. Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8(3):275–306, 1987.
9. G. Gottlob. NP trees and Carnap’s modal logic. *J. ACM*, 42(2):421–457, 1995.
10. D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. A, chapter 2, pp 67–161. Elsevier Science, 1990.
11. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
12. O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000.
13. F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking CTL^+ and $FCTL$ is hard. In *Proc. 4th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS’2001)*, vol. 2030 of *Lect. Notes Comp. Sci.*, pp 318–331. Springer, 2001.
14. F. Laroussinie, N. Markey, and Ph. Schnoebelen. On model checking durational Kripke structures. In *Proc. 5th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS’2002)*, vol. 2303 of *Lect. Notes in Comp. Sci.*, pp 264–279. Springer, 2002.
15. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
16. Ph. Schnoebelen. The complexity of temporal logic model checking (invited lecture). In *Advances in Modal Logic, papers from 4th Int. Workshop on Advances in Modal Logic (AiML’2002)*. World Scientific, 2003. To appear.
17. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985.
18. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. Logic in Computer Science (LICS’86)*, pp 332–344. IEEE Comp. Soc. Press, 1986.
19. K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theor. Comp. Sci.*, 51(1–2):53–80, 1987.
20. K. W. Wagner. Bounded query classes. *SIAM J. Computing*, 19(5):833–846, 1990.
21. C. B. Wilson. Relativized NC. *Mathematical Systems Theory*, 20(1):13–29, 1987.

There Are Spanning Spiders in Dense Graphs (and We Know How to Find Them)*

Luisa Gargano and Mikael Hammar

Dipartimento di Informatica ed Applicazioni
Università di Salerno, 84081 Baronissi (SA), Italy
fax:+39 089965272, {lg,hammar}@dia.unisa.it

Abstract. A spanning spider for a graph G is a spanning tree T of G with at most one vertex having degree three or more in T . In this paper we give density criteria for the existence of spanning spiders in graphs. We constructively prove the following result: Given a graph G with n vertices, if the degree sum of any independent triple of vertices is at least $n - 1$, then there exists a spanning spider in G . We also study the case of bipartite graphs and give density conditions for the existence of a spanning spider in a bipartite graph. All our proofs are constructive and imply the existence of polynomial time algorithms to construct the spanning spiders. The interest in the existence of spanning spiders originally arises in the realm of multicasting in optical networks. However, the graph theoretical problems discussed here are interesting in their own right.

Keywords: Graph theory, Graph and network algorithms.

1 Introduction

We consider the problem of constructing, for a given graph G , a spanning spider, that is, a spanning tree of G in which at most one vertex has degree larger than 2.

Much work has been devoted to the study of the existence of a Hamilton path in a given graph both from the algorithmic and the graph-theoretic point of view. Deciding if a graph admits a Hamilton path is a well known *NP*-complete problem, even in cubic graphs [11]. On the other hand, if the graph G satisfies any of a number of density conditions, a Hamilton path is guaranteed to exist. Dirac's classical theorem asserts that if G is a graph on n vertices and each vertex of G has degree at least $n/2$, then G has a Hamilton cycle. Dirac's proof also shows that if the sum of the degrees of any pair of independent vertices of G is at least $n - 1$, then G has a Hamilton path [5]. It is also well known that the

* This work is partially supported by the ministero dell'istruzione dell'università e della ricerca: the resource allocation in wireless networks project; and the European Union research training network: approximation and randomized algorithms in communication networks.

above density condition also provides an efficient algorithm to find the Hamilton path (start with any path and extend it by one edge at one of its endpoints; when this process cannot be iterated anymore, one gets the desired Hamilton path).

There are several natural generalizations of the Hamilton path problem. One may want for instance to minimize the maximum degree in a spanning tree of G — when asking for a spanning tree of maximum degree at most k , Dirac's density condition can be generalized to ask that the sum of the degrees of any k pairwise independent vertices is at least $n-1$ [17].

Another direction for generalizing the Hamilton path problem was considered in [7]: find a spanning tree T of a given graph G having the minimum possible number of branch vertices, where a branch vertex is a vertex of degree larger than two in T . The above minimum is zero if and only if the graph G has a Hamilton path. The interest in minimizing the number of branch vertices arises from a problem in optical networks [14,16]; it is motivated by an efficient use of new technologies (e.g., light splitting devices) in the realm of multicasting in optical networks; the interested reader is referred to [7,18].

Several algorithmic and graph-theoretic questions were studied in [7] concerning the construction of spanning trees with few branch vertices. In particular, density conditions that are sufficient to give upper bounds on the minimum number of branch vertices were studied; such conditions add to a degree bound the assumption that the graph is claw-free (e.g., it does not contain an induced $K_{1,3}$ (or $K_{1,4}$) subgraph). No non-trivial density condition is known to be sufficient without any additional assumption on the graph. The following conjecture was made in [7].

Conjecture 1. [7] Let G be a connected graph and k a nonnegative integer. If each vertex of G has degree at least $\frac{n-1}{k+2}$ (or more generally, if the sum of the degrees of any $k+2$ independent vertices is at least $n-1$) then there exists a spanning tree in G with at most k vertices of degree higher than 2.

For $k=0$ the conjecture is true, being Dirac's condition for the existence of a Hamilton path.

A tree with at most one branch vertex is called a *spider*. Here we are interested in the existence (and construction) for a given graph G of a spanning spider — e.g. the case $k=1$ in the above conjecture. We notice that the problem of deciding whether a given graph admits a spanning spider is computationally intractable in general.

Theorem 1. [7] *It is NP-complete to decide whether a graph G admits a spanning spider.*

1.1 Our Results

In this paper we study the problem of existence (and construction) of spanning spiders both in general and in bipartite graphs.

In case of general graphs we show that any graph G on n vertices in which the sum of the degrees of any three pairwise independent vertices is at least $n - 1$ admits a spanning spider and this spider can be efficiently found. Namely, we prove the following Theorem.

Theorem 2. *Let G be a connected graph in which the sum of the degrees of any three independent vertices is at least $n - 1$. Then G contains a spanning spider. Furthermore, there is an $O(n^3)$ time algorithm to find a spanning spider in G .*

We also consider the case of bipartite graphs. It is well known that the degree bound for the existence of a Hamilton path can be improved when considering bipartite graphs [2]. The same holds for the existence of spanning spiders.

Theorem 3. *Let $G = (U, V, E)$ be a connected bipartite graph with $|U| \leq |V|$ such that for all $u \in U$ and $v \in V$, it holds that*

$$d(u) + d(v) \geq |U| \text{ and } d(v) \geq \frac{|V||U|}{|V| + |U|}.$$

Then G contains a spanning spider. Furthermore, there is an $O(n^3)$ time algorithm to find a spanning spider in G .

The stronger density condition given in the following theorem assures, for a bipartite graph $G = (U, V, E)$ with $|U| \leq |V|$, the existence of a spanning spider centered in u , for each vertex $u \in U$. Notice that if $|V| \geq |U| + 2$ then G cannot contain a spanning spider centred at a vertex $v \in V$, even if G is the complete bipartite graph $K_{|U|, |V|}$.

Theorem 4. *Let $G = (U, V, E)$ be a connected bipartite graph with $|U| \leq |V|$ such that for all $u \in U$ and $v \in V$, it holds that $d(u) + d(v) \geq |V|$. Then G contains a spanning spider centred at any vertex $u \in U$. Furthermore, there is an $O(n^3)$ time algorithm to find a spanning spider in G .*

A basic tool in the construction of the desired spanning spiders is the construction of a sufficiently long path in the given graph. In Section 3 we give a local optimisation heuristic to find such paths. We define a set of *maximality criteria* and show how to find paths satisfying these criteria.

For the case of general graphs, our paths actually are Hamilton paths if the graph satisfies the density criterion of Dirac [5]. Indeed, our maximality criterion includes the simple one used in the original proof by Dirac (in that case a path is called maximal if it cannot be extended by adding one new vertex at one of its endpoints).

We also give a density criterion for the existence and construction of long paths in bipartite graphs — a generalization of our interest in the construction of paths that contain all vertices from the smaller partition of the vertex set. This criterion is a bit stronger than what is given by the more general theorem by Jackson [10], and we show a simple and efficient algorithm to generate such paths. In particular we prove the following result

Lemma 1. *Given a bipartite graph $G = (U, V, E)$, with $|V| \geq |U|$. If $d(u) + d(v) \geq \delta$ for any $u \in U$ and $v \in V$, then we can find, in time $O(n^3)$, a path in G that either spans all vertices in U or has size at least 2δ .*

1.2 Summary of the Paper

In Section 2 we state the notation used in the rest of the paper. In Section 3 we define maximal paths and show how to construct them. In Section 4 we show how to turn a maximal path into a spanning spider, in any graph satisfying the density condition of Theorem 2. In Section 5, we consider the construction of maximal paths and spanning spiders in bipartite graphs. In Section 6 we conclude and state some open problems. Please note that some proofs are omitted due to space limitations.

2 Notation

Let $G = (V, E)$ denote a connected graph on n vertices (in the rest of the paper any graph should be intended as a connected graph and we will reserve n to denote the number of vertices).

For a vertex $v \in V$ we let $d(v)$ denote the degree of v . For a subset $X \subseteq V$ we define $d_X(v)$ to be the number of vertices in X that are adjacent to v .

We use $\delta(G) = \min_{v \in V} d(v)$ to denote the minimum degree of any vertex in G , and let $\delta_k(G)$ denote the minimum degree sum of any k pairwise independent vertices in G .

The neighborhood in G of a vertex x is denoted by $N(x)$, for a subset $X \subseteq V$ we define the neighbourhood of $v \in V$ with respect to X as

$$N_X(v) = \{u \in X \mid (u, v) \in E\}.$$

For sake of simplicity, whenever it is clear from the context, we will identify the vertex set of a (sub)graph H of G with H itself. Hence, we will use $|H|$ to indicate the number of vertices in the graph and $d_H(v)$ and $N_H(v)$ will represent, respectively, the degree and the neighborhood of v with respect to the vertex set of H .

Let $P = [v_0 v_1 \dots v_t]$ denote a path in G . The *left neighbourhood* of $x \in V$ on P is the set

$$N_P^-(x) = \{v_i \mid (v_{i+1}, x) \in E\}.$$

The *right neighbourhood* of $x \in V$ on P is defined analogously as

$$N_P^+(x) = \{v_i \mid (v_{i-1}, x) \in E\}.$$

When the underlying path is evident from the context we write $N^-(x)$ and $N^+(x)$ for the left and right neighbourhoods, respectively.

Any left neighbour $v_i \in N^-(v_0)$ of v_0 is the end point of the path $P - (v_i, v_{i+1}) + (v_0, v_{i+1})$ containing the same set of vertices as P ; by symmetry, the same holds for $N^+(v_t)$; see Figure 1. Therefore, we say that the elements in $N^-(v_0)$ and $N^+(v_t)$ are *potential endpoints* with respect to P .

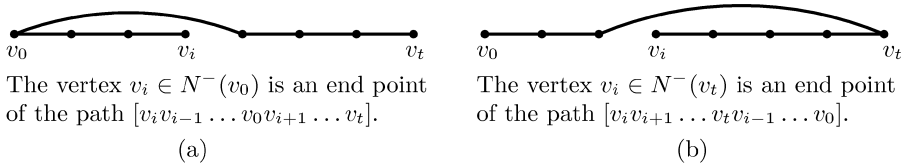


Fig. 1. Potential end points in a path.

3 Maximal Paths in General Graphs

In order to construct a spanning spider in a dense graph, we first find a suitable long path in the graph. This path will then be turned into a spider that in a last step can be extended to span the whole graph. This section is devoted to finding the desired long paths in dense general graphs.

The following set of *maximality criteria* implicitly suggest a local optimisation heuristic to find suitably long paths in general dense graphs. We obtain this heuristic by showing how to find paths that satisfy the criteria.

Definition 1. A path $P = [v_0 \dots v_t]$ is called maximal if either it is a Hamilton path or it satisfies each of the following conditions:

- i) $N(r) \cap N^-(v_0) = \emptyset = N(r) \cap N^+(v_t)$, for every $r \in V - P$.
- ii) $N(v_0) \cap N^+(v_t) = \emptyset$.
- iii) $N^-(v_0)$ is an independent set, for every $r \in V - P$.
- iv) If $N^-(v_0) \cap N^+(v_t) \neq \emptyset$ then
 - (a) no two consecutive vertices in P both have neighbours in $V - P$,
 - (b) $V - P$ is an independent set.

We show now that any non-maximal path $P = [v_0 \dots v_t]$ can be extended in polynomial time.

If condition i) is violated then there is a vertex r outside P that is adjacent to a potential end point of a path P' . Thus, we construct P' (if r is adjacent to v_0 or v_t then $P' = P$) as described in Figure 1 and add r to this path.

If condition ii) is violated then we can find a cycle in G that contains all the vertices of P ; see Figure 2. Since G is connected and P is not a Hamilton path there is a vertex r outside P that is adjacent to a vertex v in P . Thus, we can

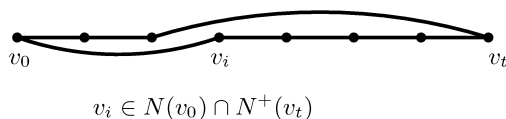


Fig. 2. If $N(v_0) \cap N^+(v_t) \neq \emptyset$ then there is a cycle in G that contains all vertices in P .

extend P by constructing the path P' obtained by adding (r, v) to the cycle and removing any other edge incident to v .

If condition **iii)** is violated we find an edge between two vertices in $N^-(r)$ and extend P as described in Figure 3.

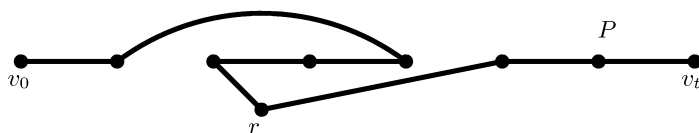


Fig. 3. If the left neighbourhood on P of a vertex r outside P is not an independent set then P can be extended to include r .

If condition **iv)** is violated then we have two cases to consider: either there are two consecutive vertices on P that are both adjacent to vertices in the subgraph $G-P$, or $V-P$ is not an independent set.

In the first case we identify the two vertices v_i and v_{i+1} that are both adjacent to vertices outside P . If they are both adjacent to the same vertex $r \in V-P$ then we directly add this new vertex to P obtaining the longer path $[v_0 \dots v_i r v_{i+1} \dots v_t]$. If they are adjacent to different vertices in $V-P$, we construct the cycle C containing all but one vertex of P as described in Figure 4. Let v denote the excluded vertex. Note that $(v_i, v_{i+1}) \in E(C)$ (otherwise either $v_i = v$ or $v_{i+1} = v$; but $v \in N^-(v_0)$ and such a vertex is not adjacent to vertices in $V-P$, by condition **i)**). Assume that $(v_i, r_1) \in E$ and $(v_{i+1}, r_2) \in E$, where $r_1, r_2 \notin P$. By removing (v_i, v_{i+1}) from C and adding (r_1, v_i) and (v_{i+1}, r_2) to C , we create a new path of size $|P|+1$, with endpoints r_1 and r_2 .

For the second case, we observe that if $V-P$ is not an independent set then we can construct a path P' in $G-P$ containing at least two vertices, which by the connectivity of G can be connected to the cycle C described above. In this way we create a new path with size at least $|P|+1$.

A careful analysis of the violation checks above shows that an algorithm to find a maximal path can be implemented to run in $O(n^3)$ time.

Theorem 5. *A maximal path in a connected graph can be found in $O(n^3)$ time.*

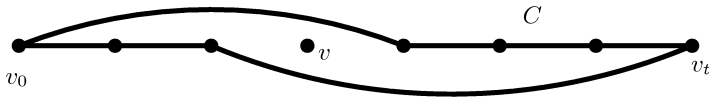


Fig. 4. The cycle C includes all vertices of the path $[v_0 \dots v_t]$ except $v \in N^-(v_0) \cap N^+(v_t)$.

4 Spanning Spiders in General Graphs

In this section we give an algorithm to find spanning spiders in dense graphs, where dense in our case means graphs G for which $\delta_3(G) \geq n - 1$. We base our algorithm on the fact that we can compute maximal paths, as defined in Section 3, in $O(n^3)$ time. The given algorithm will prove Theorem 2.

Let P denote a maximal path in G according to Definition 1, with $P = [v_0 v_1 \dots v_{t-1} v_t]$ and let $R = V - P$ denote the vertices of G outside P .

Recall that Definition 1 includes two additional conditions if the set $N^-(v_0) \cap N^+(v_t)$ is non-empty. We start considering the other case, i.e., $N^-(v_0) \cap N^+(v_t) = \emptyset$.

Lemma 2. *If P is maximal then either $N^-(v_0) \cap N^+(v_t) \neq \emptyset$ or there is a spanning spider in G whose centre is adjacent to all vertices outside P .*

Assume from now on that

$$N^-(v_0) \cap N^+(v_t) \neq \emptyset.$$

This implies that condition **iv (a)** and **iv (b)** of Definition 1 hold.

We will give an algorithm proving the following weaker theorem. Later we will extend it to the general case considered in Theorem 2.

Theorem 6. *Any connected graph G with $\delta(G) \geq (n - 1)/3$ contains a spanning spider. Furthermore, there is an $O(n^3)$ time algorithm to find a spanning spider in G .*

The following lemma gives Theorem 6 when the size of R is small.

Lemma 3. *If $|R| \leq 2$ then G contains a spanning spider.*

Assume now that $|R| \geq 3$, with $R = \{r_1, r_2, \dots, r_{|R|}\}$, and let r^* denote an arbitrary vertex in R . In order to prove Theorem 6, we construct a spanning spider out of the maximal path P . First we need to find a suitable centre for the spider. It turns out that a convenient property of such a centre is to be adjacent to many independent vertices which in turn are independent of R .

Lemma 4. *The set $N^-(r^*) \cup R$ is independent, with size $|R| + (n - 1)/3$. Furthermore, there exists a vertex $v_i \in P - N^-(r^*)$ whose number of neighbours in $N^-(r^*) \cup R$ is at least*

$$\frac{(n - 1)}{6} + \frac{3|R| - 1}{4}.$$

Proof. The independence is given by Definition 1 as follows. If $r \in R$ and $v \in N^-(r^*)$ then $(r, v) \notin E$ by condition **iv**) point **(a)**. R is an independent set by condition **iv**) point **(b)**. Left is to prove that $N^-(r^*)$ is independent, but this follows from condition **iii**). The size of the union follows from the degree condition on r^* , and the fact that R and $N^-(r^*)$ are disjoint.

For the second part of the proof, consider the vertices in $N^-(r^*) \cup R$. Each of them is adjacent only to vertices in $P - N^-(r^*)$, since $N^-(r^*) \cup R$ is an independent set and $R \cap P = \emptyset$. By the pigeonhole principle there exists a vertex $v_i \in P - N^-(r^*)$ adjacent to at least

$$\frac{\binom{n-1}{3} |N^-(r^*) \cup R|}{|P - N^-(r^*)|} = \frac{\frac{n-1}{3} \left(\frac{n-1}{3} + |R| \right)}{n - \frac{n-1}{3} - |R|} = \frac{\frac{n-1}{3} \left(\frac{n-1}{3} - \frac{|R|-1}{2} + \frac{3|R|-1}{2} \right)}{2 \left(\frac{n-1}{3} - \frac{|R|-1}{2} \right)} \geq \frac{n-1}{6} + \frac{3|R|-1}{4}$$

vertices in $N^-(r^*) \cup R$. □

Let v_i be a vertex in $P - N^-(r^*)$ satisfying the condition given in Lemma 4.¹ Let Δ be the number of vertices in $N^-(r^*) \cup R$ adjacent to v_i , i.e.,

$$\Delta \geq \frac{n-1}{6} + \frac{3|R|-1}{4}. \quad (1)$$

Using the algorithm in Table 1 we construct a spider S , centred at v_i , with branches beginning at vertices in $N^-(r^*)$ and ending at vertices in $N(r^*)$. Note that S fails to include the tail of P . We let T denote this tail; see Figure 5.

Table 1. The spider construction algorithm for general graphs.

Algorithm Spider construction in general graphs. Input: A graph $G = (V, E)$, a maximal path P, and a vertex v_i satisfying the condition of Lemma 4. Output: A spider S, centred at v_i, and a tail T, that collectively span P and a portion of R. 1 Initially let $S := P$. 2 For each $r \in R$ such that $(v_i, r) \in E$: add the edge (v_i, r) to S. 3 If all $r \in R$ are adjacent to v_i: return the spanning spider S. Otherwise, 4 For each $v_j \in P$ such that both (v_{j-1}, v_i) and (v_j, r^*) are in E: remove (v_{j-1}, v_j) from S, and add the edge (v_i, v_{j-1}) to S. 5 If there is an edge $(v_i, v_j) \in S$ with $j > i + 1$: remove the edge (v_i, v_{i+1}) from S (recall that v_i is the centre of the spider). 6 Return the spider S and the tail $T := P - S$. End

Let L denote the leaves in S and let $R' = S - P - r^* \subset R$. We note that the number of leaves in S is at least $\Delta + 1$ but more importantly, the number of leaves adjacent to r^* is

$$d_L(r^*) \geq \Delta - |R'| - 2. \quad (2)$$

¹ Notice that $i < t$, since by condition **i**) of Definition 1, $(N^-(r^*) \cup R) \cap N(v_t) = \emptyset$.

To see this, note first of all that r^* is not adjacent to any leaf that belongs to R' . Secondly, the tail T is not in S , but contains exactly one vertex in $N(v_i)$ that also lies in $N^-(r^*)$. Finally, if v_i is adjacent to r^* , then r^* is itself a leaf in S , but is of course not adjacent to itself.

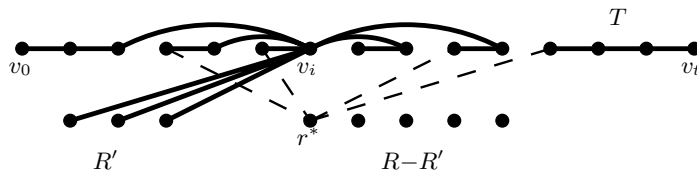


Fig. 5. The spider S , the tail T and the set $R - R'$, after the spider construction algorithm.

If there is a matching between the vertices in $R - R'$ and $L - R'$, then we can construct a spider covering G . Next we prove that there is such a matching. A vertex v in $S \cup T$ is called an internal vertex if $v \notin L$. We let I denote the set of internal vertices.

Lemma 5. *There exists a matching between $R - R'$ and $L - R'$.*

Proof. Since r^* is adjacent to more than $|R - R'|$ leaves in S , it suffices to show that there is a matching between $R - R' - \{r^*\}$ and $L - R'$. Let r denote an arbitrary vertex in $R - R' - \{r^*\}$. By definition,

$$d(r) = d_I(r) + d_L(r). \quad (3)$$

Since r is not adjacent to v_i , and v_{i+1} is a leaf by construction, neither v_i nor v_{i+1} is counted in $d_I(r)$. Neither are they counted in $d_L(r^*)$. This time, v_i is not counted, since it is not a leaf, and v_{i+1} is not counted because $v_i \notin N^-(r^*)$.

Therefore,

$$d_I(r) + d_L(r^*) \leq (P - 2)/2,$$

since r and r^* cannot be adjacent to v_0 or v_t (Definition 1, condition **ii**)), nor to consecutive vertices on P (Definition 1, point **(a)** of condition **iv**)). Hence,

$$d_L(r) \geq d(r) + d_L(r^*) - (P - 2)/2. \quad (4)$$

Recalling that $d_L(r^*) \geq \Delta - |R'| - 2$ (by (2)) and that $|P| = n - |R|$, by using (4) we get

$$d_L(r) \geq \frac{n-1}{3} + (\Delta - |R'| - 2) - \frac{n - |R| - 2}{2}. \quad (5)$$

By using (1) we obtain

$$d_L(r) \geq \frac{n-1}{3} + \frac{n-1}{6} + \frac{3|R| - 1}{4} - |R'| - 2 - \frac{n - |R| - 2}{2}$$

$$= |R| - |R'| + (|R| - 7)/4$$

$$\geq |R - R'| - 1.$$

the last inequality holds since $|R| \geq 3$ by Lemma 3. Thus, each vertex in $R - R' - \{r^*\}$ is adjacent to at least $|R - R'| - 1$ leaves in S , so there exists a matching between $R - R' - \{r^*\}$ and $L - R'$. $\square$

Given the above guarantee of a matching we construct the spider as follows. Compute a matching between $R - R'$ and L . This gives us a new spider S' that contains all vertices except the tail T . The head of the tail is adjacent to r^* , and r^* is a leaf in S' . Add to S' the edge between r^* and the head of the tail to complete the spanning spider. This concludes the proof of Theorem 6.

Our main theorem follows easily from previous discussion.

Proof of Theorem 2 (sketch). We begin with the following observation. In any independent set, there can be at most two vertices with degree less than $(n - 1)/3$. This follows directly from the degree sum criteria. Thus, in the set R there are at most two vertices with degree less than $(n - 1)/3$, call these r and r' . It is easy to modify any maximal path so to contain the eventual low degree vertices, i.e., every vertex in R has at least $(n - 1)/3$ neighbours. $\square$

5 Bipartite Graphs

In this section we consider bipartite graphs. As in the case of general graph, our spanning spider construction starts with the construction of a suitable long path.

5.1 Maximal Paths in Bipartite Graphs

In case of bipartite graphs the construction of a maximal path can be specialized to get Lemma 1, that is, to efficiently construct a path of size at least $2 \min\{|U|, \delta\}$ in any bipartite graph $G = (U, V, E)$, with $|V| \geq |U|$ and $d(u) + d(v) \geq \delta$ for any $u \in U$ and $v \in V$.

The following special case will be used for the construction of spanning spiders in bipartite graphs.

Corollary 1. *Given a bipartite graph $G = (U, V, E)$, with $|V| \geq |U|$. If for any $u \in U$ and $v \in V$, $d(u) + d(v) \geq |U|$ then we can find, in time $O(n^3)$, a path in G that includes all vertices in U .*

Proof of Lemma 1. We first define a bipartite maximal path and then show that any such path has the desired property.

A path $P = [u_0 v_0 \dots u_t v_t]$ in $G = (U, V, E)$ is called bipartite maximal if $|P| = 2|U|$ or it satisfies the following two conditions.

- 1) For any $u \in U$ and $v \in V$ the path P cannot be extended as either of the following

$$[uvu_0v_0 \dots u_tv_t], \quad [vu_0v_0 \dots u_tv_tu], \quad [u_0v_0 \dots u_tv_tuv].$$

- 2) $N(u_0) \cap N^+(v_t) = \emptyset$.

It is possible to show that any bipartite maximal path has the desired length. To this aim, we show that any non maximal path P of size

$$|P| < 2 \min\{|U|, \delta\} \quad (6)$$

can be extended to a path of size $|P| + 2$. This is obvious if condition **1**) does not hold. Consider then condition **2**) and let $v_i \in N(u_0) \cap N^+(v_t)$. Consider the cycle $P - \{(u_i, v_i)\} + \{(u_0, v_i), (u_i, v_t)\}$. W.l.o.g. denote it by

$$C = [u'_0v'_0 \dots u'_tv'_t].$$

We show that it is possible to obtain a path of size $|C| + 2 = |P| + 2$ from C . Let U' and V' denote the set of vertices outside C in U and V , respectively, i.e.,

$$U' = U - \{u'_0, \dots, u'_t\}, \text{ and } V' = V - \{v'_0, \dots, v'_t\}.$$

Since $|C| = |P| < 2|U|$ then $U' \neq \emptyset$.

Let us first assume that $R = U' \cup V'$ form an independent set. Let $u \in U'$ and $v \in V'$. If u and v are neighbours of two vertices that are adjacent on C then we can get a path of size $|C| + 2$ including all vertices in C together with u and v (see Figure 6).

Otherwise, for each pair u'_i, v'_i of adjacent vertices in C it cannot hold that both $(u'_i, v) \in E$ and $(u, v'_i) \in E$. This implies that $d(u) + d(v) \leq |C|/2$, that is

$$|C| \geq 2(d(u) + d(v)) \geq 2\delta \geq 2 \min\{|U|, \delta\}$$

contradicting (6).

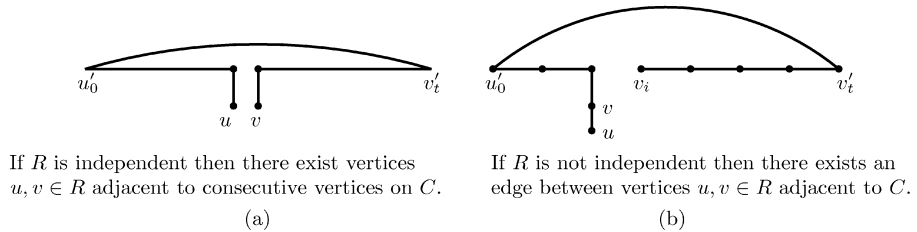


Fig. 6. Extending paths in the bipartite case.

Suppose now that there exists an edge (u, v) between two vertices in R . Since G is connected, vertices u and v must be connected to C . This immediately implies that a path of at least 2 vertices in R can be added to C , thus giving a path of size at least $|C| + 2$. $\square$

5.2 Spanning Spiders in Bipartite Graphs

In this section we prove Theorem 3 and Theorem 4. We show how to construct the desired spanning spider starting from a bipartite maximal path as defined in Section 5.1.

Corollary 1 assures that if for each $u \in U$ and $v \in V$ it holds that $d(u)+d(v) \geq |U|$ then a bipartite maximal path includes all vertices of U (notice that this condition is satisfied by the hypothesis of both Theorem 4 and Theorem 3). Let $P = [u_0v_0 \dots u_{|U|-1}v_{|U|-1}]$ be such a path. Given a vertex $u_j \in U$, define the sets

$$R = V - P \text{ and } R' = \{v \in R \mid (u_j, v) \in E\}. \quad (7)$$

We construct the spider S centred at u_j using the algorithm stated in Table 2.

Table 2. The spider construction algorithm for bipartite graphs.

Algorithm	Spider construction in bipartite graphs.
Input:	A bipartite graph $G = (U, V, E)$, a maximal path P , and a vertex $u_j \in U$.
Output:	A spider S , centred at u_j , and spanning P and R' .
1	Initially let $S := P$.
2	For each $v_i \in N_P(u_j)$ with $0 \leq i \leq U - 1$, $i \neq j - 1$, $i \neq j$: insert the edge (u_j, v) in S .
3	For each $v_i \in N_P(u_j)$ with $0 \leq i \leq U - 1$, $i \neq j - 1$, $i \neq j$: insert the edge (u_j, v_j) in S , remove from S the edge $\begin{cases} (u_i, v_i) & \text{if } i \geq j + 1, \\ (v_i, u_{i+1}) & \text{if } i \leq j - 2. \end{cases}$
End	

This gives a spider S centred at u_j (see Figure 7) spanning all vertices in the path P and the set R' .

Let L and I denote, respectively, the leaves and the internal vertices of S that also belong to U . In order to obtain a spanning spider, one needs to include in S the vertices in $R - R'$. This will be done by finding a matching between the vertices in $R - R'$ and L .

Since the spider is centred in $u_j \in U$, it follows that all its leaves except $v_{|U|-1}$ and the elements in R' belong to U , i.e., $|L| = d(u_j) - 1 - |R'|$. Hence, the number of internal vertices of S that belong to U is

$$|I| = |U - L| = |U| - |L| = |U| - d(u_j) + |R'| + 1. \quad (8)$$

For each vertex $r \in R - R'$, we count the number of leaves in S that are adjacent to r , i.e., $d_L(r)$. Since, by definition, u_j is not adjacent to r , by (8)

$$d_L(r) = d(r) - d_I(r) \geq d(r) - (|I| - 1) = d(r) - |U| + d(u_j) - |R'|. \quad (9)$$

We need now to differentiate our reasoning in order to prove Theorem 3 and Theorem 4.

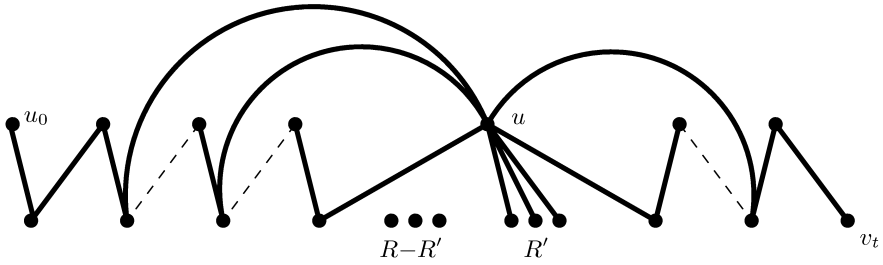


Fig. 7. Constructing a spider for a bipartite graph.

Proof of Theorem 3. Recall that we want to prove the existence of at least one spanning spider under the density criteria

$$(a) \ d(u) + d(v) \geq |U|, \quad (b) \ d(v) \geq \alpha|U|, \text{ where } \alpha = \frac{|V|}{|V|+|U|}.$$

In order to conclude the proof of the theorem we need to choose a suitable center u_j for the above defined spider S . We choose u_j as the vertex of highest degree in U . By the pigeonhole principle

$$d(u_j) \geq \frac{\sum_{v \in V} d(v)}{|U|} \geq \frac{|V|(\alpha|U|)}{|U|} = \alpha|V|. \quad (10)$$

Take an arbitrary vertex $r \in R - R'$ (cfr. (7)). From (9) and (10) one has

$$d_L(r) \geq d(r) + d(u_j) - |R'| - |U| \geq d(r) + \alpha|V| - |R'| - |U|.$$

Using the density criteria (b), we get

$$d_L(r) \geq \alpha|U| + \alpha|V| - |R'| - |U|.$$

Recalling that $\alpha = \frac{|V|}{|V|+|U|}$, and the equality $|R| = |V| - |U|$, we obtain

$$d_L(r) \geq \alpha(|U| + |V|) - |U| - |R'| = |V| - |U| - |R'| = |R| - |R'|.$$

By Hall's Theorem [9] there is a matching between $R - R'$ and the leaves of S and we can create a spanning spider by adding the vertices in $R - R'$ to the initial spider S centered in u_j . $\square$

Proof of Theorem 4. Recall that here u_j can be any vertex in U and that we are assuming that $d(u) + d(v) \geq |V|$, for any $u \in U$ and $v \in V$.

Using the fact that $d(r) + d(u_j) \geq |V|$, by (9) we get

$$d_L(r) \geq |V| - |U| - |R'| = |R - R'|.$$

By Hall's Theorem [9] there is a matching between the leaves of S and $R - R'$. Using this matching we create a spanning spider by adding the vertices in $R - R'$ to S . $\square$

6 Conclusions and Open Problems

We have considered the problem of constructing, for a given graph G , a spanning spider, that is, a spanning tree of G in which at most one vertex has degree larger than 2. We have considered both general and bipartite graphs. In particular, in the case of general graphs we have proved that given a graph G with n vertices, if the degree sum of any independent triple of vertices is at least $n - 1$, then there exists a spanning spider in G , thus proving a conjecture in [7].

The interest in the existence of spanning trees with limited number of vertices of degree larger than 2 (and of spanning spiders in particular) originally arises in the realm of multicasting in optical networks. However, the related algorithmic and graph theoretical problems are interesting in their own right.

The first obvious open question is whether Conjecture 1 holds for $k \geq 2$. Moreover, it would be interesting to extend the result presented in this paper in order to have non trivial density conditions for a graph to admit a spider covering at least a given fraction of its vertices.

References

1. C. Bazgan, M. Shanta, Z. Tuza, "On the Approximability of Finding A(nother) Hamiltonian Cycle in Cubic Hamiltonian Graphs", *Proc. 15th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS, Vol. 1373, 276–286, Springer, (1998).
2. C. Berge. Graphs and Hypergraphs. North-Holland Publishing Company – Amsterdam and London, 1973.
3. J. A. Bondy. Properties of graphs with constraints on degrees. *Studia Sc. Math. Hung.*, 4:473–475, 1969.
4. V. Chvatal. On hamilton ideal's. *J. Comb. Theory*, 12 B:163–168, 1972.
5. G. A. Dirac. Some theorems on abstract graphs. *Proc. London Mathematical Society*, 2:69–81, 1952.
6. T. Feder, R. Motwani, C. Subi, "Finding Long Paths and Cycles in Sparse Hamiltonian Graphs", *Proc. Thirty second annual ACM Symposium on Theory of Computing (STOC'00)* Portland, Oregon, May 21–23, 524–529, ACM Press, 2000.
7. L. Gargano, P. Hell, L. Stacho, and U. Vaccaro. Spanning trees with bounded number of branch vertices. In *29-th International Colloquium on Automata, Languages, and Programming*, pages 355–365, 2002.
8. R. J. Gould. Updating the Hamiltonian problem—a survey. *J. Graph Theory*, 15(2):121–157, 1991.
9. P. Hall. On representatives of subsets. *Journal of London Mathematical Society*, pages 26–30, 1935.
10. B. Jackson. Long cycles in bipartite graphs. *Journal of Combinatorial Theory*, 38 B:118–131, 1985.
11. R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Applications*, pages 85–103. Plenum Press, New York, 1972.
12. S. Khuller, B. Raghavachari, N. Young, "Low degree spanning trees of small weight", *SIAM J. Comp.*, 25 (1996), 335–368.
13. J. Könemann, R. Ravi, "A Matter of Degree: Improved Approximation Algorithms for Degree-Bounded Minimum Spanning Trees", *Proc. Thirty second annual ACM Symp. on Theory of Computing (STOC'00)*, Portland, Oregon, 537–546, (2000).

14. B. Mukherjee, *Optical Communication Networks*, McGraw-Hill, New York, 1997.
15. L. Posa. A theorem concerning hamilton lines. *Magyar Tud. Akad. Mat. Kutato Int. Közl.*, 7:225–226, 1962.
16. T. E. Sterne, and K. Bala, *MultiWavelength Optical Networks*, Addison-Wesley, (1999).
17. S. Win, “Existenz von Gerüsten mit vorgeschriebenem Maximalgrad in Graphen”, *Abh. Mat. Sem. Univ. Hamburg*, 43: 263–267, 1975.
18. X. Zhang, J. Wei, and C. Qiao Constrained Multicast Routing in WDM Networks with Sparse Light Splitting *Proc. of IEEE INFOCOM 2000*, vol. 3: 1781–1790, Mar. 2000.

The Computational Complexity of the Role Assignment Problem

Jiří Fiala^{1*} and Daniël Paulusma^{2**}

¹ Charles University, Faculty of Mathematics and Physics,
DIMATIA and Institute for Theoretical Computer Science (ITI)^{***},
Malostranské nám. 2/25, 118 00, Prague, Czech Republic.

`fiala@kam.mff.cuni.cz`

² University of Twente,
Faculty of Electrical Engineering, Mathematics and Computer Science,
Department of Applied Mathematics,
P.O. Box 217, 7500 AE Enschede, The Netherlands,
Phone: + 31 53 489 3421,
Fax: + 31 53 489 4858.
`d.paulusma@math.utwente.nl`

Abstract. A graph G is R -role assignable if there is a locally surjective homomorphism from G to R , i.e. a vertex mapping $r : V_G \rightarrow V_R$, such that the neighborhood relation is preserved: $r(N_G(u)) = N_R(r(u))$. Kristiansen and Telle conjectured that the decision problem whether such a mapping exists is an NP-complete problem for any connected graph R on at least three vertices. In this paper we prove this conjecture, i.e. we give a complete complexity classification of the role assignment problem for connected graphs. We show further corollaries for disconnected graphs and related problems.

Keywords: computational complexity, graph homomorphism, role assignment

2002 Mathematics Subject Classification: 05C15, 03D15.

1 Introduction

Given two graphs, say G and R , an R -role assignment for G is a vertex mapping $r : V_G \rightarrow V_R$, such that the neighborhood relation is maintained, i.e. all roles of the image of a vertex appear on the vertex's neighborhood. Such a condition can be formally expressed as

$$\text{for all } u \in V_G : r(N_G(u)) = N_R(r(u)),$$

where $N(u)$ denotes the set of neighbors of u in the corresponding graph.

* This author was partially supported by research grant GAUK 158/99.

** This author was partially supported by NWO grant R 61-507 and by Czech research grant GAČR 201/99/0242 during his stay at DIMATIA center in Prague.

*** Supported by the Ministry of Education of the Czech Republic as project LN00A056.

Such assignments have been introduced by Everett and Borgatti [6], who called them role colorings. They originated in the theory of social behavior. The graph R , i.e. the *role graph*, models roles and their relationships, and for a given society we can ask whether its individuals can be assigned roles such that the relationships are preserved: Each person playing a particular role has among its neighbors exactly all necessary roles as they are prescribed by the model.

From the computational complexity point of view it is interesting to know whether it is possible to decide quickly (i.e. in polynomial time) whether such assignment exists. This problem was considered by Roberts and Sheng [15], who focus on a more generalized problem called the 2-role assignment problem. If both graphs G and R are part of the input, the problem is NP-complete already for $R = K_3$ [12].

In order to make a more precise study we consider a class of R -role assignment problems, $RA(R)$, parameterized by the role graph R . Here the instance is formed only by the graph G , and we ask whether an R -role assignment of G exists.

The complexity study of this class of problems is closely related to a similar approach for locally constrained graph homomorphism problems [9]. A graph homomorphism from G to H is a vertex mapping $f : V_G \rightarrow V_H$ satisfying the property that whenever an edge (u, v) appears in E_G , then $(f(u), f(v))$ belongs to E_H as well.

The adjective “locally constrained” expresses the condition that the mapping f restricted to the neighborhood of any vertex u must satisfy further properties. (See [14,7] for a general model of such conditions.)

It may be required to be locally

- *bijective*, then the mapping is called a *full cover* of H , and the corresponding decision problem is called H -COVER [1,13],
- *injective*, then it is called a *partial cover* of H , and the problem H -PCOVER [8,9],
- *surjective*, then we get a *locally surjective cover* of H , and decision problem H -COLORDOMINATION [14].

All these problems are parameterized by a fixed graph H , and the instance is formed only by a graph G . The question is whether an appropriate graph homomorphism from G to H exists. Observe that the definition of a locally surjective cover is equivalent with the definition of an R -role assignment for $R = H$.

Full covers have important applications, for example in distributed computing [5], in recognizing graphs by networks of processors [2,3], or in constructing highly transitive regular graphs [4]. Similarly partial covers are used in distance constrained labelings of graphs [10].

Even if the first attempt to get some results on the computational complexity for the class of H -COVER problems was made a decade ago in [1], it is not fully classified yet neither for H -PCOVER nor for H -COLORDOMINATION ($RA(H)$) problems. However, several partial results are known. For example, if the H -COVER problem is NP-complete, then the corresponding H -PCOVER [9] and

H -COLORDOMINATION problems [14] are NP-complete as well. Moreover, the H -COVER problem is known to be NP-complete for all k -regular graphs H of valency $k \geq 3$ [9], and the NP-hardness hence propagates for partial and locally surjective covers of such graphs as well.

The H -COLORDOMINATION problem was proven to be NP-complete for paths, cycles and stars in [14]. It was conjectured there that for simple connected graphs the H -COLORDOMINATION problem is NP-complete if and only if H has at least three vertices.

Our Results

Our main result completely classifies the computational complexity of the H -COLORDOMINATION problem for all connected role graphs. This proves the conjecture made by Kristiansen and Telle [14]. We also fully determine the complexity of the problem for disconnected role graphs under the extra condition that each role must appear as the image of a vertex of the instance graph (cf. [15]). We finally generalize the result of Roberts and Sheng [15] on 2-role assignment problems by proving NP-completeness for the k -role assignment problem for any fixed $k \geq 2$.

The paper is organized as follows. The next section provides necessary definitions and basic observations. In the third section we show the construction of the main theorem, which proves the conjecture made in [14]. The fourth section describes the complexity of the role assignment problem for disconnected role graphs. We apply the main theorem to prove NP-completeness for the k -role assignment problem in the fifth section.

2 Preliminaries

Through the paper we use terminology stemming from the role assignment problems.

We consider simple graphs, denoted by $G = (V_G, E_G)$, where V_G is a finite vertex set of vertices and E_G is a set of unordered pairs of vertices, called edges. For a vertex $u \in V_G$ we denote its neighborhood, i.e. the set of adjacent vertices, by $N_G(u) = \{v \mid (u, v) \in E_G\}$.

The *degree* $\deg_G(u)$ of a vertex u is the number of edges incident with it, or equivalently the size of its neighborhood. The symbol $\delta(G)$ is the minimum degree among all vertices of G .

A graph G is called *connected* if for every pair of distinct vertices u and v , there exists a *path* connecting u and v , i.e. a sequence of distinct vertices starting by u and ending by v where each pair of consecutive vertices forms an edge of G . The *length* of the path is the number of its edges.

A graph that is not connected is called *disconnected*. Each maximal connected subgraph of a graph is called a *component*. A vertex whose removal causes a component of a graph to become disconnected is called a *cutvertex*. We say

that a cutvertex u separates vertex v from w in G if v, w belong to different components of $G \setminus u$.

Two graphs G and $\tilde{G}$ are called *isomorphic*, denoted by $G \simeq \tilde{G}$, if there exists a one-to-one mapping f of vertices of G onto vertices of $\tilde{G}$ such that $(u, v) \in E_G$ if and only if $(f(u), f(v)) \in E_{\tilde{G}}$.

In the sequel the symbol G denotes the instance graph and R the so-called *role graph*.

Definition 1. We say that G is R -role assignable if a mapping $r : V_G \rightarrow V_R$ exists satisfying:

$$\text{for all } u \in V_G : r(N_G(u)) = N_R(r(u)),$$

where we use the notation $r(S) = \bigcup_{u \in S} r(u)$ for a set of vertices $S \subseteq V_G$. The function r is called an R -role assignment of G .

The goal of this paper is a full characterization of the computational complexity for the following class of problems:

R -ROLE ASSIGNMENT (RA(R))

Instance: A graph G .

Question: Does the graph G allow an R -role assignment?

We continue with some observations that we use later in the paper.

Observation 1 If G is R -role assignable, then $\deg_G(u) \geq \deg_R(r(u))$ for all vertices $u \in V_G$.

Proof. $\deg_G(u) = |N_G(u)| \geq |r(N_G(u))| = |N_R(r(u))| = \deg_R(r(u))$. $\square$

From this we easily derive that $\delta(G) \geq \delta(R)$, and moreover:

Lemma 1. If G is R -role assignable and u is a vertex of G with $\deg_G(u) = \delta(R)$, then $\deg_R(r(u)) = \delta(R)$ and r restricted to $N_G(u)$ is an isomorphism between $N_G(u)$ and $N_R(r(u))$.

Lemma 2. Let G be R -role assignable and x, y be vertices of R connected by a path P_R . Then for each u with $r(u) = x$ a vertex $v \in V_G$ and a path P_G connecting u and v exist, such that r restricted to P_G is an isomorphism between P_G and P_R .

Proof. We prove the statement by induction on the length of the path P_R . If x and y are adjacent, then the vertex u has a neighbor v mapping onto y , by the definition of the R -role assignment r .

Now assume that the path P_R is of length $k \geq 2$, and that the hypothesis is valid for all paths of length at most $k - 1$. Denote by y' the predecessor of y in P_R and by P'_R the truncation of P_R by the last edge, i.e. the path of length $k - 1$ connecting x and y' . By the induction hypothesis G contains a vertex v' and a path P'_G such that $P'_G \simeq P'_R$ under r . Then it is easy to find a neighbor v of v' satisfying $r(v) = y$ and tack it to P'_G to get the desired path P_G . $\square$

We get immediately the following:

Observation 2 *If G is R -role assignable and R is connected, then each vertex $v \in V_R$ appears as a role for some vertex $u \in V_G$.*

Lemma 3. *Let G be R -role assignable, u, u' be vertices of G such that $N_G(u) \subseteq N_G(u')$, and $\deg_G(u) = \delta(R)$. If all vertices of minimum degree in R are cutvertices then $r(u) = r(u')$.*

Proof. We denote $z = r(u)$. Since $\deg_R(z) \leq \deg_G(u) = \delta(R)$ we get that z is a vertex of minimum degree, and by our assumptions it is also a cutvertex in R . Let x, y be two of its neighbors that are separated by z and let $v, w \in N_G(u)$ be their preimages. (Their uniqueness is even guaranteed by Lemma 1.) The image of the path v, u', w is connected, hence it contains the vertex z as the role of u' . $\square$

3 The Main Result

In this section we prove the conjecture of Kristiansen and Telle [14].

Theorem 1. *Let R be a connected role graph. Then the R -role assignment problem is polynomially solvable if $|V_R| \leq 2$ and it is NP-complete if $|V_R| \geq 3$.*

3.1 Sketch of the Proof

It is straightforward to see that the problem is polynomially solvable if the number of vertices of the role graph is at most two. For larger role graphs we prove NP-completeness by making a reduction from hypergraph 2-colorability.

The main idea is to split the problem in various cases depending on the number of cutvertices of minimum degree, the minimum degree and the second common neighborhood of a vertex of minimum degree of R .

For each case we construct an appropriate instance graph from an instance of the hypergraph 2-colorability problem. For this purpose we need several gadgets, which are explained in the next section.

3.2 Gadgets

For the garbage collection in our NP-completeness proof we need to construct a graph that allows two different role assignments.

Lemma 4. *Let R be a role graph. Then a graph H exists that has two R -role assignments r_1 and r_2 , such that for any two roles v and w , a vertex u exists in H with $r_1(u) = v$, and $r_2(u) = w$. Moreover, H can be constructed in time being polynomial with respect to the size of R .*

Proof. Take H as the Cartesian product $R \times R$, defined by the vertex set $V_H = V_R \times V_R$, and edges $((a, b), (c, d)) \in E_H$ if and only if $(a, c), (b, d) \in E_R$.

The projections $r_1 : (a, b) \rightarrow a$ and $r_2 : (a, b) \rightarrow b$ are valid R -role assignments, and the vertex $u = (v, w)$ satisfies the statement of the Lemma. $\square$

Note that for our purposes, it is possible for any two roles v, w to construct a connected H with two role assignments — it is enough to select the component of $R \times R$ containing the vertex $u = (v, w)$.

Definition 2. We say that a graph $\tilde{R}$ is glued in a graph G by a vertex $\tilde{v}$, if G can be obtained from $\tilde{R}$ and some other graph G' by identifying a vertex $x \in V_{G'}$ with the vertex $\tilde{v}$.

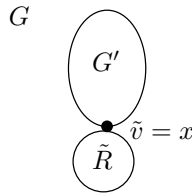


Fig. 1. A graph with a glued subgraph

As a convention we use letters x, y, z to denote roles, while u is reserved for vertices of the instance. The symbols v, w stand for roles, while $\tilde{v}$ or $\tilde{w}$ are vertices of the instance graph isomorphic to v, w . The proof of the following lemma is omitted in this extended abstract.

Lemma 5. Let R be a connected role graph. Let G be an R -role assignable graph and $\tilde{R}$ be glued in G by a vertex $\tilde{v}$, where $\tilde{R}$ is isomorphic to R and $\tilde{v}$, the isomorphic copy of $\tilde{v}$ in $\tilde{R}$, is not a cutvertex of R . Then an R -role assignment r exists such that $r(\tilde{w}) = w$ for every $w \in V_R$.

3.3 Proof of the Main Theorem

Proof. First we show that $\text{RA}(R)$ is polynomially solvable for $|V_R| \leq 2$.

- $|V_R| = 1$. Clearly, a graph G is R -role assignable if and only if G contains only isolated vertices.
- $|V_R| = 2$. Clearly, a graph G is R -role assignable if and only if G is a bipartite graph that does not contain any isolated vertices.

Now let $|V_R| \geq 3$. Since we can guess a mapping $r : V_G \rightarrow V_R$ and check in polynomial time if r is an R -role assignment, the problem $\text{RA}(R)$ is a member of NP. We prove NP-completeness by reduction from hypergraph 2-colorability. This is a well-known NP-complete problem (cf. [11]).

HYPERGRAPH 2-COLORABILITY (H2C)

Instance: A set $Q = \{q_1, \dots, q_m\}$ and a set $\mathcal{S} = \{S_1, \dots, S_n\}$ with $S_j \subseteq Q$ for $1 \leq j \leq n$.

Question: Is there a 2-coloring of $(Q, \mathcal{S})$, i.e., a partition of Q into $Q_1 \cup Q_2$ such that $Q_1 \cap S_j \neq \emptyset$ and $Q_2 \cap S_j \neq \emptyset$ for $1 \leq j \leq n$?

With such a hypergraph we associate its incidence graph I , which is a bipartite graph on $Q \cup \mathcal{S}$, where (q, S) forms an edge if and only if $q \in S$.

To prove the theorem we choose a vertex $v \in V_R$ of minimum degree. Because we cannot apply Lemma 5 if v is a cutvertex, we have to distinguish between the case, in which all vertices of minimum degree are cutvertices, and the case, in which a non-cutvertex of minimum degree exists.

Assume first that the vertex v is a vertex of minimum degree that *is not* a cutvertex. Denote the neighbors of v by $N_R(v) = \{w_1, \dots, w_p\}$ and also the second common neighborhood as $M_R(v) = \bigcap_{u \in N_R(v)} N_R(u) = \{v, v_2, \dots, v_l\}$. See Fig. 2 for a drawing of a possible situation.

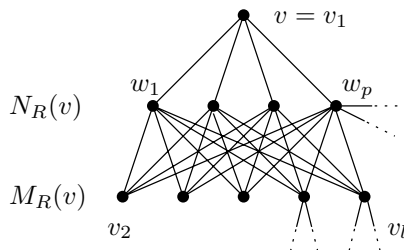


Fig. 2. Neighborhood of a vertex v in R .

We distinguish four cases according to possible values of p and l :

Case 1: $p = 1, l = 1$. Then $R = K_2$ and we have already discussed this case above.

Case 2: $p = 1, l \geq 3$. We extend the incidence graph I as follows: According to Lemma 4 we construct a graph H for which two role assignments exist mapping a particular vertex u to v_2 and v_3 . We form an instance G as the union of the graph I and m disjoint copies of the graph H , where the vertex u of the i -th copy is identified with the vertex q_i of I . Finally we insert into G two extra copies $\tilde{R}, R'$ of the role graph R and add the following edges (cf. Fig 3):

- $(\tilde{v}, S_j)$ for all $S_j \in \mathcal{S}$,
- (v'_k, S_j) for all $S_j \in \mathcal{S}$ and all $4 \leq k \leq l$ (this set may be empty).

We show that the graph G formed in this way allows an R -role assignment if and only if $(Q, \mathcal{S})$ is 2-colorable.

Assume first that G is R -role assignable. Then according to Lemma 5 we assume that the vertex $\tilde{v}$ is assigned role v and all vertices S_j are mapped to role w_1 . Since their neighborhoods are saturated by common $l - 3$ roles on

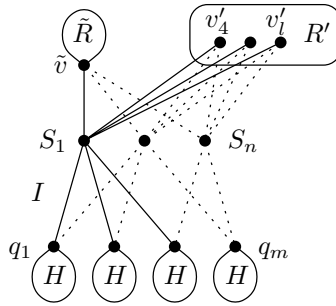


Fig. 3. Construction of the graph G in Case 2.

$v'_4, \dots, v'_l$, at least two distinct roles $v_a, v_b \in M_R(v) \setminus r(\{v'_4, \dots, v'_l\})$ exist that are used on some neighbors of each S_j in the set $\mathcal{S}$.

The partition $Q_1 = \{q_i \mid r(q_i) = v_a\}$ and $Q_2 = Q \setminus Q_1 \supseteq \{q_i \mid r(q_i) = v_b\}$ is the desired 2-coloring of $(Q, \mathcal{S})$.

In the opposite direction, any 2-coloring Q_1, Q_2 can be transformed into an R -role assignment r of G by letting $r(q_i) = v_a$ if $q_i \in Q_a$ for $a = 1, 2$ and by further extension according to the two projections of the graph H and graph isomorphisms $\tilde{R} \rightarrow R, R' \rightarrow R$.

Case 3: $p = 1, l = 2$. The case when R is isomorphic to the path P_4 was already shown to be NP-complete in [14]. If R is not isomorphic to a path on four vertices but v_2 is incident with a vertex v^* of degree one, then we can reduce this case to the previous case ($p = 1, l \geq 3$) by selecting v^* as the non-cutvertex of minimum degree. So without loss of generality we may assume that v_2 is not incident with a vertex of degree one.

We construct G from I as follows. First we insert n new vertices $S'_1, \dots, S'_n$ and a copy $\tilde{R}$ of the role graph R . We identify each q_i with the vertex u of an extra copy of the graph H as in the previous case, but here H is constructed such that u can be assigned v or v_2 .

These parts are linked as follows (cf. Fig. 4):

- $(\tilde{v}, S'_j) \in E_G$ for all $j \in \{1, \dots, n\}$,
- $(q_i, S'_j) \in E_G$ if and only if $(q_i, S_j) \in E_I$.

If G is R -role assignable, then without loss of generality we may assume that $\tilde{v}$ has role v . Then all S'_j have role w_1 since w_1 is the only neighbor of v . The roles of all q_i hence belong to $N_R(w_1) = \{v, v_2\}$. Each S'_j requires the role v_2 to be present among its neighbors in Q . Moreover, if all neighbors of some S'_j in Q are assigned the role v_2 , we get that S_j must be mapped to a neighbor of v_2 that is a leaf, but this is in contradiction with our assumptions. We conclude that each S_j is mapped to w_1 . Hence both roles v, v_2 appear on its neighborhood and the partition $Q_1 = \{q_i \mid r(q_i) = v\}$ and $Q_2 = \{q_i \mid r(q_i) = v_2\}$ is a 2-coloring of $(Q, \mathcal{S})$.

In the opposite direction, an R -role assignment of G can be constructed from a 2-coloring of $(Q, \mathcal{S})$ in a straightforward way as in the previous case.

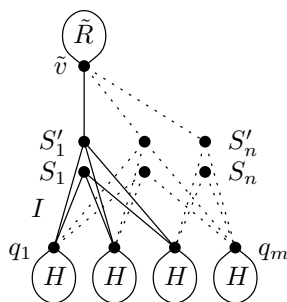


Fig. 4. Construction of the graph G in Case 3.

Case 4: $p \geq 2$. As above we first build the graph H which allows two R -role assignments mapping a vertex u either to w_1 or to w_2 .

The graph G consists of the graph I , where each q_i is unified with the vertex u of an extra copy of H . We further include two copies of R denoted by $\tilde{R}$ and R' . Finally we extend the set of edges by (cf. Fig. 5):

- $(\tilde{v}, q_i)$ for all $q_i \in Q$,
- $(\tilde{v}, w'_k)$ for all $1 \leq k \leq p$,
- (S_j, w'_k) for all $3 \leq k \leq p$ (this set may be empty).

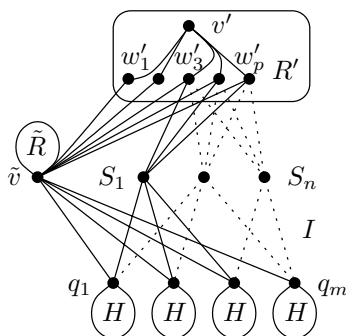


Fig. 5. Construction of the graph G in Case 4.

If an R -role assignment exists, then we assume that $r(\tilde{v}) = v$. For each S_j we have $N_G(S_j) \subseteq N_G(\tilde{v})$. So we know that S_j is assigned some role v_i for which $N_R(v_i) = N_R(v)$. However only $p - 2$ roles appear on vertices $w'_3, \dots, w'_p$, so two distinct roles w_a and w_b are used on none of $w'_3, \dots, w'_p$. Then we define a 2-coloring of (Q, S) by selecting $Q_1 = \{q_i \mid r(q_i) = w_a\}$ and $Q_2 = Q \setminus Q_1 \supseteq \{q_i \mid r(q_i) = w_b\}$.

An R -role assignment can be derived from a 2-coloring of (Q, S) as in the previous cases.

Finally, we return to the situation when all vertices of minimum degree in R are cutvertices. (Observe, that $\delta(R) \geq 2$ since vertices of degree one are not cutvertices.)

We construct the graph G as in Case 4 above (cf. Fig. 5). The argumentation goes in the same manner: Since $N_G(v') \subseteq N_G(\tilde{v})$, we get by Lemma 3 that $\tilde{v}$ is mapped to a role of minimum degree. For each S_j we have $N_G(S_j) \subseteq N_G(\tilde{v})$. So we know that S_j is assigned a role that has the same neighbors in R as role $r(\tilde{v})$. Each S_j then lacks two roles w_a, w_b that do not appear on $w'_3, \dots, w'_p$. Hence we can define a valid 2-coloring of $(Q, \mathcal{S})$ according to the appearance of roles w_a and w_b on the set Q . $\square$

4 Disconnected Role Graphs

Up to now we have only considered role graphs that were connected. Due to this property we could easily derive that all roles appear as the image of a vertex in the instance graph (cf. Observation 2). We now focus our attention to the case of disconnected role graphs. Suppose R is a role graph with set of components $\mathcal{C} = \{C_1, \dots, C_m\}$. We order the components such that the latter have a higher number of vertices. (Formally, for all $i \leq j : |V_{C_i}| \leq |V_{C_j}|$.)

Note that the identity mapping $\pi : V_{C_1} \rightarrow V_R$ preserves the local constraint for role assignment, but Observation 2 is no longer valid here (take $G \simeq C_1$). Our argument guarantees that a locally surjective cover is globally surjective only for connected role graphs. Within some social network models it is natural to demand that all roles appear on the vertices of the instance graph. We show below that the computational complexity of the role assignment problem for disconnected role graphs depends whether such a property $r(V_G) = V_R$ is required or not.

We call an R -role assignment $r : V_G \rightarrow V_R$ a *globally R -role assignment* for G if r is an R -role assignment and $r(V_G) = V_R$ holds. Our generalized role assignment problem can now be formulated as

GLOBAL R -ROLE ASSIGNMENT (GRA(R))

Instance: A graph G .

Question: Is G globally R -role assignable?

With respect to the computational complexity we obtain the following result. (The proof is omitted in this abstract.)

Theorem 2. *Let R be a disconnected role graph. Then the GRA(R) problem is polynomially solvable if all components have at most two vertices and it is NP-complete otherwise.*

Now we show that without the condition of global surjectivity “ $r(V_G) = V_R$ ”, some polynomially solvable RA(R) problems exist for role graphs R with large components.

Take any role graph R with bipartite components (of arbitrary size) but assure that at least one of these components is isomorphic to K_2 (i.e. to a graph

consisting of two vertices forming an edge). For simplicity assume that R has no isolated vertices. We claim that G is R -role assignable if and only if G is bipartite without isolated vertices. The necessity of such condition follows from the fact that non-bipartite graphs have no homomorphism to bipartite graphs. In the opposite direction, any homomorphism from G to K_2 can be viewed as an R -role assignment of G .

Our conjecture is that for all other simple role graphs the problem is NP-complete. Although we have shown above a proof of the polynomial part of the statement, we do not see a direct way for a possible NP-hardness construction.

5 k -Role Assignability

In this section we study a more general version of the role assignment problem. We call a graph G *k -role assignable* if there exists a role graph R on k vertices, such that G is globally R -role assignable.

k -ROLE ASSIGNMENT (k -RA)

Instance: A graph G .

Question: Is G k -role assignable?

This problem was studied by [15] and is of interest in social network theory where networks are modeled in which individuals of the same social role relate to other individuals in the same way. The networks of individuals are represented by simple graphs. Contrary to our previous results, in this new model two individuals that are related to each other may have the same role. Hence role graphs that contain loops are allowed.

Again our aim is to fully characterize the computational complexity of the k -RA problem. Clearly the 1-RA problem is solvable in linear time, since it is sufficient to check whether G has no edges ($R = K_1$) or whether all vertices in G have degree at least one (R consists of one vertex with a loop). The 2-RA problem is proven to be NP-complete in [15]. We generalize this result as follows. (The proof is omitted in this abstract.)

Corollary 1. *The k -RA problem is polynomially solvable for $k = 1$ and it is NP-complete for all $k \geq 2$.*

The computational complexity of the role assignment problem can be studied also for role graphs that contain some loops. If all components of R either consist of exactly one vertex or are isomorphic to K_2 , the $\text{RA}(R)$ problem is polynomially solvable. The conjecture is that in all other cases the problem is NP-complete, even if instances are restricted to simple graphs.

We expect that our constructions would work in a similar way. Instead of a graph isomorphic to the role graph an other appropriate graph should be glued in the instance graph to obtain a reduction from the H2C problem as we have used in the proof of Theorem 1.

References

1. ABELLO, J., FELLOWS, M. R., AND STILLWELL, J. C. On the complexity and combinatorics of covering finite complexes. *Australian Journal of Combinatorics* 4 (1991), 103–112.
2. ANGLUIN, D. Local and global properties in networks of processors. In *Proceedings of the 12th ACM Symposium on Theory of Computing* (1980), 82–93.
3. ANGLUIN, D., AND GARDINER, A. Finite common coverings of pairs of regular graphs. *Journal of Combinatorial Theory B* 30 (1981), 184–187.
4. BIGGS, N. Constructing 5-arc transitive cubic graphs. *Journal of London Mathematical Society II*. 26 (1982), 193–200.
5. BODLAENDER, H. L. The classification of coverings of processor networks. *Journal of Parallel Distributed Computing* 6 (1989), 166–182.
6. EVERETT, M. G., AND BORGATTI, S. Role coloring a graph. *Mathematical Social Sciences* 21, 2 (1991), 183–188.
7. FIALA, J., HEGGERNES, P., KRISTIANSEN, P., AND TELLE, J. A. Generalized H -coloring and H -covering of trees. In *Graph-Theoretical Concepts in Computer Science, 28th WG '02, Český Krumlov* (2002), no. 2573 in Lecture Notes in Computer Science, Springer Verlag, pp. 198–210.
8. FIALA, J., AND KRATOCHVÍL, J. Complexity of partial covers of graphs. In *Algorithms and Computation, 12th ISAAC '01, Christchurch, New Zealand* (2001), no. 2223 in Lecture Notes in Computer Science, Springer Verlag, pp. 537–549.
9. FIALA, J., AND KRATOCHVÍL, J. Partial covers of graphs. *Discussiones Mathematicae Graph Theory* 22 (2002), 89–99.
10. FIALA, J., KRATOCHVÍL, J., AND KLOKS, T. Fixed-parameter complexity of λ -labelings. *Discrete Applied Mathematics* 113, 1 (2001), 59–72.
11. GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability*. W. H. Freeman and Co., New York, 1979.
12. HEGGERNES, P., AND TELLE, J. A. Partitioning graphs into generalized dominating sets. *Nordic Journal of Computing* 5, 2 (1998), 128–142.
13. KRATOCHVÍL, J., PROSKUROWSKI, A., AND TELLE, J. A. Covering regular graphs. *Journal of Combinatorial Theory B* 71, 1 (1997), 1–16.
14. KRISTIANSEN, P., AND TELLE, J. A. Generalized H -coloring of graphs. In *Algorithms and Computation, 11th ISAAC '01, Taipei, Taiwan* (2000), no. 1969 in Lecture Notes in Computer Science, Springer Verlag, pp. 456–466.
15. ROBERTS, F. S., AND SHENG, L. How hard is it to determine if a graph has a 2-role assignment? *Networks* 37, 2 (2001), 67–73.

Fixed-Parameter Algorithms for the (k, r) -Center in Planar Graphs and Map Graphs

Erik D. Demaine¹, Fedor V. Fomin², Mohammad Taghi Hajiaghayi¹, and
Dimitrios M. Thilikos³

¹ MIT Laboratory for Computer Science, 200 Technology Square, Cambridge,
Massachusetts 02139, USA, {edemaine,hajiaghayi}@mit.edu

² Department of Informatics, University of Bergen, N-5020 Bergen, Norway,
fomin@ii.uib.no

³ 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. The (k, r) -center problem asks whether an input graph G has $\leq k$ vertices (called *centers*) such that every vertex of G is within distance $\leq r$ from some center. In this paper we prove that the (k, r) -center problem, parameterized by k and r , is fixed-parameter tractable (FPT) on planar graphs, i.e., it admits an algorithm of complexity $f(k, r)n^{O(1)}$ where the function f is independent of n . In particular, we show that $f(k, r) = 2^{O(r \log r) \sqrt{k}}$, where the exponent of the exponential term grows sublinearly in the number of centers. Moreover, we prove that the same type of FPT algorithms can be designed for the more general class of *map graphs* introduced by Chen, Grigni, and Papadimitriou. Our results combine dynamic-programming algorithms for graphs of small branchwidth and a graph-theoretic result bounding this parameter in terms of k and r . Finally, a byproduct of our algorithm is the existence of a PTAS for the r -domination problem in both planar graphs and map graphs.

Our approach builds on the seminal results of Robertson and Seymour on Graph Minors, and as a result is much more powerful than the previous machinery of Alber et al. for exponential speedup on planar graphs. To demonstrate the versatility of our results, we show how our algorithms can be extended to general parameters that are “large” on grids. In addition, our use of branchwidth instead of the usual treewidth allows us to obtain much faster algorithms, and requires more complicated dynamic programming than the standard leaf/introduce/forget/join structure of nice tree decompositions. Our results are also unique in that they apply to classes of graphs that are not minor-closed, namely, constant powers of planar graphs and map graphs.

1 Introduction

Clustering is a key tool for solving a variety of application problems such as data mining, data compression, pattern recognition and classification, learning, and

facility location. Among the algorithmic problem formulations of clustering are k -means, k -medians, and k -center. In all of these problems, the goal is to partition n given points into k clusters so that some objective function is minimized.

In this paper, we concentrate on the (unweighted) (k, r) -center problem [7], in which the goal is to choose k centers from the given set of n points so that every point is within distance r from some center in the graph. In particular, the k -center problem [17] of minimizing the maximum distance to a center is exactly (k, r) -center when the goal is to minimize r subject to finding a feasible solution. In addition, the r -domination problem [7,16] of choosing the fewest vertices whose r -neighborhoods cover the whole graph is exactly (k, r) -center when the goal is to minimize k subject to finding a feasible solution.

A sample application of the (k, r) -center problem in the context of facility location is the installation of emergency service facilities such as fire stations. Here we suppose that we can afford to buy k fire stations to cover a city, and we require every building to be within r city blocks from the nearest fire station to ensure a reasonable response time. Given an algorithm for (k, r) -center, we can vary k and r to find the best bicriterion solution according to the needs of the application. In this scenario, we can afford high running time (e.g., several weeks of real time) if the resulting solution builds fewer fire stations (which are extremely expensive) or has faster response time; thus, we prefer fixed-parameter algorithms over approximation algorithms.

In this application, and many others, the graph is typically planar or nearly so. Chen, Grigni, and Papadimitriou [9] have introduced a generalized notion of planarity which allows local nonplanarity. In this generalization, two countries of a map are adjacent if they share at least one point, and the resulting graph of adjacencies is called a *map graph*. (See Section 2 for a precise definition.) Planar graphs are the special case of map graphs in which at most three countries intersect at a point.

Previous results. r -domination and k -center are NP-hard even for planar graphs. For r -domination, the current best approximation (for general graphs) is a $(\log n + 1)$ -factor by phrasing the problem as an instance of set cover [7]. For k -center, there is a 2-approximation algorithm [17] which applies more generally to the case of weighted graphs satisfying the triangle inequality. Furthermore, no $(2 - \epsilon)$ -approximation algorithm exists for any $\epsilon > 0$ even for unweighted planar graphs of maximum degree 3 [22]. For geometric k -center in which the weights are given by Euclidean distance in d -dimensional space, there is a PTAS whose running time is exponential in k [1]. Several relations between small r -domination sets for planar graphs and problems about organizing routing schemes with compact structures is discussed in [16].

The (k, r) -center problem can be considered as a generalization of the well-known dominating set problem. During the last two years in particular much attention has been paid to constructing fixed-parameter algorithms with exponential speedup for this problem. Alber et al. [2] were the first who demonstrated an algorithm checking whether a planar graph has a dominating set of size $\leq k$ in time $O(2^{70\sqrt{k}}n)$. This result was the first non-trivial result for the parame-

terized version of an NP-hard problem in which the exponent of the exponential term grows sublinearly in the parameter. Recently, the running time of this algorithm was further improved to $O(2^{27\sqrt{k}}n)$ [20] and $O(2^{15.13\sqrt{k}}k + n^3 + k^4)$ [14]. Fixed-parameter algorithms for solving many different problems such as vertex cover, feedback vertex set, maximal clique transversal, and edge-dominating set on planar and related graphs such as single-crossing-minor-free graphs are considered in [11,21]. Most of these problems have reductions to the dominating set problem. Also, because all these problems are closed under taking minors or contractions, all classes of graphs considered so far are minor-closed.

Our results. In this paper, we focus on applying the tools of *parameterized complexity*, introduced by Downey and Fellows [12], to the (k, r) -center problem in planar and map graphs. We view both k and r as parameters to the problem. We introduce a new proof technique which allows us to extend known results on planar dominating set in two different aspects.

First, we extend the exponential speed-up for a generalization of dominating set, namely the (k, r) -center problem, on planar graphs. Specifically, the running time of our algorithm is $O((2r + 1)^{6(2r+1)\sqrt{k}+12r+3/2}n + n^4)$, where n is the number of vertices. Our proof technique is based on combinatorial bounds (Section 3) derived from the Robertson, Seymour, and Thomas theorem about quickly excluding planar graphs, and on a complicated dynamic program on graphs of bounded branchwidth (Section 4). Second, we extend our fixed-parameter algorithm to map graphs which is a class of graphs that is not minor-closed. In particular, the running time of the corresponding algorithm is $O((2r + 1)^{6(4r+1)\sqrt{k}+24r+3}n + n^4)$.

Notice that the exponential component of the running times of our algorithms depends only on the parameters, and is multiplicatively separated from the problem size n . Moreover, the contribution of k in the exponential part is sublinear. In particular, our algorithms have polynomial running time if $k = O(\log^2 n)$ and $r = O(1)$, or if $r = O(\log n / \log \log n)$ and $k = O(1)$. We stress the fact that we design our dynamic-programming algorithms using branchwidth instead of treewidth because this provides better running times.

Finally, in Section 6, we present several extensions of our results, including a PTAS for the r -dominating set problem and a generalization to a broad class of graph parameters.

2 Definitions and Preliminary Results

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. We let n denote the number of vertices of a graph when it is clear from context. For every nonempty $W \subseteq V(G)$, the subgraph of G induced by W is denoted by $G[W]$.

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 . A graph H is a *minor* of a

graph G if H is a subgraph of a contraction of G . We use the notation $H \preceq G$ (resp. $H \preceq_c G$) for H is a minor (a contraction) of G .

(k, r) -center. We define the r -neighborhood of a set $S \subseteq V(G)$, denoted by $N_G^r(S)$, to be the set of vertices of G at distance at most r from at least one vertex of S ; if $S = \{v\}$ we simply use the notation $N_G^r(v)$. We say a graph G has a (k, r) -center or interchangeably has an r -dominating set of size k if there exists a set S of centers (vertices) of size at most k such that $N_G^r(S) = V(G)$. We denote by $\gamma_r(G)$ the smallest k for which there exists a (k, r) -center in the graph. One can easily observe that for any r the problem of checking whether an input graph has a (k, r) -center, parameterized by k is $W[2]$ -hard by a reduction from dominating set. (See Downey and Fellows [12] for the definition of the W Hierarchy.)

Map graphs. Let Σ be a sphere. A Σ -plane graph G is a planar graph G drawn in Σ . To simplify notation, we usually do not distinguish between a vertex of the graph and the point of Σ used in the drawing to represent the vertex, or between an edge and the open line segment representing it. We denote the set of regions (faces) in the drawing of G by $R(G)$. (Every region is an open set.) An edge e or a vertex v is *incident* to a region r if $e \subseteq \bar{r}$ or $v \subseteq \bar{r}$, respectively. ($\bar{r}$ denotes the *closure* of r .)

For a Σ -plane graph G , a *map* $\mathcal{M}$ is a pair (G, ϕ) , where $\phi : R(G) \rightarrow \{0, 1\}$ is a two-coloring of the regions. A region $r \in R(G)$ is called a *nation* if $\phi(r) = 1$ and a *lake* otherwise.

Let $N(\mathcal{M})$ be the set of nations of a map $\mathcal{M}$. The graph F is defined on the vertex set $N(\mathcal{M})$, in which two vertices r_1, r_2 are adjacent precisely if $\bar{r}_1 \cap \bar{r}_2$ contains at least one edge of G . Because f is the subgraph of the dual graph G^* of G , it is planar. Chen, Grigni, and Papadimitriou [9] defined the following generalization of planar graphs. A *map graph* $G_{\mathcal{M}}$ of a map $\mathcal{M}$ is the graph on the vertex set $N(\mathcal{M})$ in which two vertices r_1, r_2 are adjacent in $G_{\mathcal{M}}$ precisely if $\bar{r}_1 \cap \bar{r}_2$ contains at least one vertex of G .

For a graph G , we denote by G^k the k th power of G , i.e., the graph on the vertex set $V(G)$ such that two vertices in G^k are adjacent precisely if the distance in G between these vertices is at most k . Let G be a bipartite graph with a bipartition $U \cup W = V(G)$. The *half square* $G^2[U]$ is the graph on the vertex set U and two vertices are adjacent in $G^2[U]$ precisely if the distance between these vertices in G is 2.

Theorem 1 ([9]). *A graph $G_{\mathcal{M}}$ is a map graph if and only if it is the half-square of some planar bipartite graph H .*

Here the graph H is called a *witness* for $G_{\mathcal{M}}$. Thus the question of finding a (k, r) -center in a map graph $G_{\mathcal{M}}$ is equivalent to finding in a witness H of $G_{\mathcal{M}}$ a set $S \subseteq V(G_{\mathcal{M}})$ of size k such that every vertex in $V(G_{\mathcal{M}}) - S$ has distance $\leq 2r$ in H from some vertex of S .

The proof of Theorem 1 is constructive, i.e., given a map graph $G_{\mathcal{M}}$ together with its map $\mathcal{M} = (G, \phi)$, one can construct a witness H for $G_{\mathcal{M}}$ in

time $O(|V(G_{\mathcal{M}})| + |E(G_{\mathcal{M}})|)$. One color class $V(G_{\mathcal{M}})$ of the bipartite graph H corresponds to the set of nations of the map $\mathcal{M}$. Each vertex v of the second color class $V(H) - V(G_{\mathcal{M}})$ corresponds to an intersection point of boundaries of some nations, and v is adjacent (in H) to the vertices corresponding to the nations it belongs. What is important for our proofs are the facts that

1. in such a witness, every vertex of $V(H) - V(G_{\mathcal{M}})$ is adjacent to a vertex of $V(G_{\mathcal{M}})$, and
2. $|V(H)| = O(|V(G_{\mathcal{M}})| + |E(G_{\mathcal{M}})|)$.

Thorup [27] provided a polynomial-time algorithm for constructing a map of a given map graph in polynomial time. However, in Thorup's algorithm, the exponent in the polynomial time bound is about 120 [8]. So from practical point of view there is a big difference whether we are given a map in addition to the corresponding map graph. Below we suppose that we are always given the map.

Branchwidth. Branchwidth was introduced by Robertson and Seymour in their Graph Minors series of papers. A *branch decomposition* of a 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 $E(G)$ to the set of leaves of T . The *order function* $\omega : E(T) \rightarrow 2^{V(G)}$ of a branch decomposition maps every edge e of T to a subset of vertices $\omega(e) \subseteq V(G)$ as follows. The set $\omega(e)$ consists of all vertices of $V(G)$ such that, for every vertex $v \in \omega(e)$, there exist edges $f_1, f_2 \in E(G)$ such that $v \in f_1 \cap f_2$ and the leaves $\tau(f_1), \tau(f_2)$ are in different components of $T - \{e\}$. The *width* of (T, τ) is equal to $\max_{e \in E(T)} |\omega(e)|$ and the *branchwidth* of G , $\mathbf{bw}(G)$, is the minimum width over all branch decompositions of G .

It is well-known that, if $H \preceq G$ or $H \preceq_c G$, then $\mathbf{bw}(H) \leq \mathbf{bw}(G)$.

The following deep result of Robertson, Seymour, and Thomas (Theorems (4.3) in [23] and (6.3) in [24]) plays an important role in our proofs.

Theorem 2 ([24]). *Let $k \geq 1$ be an integer. Every planar graph with no $(k \times k)$ -grid as a minor has branchwidth $\leq 4k - 3$.*

Branchwidth is the main tool in this paper. All our proofs can be rewritten in terms of the related and better-known parameter *treewidth*, and indeed treewidth would be easier to handle in our dynamic program. However, branchwidth provides better combinatorial bounds resulting in exponential speed-up of our algorithms.

3 Combinatorial Bounds

Lemma 1. *Let $\rho, k, r \geq 1$ be integers and G be a planar graph having a (k, r) -center and with a $(\rho \times \rho)$ -grid as a minor. Then $k \geq \left(\frac{\rho-2r}{2r+1}\right)^2$.*

Proof. We set $V = \{1, \dots, \rho\} \times \{1, \dots, \rho\}$. Let

$$F = (V, \{((x, y), (x', y')) \mid |x - x'| + |y - y'| = 1\})$$

be a plane $(\rho \times \rho)$ -grid that is a minor of some plane embedding of G . W.l.o.g. we assume that the external (infinite) face of this embedding of F is the one that is incident to the vertices of the set $V_{\text{ext}} = \{(x, y) \mid x = 1 \text{ or } x = \rho \text{ or } y = 1 \text{ or } y = \rho\}$, i.e., the vertices of F with degree < 4 . We call the rest of the faces of F *internal faces*. We set $V_{\text{int}} = \{(x, y) \mid r + 1 \leq x \leq \rho - r, r + 1 \leq y \leq \rho - r\}$, i.e., V_{int} is the set of all vertices of F within distance $\geq r$ from all vertices in V_{ext} . Notice that $F[V_{\text{int}}]$ is a sub-grid of F and $|V_{\text{int}}| = (\rho - 2r)^2$. Given any pair of vertices $(x, y), (x', y') \in V$ we define $\delta((x, y), (x', y')) = \max\{|x - x'|, |y - y'|\}$.

We also define $d_F((x, y), (x', y'))$ to be the distance between any pair of vertices (x, y) and (x', y') in F . Finally we define J to be the graph occurring from F by adding in it the edges of the following sets:

$$\{((x, y), (x + 1, y + 1)) \mid 1 \leq x \leq \rho - 1, 1 \leq y \leq \rho - 1\}$$

$$\{((x, y + 1), (x + 1, y)) \mid 1 \leq x \leq \rho - 1, 1 \leq y \leq \rho - 1\}$$

(In other word we add all edges connecting pairs of non-adjacent vertices incident to its internal faces). It is easy to verify that $\forall (x, y), (x', y') \in V$ $\delta((x, y), (x', y')) = d_J((x, y), (x', y'))$. This implies the following.

If R is a subgraph of J , then $\forall (x, y), (x', y') \in V$ $\delta((x, y), (x', y')) \leq d_R((x, y), (x', y'))$ (1)

For any $(x, y) \in V$ we define $B_r((x, y)) = \{(a, b) \in V \mid \delta((x, y), (a, b)) \leq r\}$ and we observe the following:

$$\forall (x, y) \in V \quad |V(B_r((x, y)))| \leq (2r + 1)^2. \quad (2)$$

Consider now the sequence of edge contractions/removals that transform G to F . If we apply on G only the contractions of this sequence we end up with a planar graph H that can be obtained by the $(\rho \times \rho)$ -grid F after adding edges to non-consecutive vertices of its faces. This makes it possible to partition the additional edges of H into two sets: a set denoted by E_1 whose edges connect non-adjacent vertices of some square face of F and another set E_2 whose edges connect pairs of vertices in V_{ext} . We denote by R the graph obtained by F if we add the edges of E_1 in F . As R is a subgraph of J , (1) implies that

$$\forall (x, y) \in V \quad N_R^r((x, y)) \subseteq B_r((x, y)) \quad (3)$$

We also claim that

$$\forall (x, y) \in V \quad N_H^r((x, y)) \subseteq B_r((x, y)) \cup (V - V_{\text{int}}) \quad (4)$$

To prove (4) we notice first that if we replace H by R in it then the resulting relation follows from (3). It remains to prove that the consecutive addition of edges of E_2 in R does not introduce in $N_H^r((x, y))$ any vertex of V_{int} . Indeed, this is correct because any vertex in V_{ext} is in distance $\geq r$ from any vertex in V_{int} . Notice now that (4) implies that $\forall (x, y) \in V \quad N_H^r((x, y)) \cap V_{\text{int}} \subseteq B_r((x, y)) \cap V_{\text{int}}$ and using (2) we conclude that

$$\forall (x, y) \in V \quad |N_H^r((x, y)) \cap V_{\text{int}}| \leq (2r + 1)^2 \quad (5)$$

Let S be a (k', r) -center in the graph H . Applying (5) on S we have that the r -neighborhood of any vertex in S contains at most $(2r + 1)^2$ vertices from V_{int} . Moreover, any vertex in V_{int} should belong to the r -neighborhood of some vertex in S . Thus $k' \geq |V_{\text{int}}|/(2r + 1)^2 = (\rho - 2r)^2/(2\rho + 1)^2$ and therefore $k' \geq (\frac{\rho - 2r}{2r + 1})^2$.

Clearly, the conditions that G has an r -dominating set of size k and $H \preceq_c G$ imply that H has an r -dominating set of size $k' \leq k$. (But this is not true for $H \preceq G$.) As H is a contraction of G and G has a (k, r) -center, we have that $k \geq k' \geq (\frac{\rho - 2r}{2r + 1})^2$ and lemma follows.

We are ready to prove the main combinatorial result of this paper:

Theorem 3. *For any planar graph G having a (k, r) -center, $\text{bw}(G) \leq 4(2r + 1)\sqrt{k} + 8r + 1$.*

Proof. Suppose that $\text{bw}(G) > p = 4(2r + 1)\sqrt{k} + 8r + \epsilon - 3$ for some ϵ , $0 < \epsilon \leq 4$, for which $p + 3 \equiv 0 \pmod{4}$. By Theorem 2, G contains a $(\rho \times \rho)$ -grid as a minor where $\rho = (2r + 1)\sqrt{k} + 2r + \frac{\epsilon}{4}$. By Lemma 1, $k \geq (\frac{\rho - 2r}{2r + 1})^2 = (\frac{(2r + 1)\sqrt{k} + \frac{\epsilon}{4}}{2r + 1})^2$ which implies that $\sqrt{k} \geq \sqrt{k} + \frac{\epsilon}{8r + 4}$, a contradiction.

Notice that the branchwidth of a map graph is unbounded in terms of k and r . For example, a clique of size n is a map graph and has a $(1, 1)$ -center and branchwidth $\geq 2/3n$.

Theorem 4. *For any map graph $G_{\mathcal{M}}$ having a (k, r) -center and its witness H , $\text{bw}(H) \leq 4(4r + 3)\sqrt{k} + 16r + 9$.*

Proof. The question of finding a (k, r) -center in a map graph $G_{\mathcal{M}}$ is equivalent to finding in a witness H of $G_{\mathcal{M}}$ a set $S \subseteq V(G_{\mathcal{M}})$ of size k such that every vertex $V(G_{\mathcal{M}}) - S$ is at distance $\leq 2r$ in H from some vertex of S . By the construction of the witness graph, every vertex of $V(H) - V(G_{\mathcal{M}})$ is adjacent to some vertex of $V(G_{\mathcal{M}})$. Thus H has a $(k, 2r + 1)$ -center and by Theorem 3 the proof follows.

4 (k, r) -Centers in Graphs of Bounded Branchwidth

In this section, we present a dynamic-programming approach to solve the (k, r) -center problem on graphs of bounded branchwidth. It is easy to prove that, for a fixed r , the problem is in MSOL (monadic second-order logic) and thus can be solved in linear time on graphs of bounded treewidth (branchwidth). However, for r part of the input, the situation is more difficult. Additionally, we are interested in not just a linear-time algorithm but in an algorithm with running time $f(k, r)n$.

It is worth mentioning that our algorithm requires more than a simple extension of Alber et al.'s algorithm for dominating set in graphs of bounded treewidth [2], which corresponds to the case $r = 1$. In fact, finding a (k, r) -center is similar to finding homomorphic subgraphs, which has been solved only

for special classes of graphs and even then only via complicated dynamic programs [18]. The main difficulty is that the path $v = v_0, v_1, v_2, \dots, v_{\leq r} = c$ from a vertex v to its assigned center c may wander up and down the branch decomposition repeatedly, so that c and v may be in radically different ‘cuts’ induced by the branch decomposition. All we can guarantee is that the next vertex v_1 along the path from v to c is somewhere in a common ‘cut’ with v , and that vertex v_1 and v_2 are in a common ‘cut’, etc. In this way, we must propagate information through the v_i ’s about the remote location of c .

Let (T', τ) be a branch decomposition of a graph G with m edges and let $\omega' : E(T') \rightarrow 2^{V(G)}$ be the order function of (T', τ) . We choose an edge $\{x, y\}$ in T' , put a new vertex v of degree 2 on this edge, and make v adjacent to a new vertex r . By choosing r as a root in the new tree $T = T' \cup \{v, r\}$, we turn T into a rooted tree. For every edge of $f \in E(T) \cap E(T')$, we put $\omega(f) = \omega'(f)$. Also we put $\omega(\{x, v\}) = \omega(\{v, y\}) = \omega'(\{x, y\})$ and $\omega(\{r, v\}) = \emptyset$.

For an edge f of T we define E_f (V_f) as the set of edges (vertices) that are “below” f , i.e., the set of all edges (vertices) g such that every path containing g and $\{v, r\}$ in T contains f . With such a notation, $E(T) = E_{\{v, r\}}$ and $V(T) = V_{\{v, r\}}$. Every edge f of T that is not incident to a leaf has two children that are the edges of E_f incident to f . We denote by G_f the subgraph of G induced by the vertices incident to edges from the following set

$$\{\tau^{-1}(x) \mid x \in V_f \wedge (x \text{ is a leaf of } T')\}.$$

The subproblems in our dynamic program are defined by a coloring of the vertices in $\omega(f)$ for every edge f of T . Each vertex will be assigned one of $2r + 1$ colors

$$\{0, \uparrow 1, \uparrow 2, \dots, \uparrow r, \downarrow 1, \downarrow 2, \dots, \downarrow r\}.$$

The meaning of the color of a vertex v is as follows:

- 0 means that the vertex v is a chosen center.
- $\downarrow i$ means that vertex v has distance exactly i to the closest center c . Moreover, there is a neighbor $u \in V(G_f)$ of v that is at distance exactly $i - 1$ to the center c . We say that neighbor u *resolves* vertex v .
- $\uparrow i$ means that vertex v has distance exactly i to the closest center c . However, there is *no* neighbor of v in $V(G_f)$ resolving v . Thus we are guessing that any vertex resolving v is somewhere in $V(G) - V(G_f)$.

Intuitively, the vertices colored by $\downarrow i$ have already been resolved (though the vertex that resolves it may not itself be resolved), whereas the vertices colored by $\uparrow i$ still need to be assigned vertices that are closer to the center.

We use the notation $\updownarrow i$ to denote a color of either $\uparrow i$ or $\downarrow i$. Also we use $\updownarrow 0 = 0$.

For an edge f of T , a coloring of the vertices in $\omega(f)$ is called *locally valid* if the following property holds: for any two adjacent vertices v and w in $\omega(f)$, if v is colored $\updownarrow i$ and w is colored $\updownarrow j$, then $|i - j| \leq 1$. (If the distance from some

vertex v to the closest center is i , then for every neighbor u of v the distance from u to the closest center can not be less than $i - 1$ or more than $i + 1$.)

For every edge f of T we define the mapping

$$A_f : \{0, \uparrow 1, \uparrow 2, \dots, \uparrow r, \downarrow 1, \downarrow 2, \dots, \downarrow r\}^{|\omega(f)|} \rightarrow \mathbb{N} \cup \{+\infty\}.$$

For a locally valid coloring $c \in \{0, \uparrow 1, \uparrow 2, \dots, \uparrow r, \downarrow 1, \downarrow 2, \dots, \downarrow r\}^{|\omega(f)|}$, the value $A_f(c)$ stores the size of the “minimum (k, r) -center restricted to G_f and coloring c ”. More precisely, $A_f(c)$ is the minimum cardinality of a set $D_f(c) \subseteq V(G_f)$ such that

- For every vertex $v \in \omega(f)$,
 - $c(v) = 0$ if and only if $v \in D_f(c)$, and
 - if $c(v) = \downarrow i$, $i \geq 1$, then $v \notin D_f(c)$ and either there is a vertex $u \in \omega(f)$ colored by $\downarrow j$, $j < i$, at distance $i - j$ from v in G_f , or there is a path P of length i in G_f connecting v with some vertex of $D_f(c)$ such that no inner vertex of P is in $\omega(f)$.
- Every vertex $v \in V(G_f) - \omega(f)$ whose closest center is at distance $i \leq r$, either is at distance i in G_f from some center in $D_f(c)$, or is at distance j , $j < i$, in G_f from a vertex $u \in \omega(f)$ colored $\downarrow(i - j)$.

We put $A_f(c) = +\infty$ if there is no such a set $D_f(c)$, or if c is not a locally valid coloring. Because $\omega(\{r, v\}) = \emptyset$ and $G_{\{r, v\}} = G$, we have that $A_{\{r, v\}}(c)$ is the smallest size of an r -dominating set in G .

We start computations of the functions A_f from leaves of T . Let x be a leaf of T and let f be the edge of T incident with x . Then G_f is the edge of G corresponding to x . We consider all locally valid colorings of $V(G_f)$ such that if a vertex $v \in V(G_f)$ is colored by $\downarrow i$ for $i > 0$ then there is an adjacent vertex w in $V(G_f)$ colored $\downarrow i - 1$. For each such coloring c we define $A_f(c)$ to be the number of vertices colored 0 in $V(G_f)$. Otherwise, $A_f(c)$ is $+\infty$, meaning that this coloring c is infeasible. The brute-force algorithm takes $O(rm)$ time for this step.

Let f be a non-leaf edge of T and let f_1, f_2 be the children of f . Define $X_1 = \omega(f) - \omega(f_2)$, $X_2 = \omega(f) - \omega(f_1)$, $X_3 = \omega(f) \cap \omega(f_1) \cap \omega(f_2)$, and $X_4 = (\omega(f_1) \cup \omega(f_2)) - \omega(f)$.

Notice that

$$\omega(f) = X_1 \cup X_2 \cup X_3. \quad (6)$$

By the definition of ω , it is impossible that a vertex belongs to exactly one of $\omega(f), \omega(f_1), \omega(f_2)$. Therefore, condition $u \in X_4$ implies that $u \in \omega(f_1) \cap \omega(f_2)$ and we conclude that

$$\omega(f_1) = X_1 \cup X_3 \cup X_4, \quad (7)$$

and

$$\omega(f_2) = X_2 \cup X_3 \cup X_4. \quad (8)$$

We say that a coloring $c \in \{0, \uparrow 1, \uparrow 2, \dots, \uparrow r, \downarrow 1, \downarrow 2, \dots, \downarrow r\}^{|\omega(f)|}$ of $\omega(f)$ is *formed* from a coloring c_1 of $\omega(f_1)$ and a coloring c_2 of $\omega(f_2)$ if

1. For every $u \in X_1$, $c(u) = c_1(u)$;
2. For every $u \in X_2$, $c(u) = c_2(u)$;
3. For every $u \in X_3$,
 - a) If $c(u) = \uparrow i$, $1 \leq i \leq r$, then $c(u) = c_1(u) = c_2(u)$. Intuitively, because vertex u is unresolved in $\omega(f)$, this vertex is also unresolved in $\omega(f_1)$ and in $\omega(f_2)$.
 - b) If $c(u) = 0$ then $c_1(u) = c_2(u) = 0$.
 - c) If $c(u) = \downarrow i$, $1 \leq i \leq r$, then $c_1(u), c_2(u) \in \{\downarrow i, \uparrow i\}$ and $c_1(u) \neq c_2(u)$. We avoid the case when both c_1 and c_2 are colored by $\downarrow i$ because it is sufficient to have the vertex u resolved in at least one coloring. This observation helps to decrease the number of colorings forming a coloring c . (Similar arguments using a so-called “monotonicity property” are made by Alber et al. [2] for computing the minimum dominating set on graphs of bounded treewidth.)
4. For every $u \in X_4$,
 - a) either $c_1(u) = c_2(u) = 0$ (in this case we say that u is *formed by 0 colors*),
 - b) or $c_1(u), c_2(u) \in \{\downarrow i, \uparrow i\}$ and $c_1(u) \neq c_2(u)$, $1 \leq i \leq r$ (in this case we say that u is *formed by $\{\downarrow i, \uparrow i\}$ colors*).

This property says that every vertex u of $\omega(f_1)$ and $\omega(f_2)$ that does not appear in $\omega(f)$ (and hence does not appear further) should finally either be a center (if both colors of u in c_1 and c_2 were 0), or should be resolved by some vertex in $V(G_f)$ (if one of the colors $c_1(u), c_2(u)$ is $\downarrow i$ and one $\uparrow i$). Again, we avoid the case of $\downarrow i$ in both c_1 and c_2 .

Notice that every coloring of $\omega(f)$ is formed from some colorings of $\omega(f_1)$ and $\omega(f_2)$. Moreover, if $D_f(c)$ is the restriction to G_f of some (k, r) -center and such a restriction corresponds to a coloring c of $\omega(f)$ then $D_f(c)$ is the union of the restrictions $D_{f_1}(c_1), D_{f_2}(c_2)$ to G_{f_1}, G_{f_2} of two (k, r) -centers where these restrictions correspond to some colorings c_1, c_2 of $\omega(f_1)$ and $\omega(f_2)$ that form the coloring c .

We compute the values of the corresponding functions in a bottom-up fashion. The main observation here is that if f_1 and f_2 are the children of f , then the vertex sets $\omega(f_1)$ $\omega(f_2)$ “separate” subgraphs G_{f_1} and G_{f_2} , so the value $A_f(c)$ can be obtained from the information on colorings of $\omega(f_1)$ and $\omega(f_2)$.

More precisely, let c be a coloring of $\omega(f)$ formed by colorings c_1 and c_2 of f_1 and f_2 . Let $\#_0(X_3, c)$ be the number of vertices in X_3 colored by color 0 in coloring c and let $\#_0(X_4, c)$ be the number of vertices in X_4 formed by 0 colors. For a coloring c we assign

$$A_f(c) = \min\{A_{f_1}(c_1) + A_{f_2}(c_2) - \#_0(X_3, c_1) - \#_0(X_4, c_1) \mid c_1, c_2 \text{ form } c\}. \quad (9)$$

(Every 0 from X_3 and X_4 is counted in $A_{f_1}(c_1) + A_{f_2}(c_2)$ twice and $X_3 \cap X_4 = \emptyset$.) The time to compute the minimum in (9) is given by

$$O\left(\sum_c |\{\{c_1, c_2\} \mid c_1, c_2 \text{ form } c\}|\right).$$

Let $x_i = |X_i|$, $1 \leq i \leq 4$. For a coloring c let z_3 be the number of vertices colored by $\downarrow$ colors in X_3 . Also we denote by z_4 the number of vertices in X_4 formed by $\{\downarrow i, \uparrow i\}$ colors, $1 \leq i \leq r$. Thus the number of pairs forming c is $2^{z_3+z_4}$. The number of colorings of $\omega(f)$ such that exactly z_3 vertices of X_3 are colored by $\downarrow$ colors and such that exactly z_4 vertices of X_4 are formed by $\{\downarrow, \uparrow\}$ colors is

$$(2r+1)^{x_1}(2r+1)^{x_2}(r+1)^{x_3-z_3} \binom{x_3}{z_3} r^{z_3} \binom{x_4}{z_4} r^{z_4}.$$

Thus the number of operations needed to estimate (9) for all possible colorings of $\omega(f)$ is

$$\sum_{p=0}^{x_3} \sum_{q=0}^{x_4} 2^{p+q} (2r+1)^{x_1+x_2} (r+1)^{x_3-p} \binom{x_3}{p} r^p \binom{x_4}{q} r^q = (2r+1)^{x_1+x_2+x_4} (3r+1)^{x_3}.$$

Let ℓ be the branchwidth of G . By (6), (7) and (8),

$$\begin{aligned} x_1 + x_2 + x_3 &\leq \ell \\ x_1 + x_3 + x_4 &\leq \ell \\ x_2 + x_3 + x_4 &\leq \ell. \end{aligned} \tag{10}$$

The maximum value of the linear function $\log_{3r+1}(2r+1) \cdot (x_1+x_2+x_4) + x_3$ subject to the constraints in (10) is $\frac{3\log_{3r+1}(2r+1)}{2}\ell$. (This is because the value of the corresponding LP achieves maximum at $x_1 = x_2 = x_4 = 0.5\ell$, $x_3 = 0$.) Thus one can evaluate (9) in time

$$(2r+1)^{x_1+x_2+x_4} (3r+1)^{x_3} \leq (3r+1)^{\frac{3\log_{3r+1}(2r+1)}{2}\ell} = (2r+1)^{\frac{3}{2}\ell}.$$

It is easy to check that the number of edges in T is $O(m)$ and the time needed to evaluate $A_{\{r,v\}}(c)$ is $O((2r+1)^{\frac{3}{2}\ell}m)$. Moreover, it is easy to modify the algorithm to obtain an optimal choice of centers by bookkeeping the colorings assigned to each set $\omega(f)$.

Summarizing, we obtain the following theorem:

Theorem 5. *For a graph G on m edges and with a given branch decomposition of width $\leq \ell$, and integers k, r , the existence of a (k, r) -center in G can be checked in $O((2r+1)^{\frac{3}{2}\ell}m)$ time and, in case of a positive answer, constructs a (k, r) -center of G in the same time.*

Similar result can be obtained for map graphs.

Theorem 6. *Let H be a witness of a map graph $G_{\mathcal{M}}$ on n vertices and let k, r be integers. If a branch-decomposition of width $\leq \ell$ of H is given, the existence of a (k, r) -center in $G_{\mathcal{M}}$ can be checked in $O((2r+1)^{\frac{3}{2}\ell}n)$ time and, in case of a positive answer, constructs a (k, r) -center of G in the same time.*

Proof. We give a sketch of the proof here. H is bipartite graph with a bipartition $(V(G_{\mathcal{M}}), V(H) - V(G_{\mathcal{M}}))$. There is a (k, r) -center in $G_{\mathcal{M}}$ if and only if H has a set $S \subseteq V(G_{\mathcal{M}})$ of size k such that every vertex $V(G_{\mathcal{M}}) - S$ is at distance $\leq 2r$ in H from some vertex of S . We check whether such a set S exists in H by applying arguments similar the proof of Theorem 5. The main differences in the proof are the following. Now we color vertices of the graph H by $\uparrow i$, $0 \leq i \leq 2r$, where i is even. Thus we are using $2r + 1$ numbers. Because we are not interested whether the vertices of $V(H) - V(G_{\mathcal{M}})$ are dominated or not, for vertices of $V(H) - V(G_{\mathcal{M}})$ we keep the same number as for a vertex of $V(G_{\mathcal{M}})$ resolving this vertex. For a vertex in $V(G_{\mathcal{M}})$ we assign a number $\downarrow i$ if there is a resolving vertex from $V(H) - V(G_{\mathcal{M}})$ colored $\uparrow(i - 2)$. Also we change the definition of locally valid colorings: for any two adjacent vertices v and w in $\omega(f)$, if v is colored $\uparrow i$ and w is colored $\uparrow j$, then $|i - j| \leq 2$.

Finally, H is planar, so $|E(H)| = O(|V(H)|) = O(n)$.

5 Algorithms for the (k, r) -Center Problem

For a planar graph G and integers k, r , we solve (k, r) -center problem on planar graphs in three steps.

Step 1: We check whether the branchwidth of G is at most $4(2r + 1)\sqrt{k} + 8r + 1$. This step requires $O((|V(G)| + |E(G)|)^2)$ time according to the algorithm due to Seymour & Thomas (algorithm (7.3) of Section 7 of [25] — for an implementation, see the results of Hicks [19]). If the answer is negative then report that G has no any (k, r) -center and stop. Otherwise go to the next step.

Step 2: Compute an optimal branch-decomposition of a graph G . This can be done by the algorithm (9.1) in the Section 9 of [25] which requires $O((|V(G)| + |E(G)|)^4)$ steps.

Step 3: Compute, if it exists, a (k, r) -center of G using the dynamic-programming algorithm of Section 4.

It is crucial that, for practical applications, there are no *large hidden constants* in the running time of the algorithms in Steps 1 and 2 above. Because for planar graphs $|E(G)| = O(|V(G)|)$, we conclude with the following theorem:

Theorem 7. *There exists an algorithm finding, if it exists, a (k, r) -center of a planar graph in $O((2r + 1)^{6(2r+1)\sqrt{k}+12r+3/2}n + n^4)$ time.*

Similar arguments can be applied to solve the (k, r) -center problem on map graphs. Let $G_{\mathcal{M}}$ be a map graph. To check whether $G_{\mathcal{M}}$ has a (k, r) -center, we compute optimal branchwidth of its witness H . By Theorem 4, if $\mathbf{bw}(H) > 4(4r + 3)\sqrt{k} + 16r + 9$, then $G_{\mathcal{M}}$ has no (k, r) -center. If $\mathbf{bw}(H) \leq 4(4r + 3)\sqrt{k} + 16r + 9$, then by Theorem 6 we obtain the following result:

Theorem 8. *There exists an algorithm finding, if it exists, a (k, r) -center of a map graph in $O((2r + 1)^{6(4r+1)\sqrt{k}+24r+13.5}n + n^4)$ time.*

By a straightforward modification to the dynamic program, we obtain the same results for the *vertex-weighted* (k, r) -center problem, in which the vertices have real weights and the goal is to find a (k, r) -center of minimum total weight.

6 Concluding Remarks

In this paper, we presented fixed-parameter algorithms with exponential speed-up for the (k, r) -center problem on planar graphs and map graphs. Our methods for (k, r) -center can also be applied to algorithms on more general graph classes like constant powers of planar graphs, which are not minor-closed family of graphs. Extending these results to other non-minor-closed families of graphs would be instructive. Faster algorithms for (k, r) -center for planar graphs and map graphs can be obtained by adopting the proof techniques for planar dominating set from [14]. The disadvantage of this approach is that proofs (but not the algorithm itself) become much more difficult.

In addition, there are several interesting variations on the (k, r) -center problem. In *multiplicity- m* (k, r) -center, the k centers must satisfy the additional constraint that every vertex is within distance r of at least m centers. In *f -fault-tolerant* (k, r) -center [7], every non-center vertex must have f vertex-disjoint paths of length at most r to centers. (For this problem with $r = \infty$, [7] gives a polynomial-time $O(f \log |V|)$ -approximation algorithm for k .) In *L -capacitated* (k, r) -center [7], each of the k centers can satisfy only L “customers”, essentially forcing the assignment of vertices to centers to be load-balanced. (For this problem, [7] gives a polynomial-time $O(\log |V|)$ -approximation algorithm for r .) In *connected* (k, r) -center [26], the k chosen centers must form a connected subgraph. In all these problems, the main challenge is to design the dynamic program on graphs of bounded treewidth/branchwidth. We believe that our approach can be used as the main guideline in this direction.

More generally, it seems that our approach should extend other graph algorithms (not just dominating-set-type problems) to apply to the r th power and/or half-square of a graph (and hence in particular map graphs). It would be interesting to explore to which other problems our approach can be applied. Also, obtaining “fast” algorithms for problems like feedback vertex set or vertex cover on constant powers of graphs of bounded branchwidth (treewidth), as we did for dominating set, would be interesting.

Map graphs can be seen as contact graphs of disc homeomorphs. A question is whether our results can be extended for another geometric classes of graphs. An interesting candidate is the class of unit-disk graphs. The current best algorithms for finding a vertex cover or a dominating set of size k on these graphs have $n^{O(\sqrt{k})}$ running time [4].

To demonstrate the versatility of our approach, notice that a direct consequence of our approach is the following theorem.

Theorem 9. *Let $\mathbf{p}$ be a function mapping graphs to non-negative integers such that the following conditions are satisfied:*

- (1) *There exists an algorithm checking whether $\mathbf{p}(G) \leq w$ in $f(\mathbf{bw}(G))n^{O(1)}$ steps.*
- (2) *For any $k \geq 0$, the class of graphs where $\mathbf{p}(G) \leq k$ is closed under taking of contractions.*
- (3) *If R is any partially triangulated $(j \times j)$ -grid¹ then $\mathbf{p}(R) = \Omega(j^2)$.*

Then there exists an algorithm checking whether $\mathbf{p}(G) \leq k$ on planar graphs in $O(f(\sqrt{k})n^{O(1)})$ steps.

For a wide source of parameters satisfying condition (1) we refer to the theory of Courcelle [10] (see also [5]). For parameters where $f(\mathbf{bw}(G)) = 2^{O(\mathbf{bw}(G))}$, this result is a strong generalization of Alber et al.'s approach which requires that the problem of checking whether $\mathbf{p}(G) \leq k$ should satisfy the “layerwise separation property” [3]. Moreover, the algorithms involved are expected to have better constants in their exponential part comparatively to the ones appearing in [3]. Similar results can also be obtained for constant powers of planar graphs and for map graphs.

Finally, let us note that combining Theorems 5 and 6 with Baker's approach [6] (see also [13] and [15]) adapted to branch decompositions instead of tree decompositions, we are able to obtain a PTAS for r -dominating set on planar and map graphs. We summarize these results in the following theorems:

Theorem 10. *For any integer $p \geq 1$, the r -dominating set problem on planar graphs has a $(1 + 2r/p)$ -approximation algorithm with running time $O(p(2r + 1)^{3(p+2r)m})$.*

Theorem 11. *For any integer $p \geq 1$, the r -dominating set problem on map graphs has a $(1 + 4r/p)$ -approximation algorithm with running time $O(p(4r + 3)^{3(p+4r)m})$.*

References

1. P. K. AGARWAL AND C. M. PROCOPIUC, *Exact and approximation algorithms for clustering*, Algorithmica, 33 (2002), pp. 201–226.
2. 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.
3. J. ALBER, H. FERNAU, AND R. NIEDERMEIER, *Parameterized complexity: Exponential speed-up for planar graph problems*, in Electronic Colloquium on Computational Complexity (ECCC), Germany, 2001.
4. J. ALBER AND J. FIALA, *Geometric separation and exact solutions for the parameterized independent set problem on disk graphs*, in Foundations of Information Technology in the Era of Networking and Mobile Computing, IFIP 17th WCC/TCS'02, Montréal, Canada, vol. 223 of IFIP Conference Proceedings, Kluwer, 2002, pp. 26–37.

¹ A *partially triangulated $(j \times j)$ -grid* is any graph obtained by adding noncrossing edges between pairs of nonconsecutive vertices on a common face of a planar embedding of an $(j \times j)$ -grid.

5. S. ARNBORG, J. LAGERGREN, AND D. SEESE, *Problems easy for tree-decomposable graphs (extended abstract)*, in Automata, languages and programming (Tampere, 1988), Springer, Berlin, 1988, pp. 38–51.
6. B. S. BAKER, *Approximation algorithms for NP-complete problems on planar graphs*, J. Assoc. Comput. Mach., 41 (1994), pp. 153–180.
7. J. BAR-ILAN, G. KORTSARZ, AND D. PELEG, *How to allocate network centers*, J. Algorithms, 15 (1993), pp. 385–415.
8. Z.-Z. CHEN, *Approximation algorithms for independent sets in map graphs*, J. Algorithms, 41 (2001), pp. 20–40.
9. Z.-Z. CHEN, E. GRIGNI, AND C. H. PAPADIMITRIOU, *Map graphs*, J. ACM, 49 (2002), pp. 127–138.
10. B. COURCELLE, *Graph rewriting: an algebraic and logic approach*, in Handbook of theoretical computer science, Vol. B, Elsevier, Amsterdam, 1990, pp. 193–242.
11. 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 Annual International Symposium on Algorithms and Computation—ISAAC 2002 (Vancouver, Canada), Springer, Lecture Notes in Computer Science, Berlin, vol.2518, 2002, pp. 262–273.
12. R. G. DOWNEY AND M. R. FELLOWS, *Parameterized complexity*, Springer-Verlag, New York, 1999.
13. D. EPPSTEIN, *Diameter and treewidth in minor-closed graph families*, Algorithmica, 27 (2000), pp. 275–291.
14. 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, 2003, pp. 168–177.
15. M. FRICK AND M. GROHE, *Deciding first-order properties of locally tree-decomposable structures*, J. Assoc. Comput. Mach., 48 (2001), pp. 1184–1206.
16. C. GAVOILLE, D. PELEG, A. RASPAUD, AND E. SOPENA, *Small k -dominating sets in planar graphs with applications*, in Graph-theoretic concepts in computer science (Boltenhagen, 2001), vol. 2204 of Lecture Notes in Comput. Sci., Springer, Berlin, 2001, pp. 201–216.
17. T. F. GONZALEZ, *Clustering to minimize the maximum intercluster distance*, Theoret. Comput. Sci., 38 (1985), pp. 293–306.
18. A. GUPTA AND N. NISHIMURA, *Sequential and parallel algorithms for embedding problems on classes of partial k -trees*, in Algorithm theory—SWAT '94 (Aarhus, 1994), vol. 824 of Lecture Notes in Comput. Sci., Springer, Berlin, 1994, pp. 172–182.
19. I. V. HICKS, *Branch Decompositions and their applications*, PhD thesis, Rice University, 2000.
20. 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.
21. 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.
22. J. PLESNÍK, *On the computational complexity of centers locating in a graph*, Apl. Mat., 25 (1980), pp. 445–452. With a loose Russian summary.
23. N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. X. Obstructions to tree-decomposition*, J. Combin. Theory Ser. B, 52 (1991), pp. 153–190.

24. N. ROBERTSON, P. D. SEYMOUR, AND R. THOMAS, *Quickly excluding a planar graph*, J. Combin. Theory Ser. B, 62 (1994), pp. 323–348.
25. P. D. SEYMOUR AND R. THOMAS, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241.
26. C. SWAMY AND A. KUMAR, *Primal-dual algorithms for connected facility location problems*, in Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, vol. 2462 of Lecture Notes in Computer Science, Rome, Italy, September 2002, pp. 256–270.
27. M. THORUP, *Map graphs in polynomial time*, in The 39th Annual Symposium on Foundations of Computer Science (FOCS 1998), IEEE Computer Society, 1998, pp. 396–405.

Genus Characterizes the Complexity of Graph Problems: Some Tight Results

Jianer Chen¹, Iyad A. Kanj², Ljubomir Perković², Eric Sedgwick², and Ge Xia¹

¹ Department of Computer Science, Texas A&M University, College Station, TX 77843-3112.*** {chen,gexia@cs.tamu.edu}

² School of Computer Science, Telecommunications and Information Systems, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604-2301. {ikanj,lperkovic,esedgwick@cs.depaul.edu}

Abstract. We study the fixed-parameter tractability, subexponential time computability, and approximability of the well-known NP-hard problems: INDEPENDENT SET, VERTEX COVER, and DOMINATING SET. We derive tight results and show that the computational complexity of these problems, with respect to the above complexity measures, is dependent on the genus of the underlying graph. For instance, we show that, under the widely-believed complexity assumption $W[1] \neq \text{FPT}$, INDEPENDENT SET on graphs of genus bounded by $g_1(n)$ is fixed parameter tractable if and only if $g_1(n) = o(n^2)$, and DOMINATING SET on graphs of genus bounded by $g_2(n)$ is fixed parameter tractable if and only if $g_2(n) = n^{o(1)}$. Under the assumption that not all SNP problems are solvable in subexponential time, we show that the above three problems on graphs of genus bounded by $g_3(n)$ are solvable in subexponential time if and only if $g_3(n) = o(n)$.

1 Introduction

NP-completeness theory [13] serves as a foundation for the study of intractable computational problems. However, this theory does not obviate the need for solving these hard problems because of their practical importance. Many approaches have been proposed to solve these problems, including polynomial time approximation, fixed parameter tractable computation, and subexponential time algorithms. The INDEPENDENT SET, VERTEX COVER, and DOMINATING SET problems are among the celebrated examples of such problems. Unfortunately, these problems refuse to give in to most of these approaches. Recent research has shown [3] that none of them has a polynomial time approximation scheme unless $P = NP$. It is also unlikely that any of them is solvable in subexponential time [15]. In terms of fixed parameter tractability, INDEPENDENT SET and DOMINATING SET do not seem to have efficient algorithms even for small parameter values [11].

Variants of these problems where the input graph is constrained to have certain structural properties (bounded degree graphs, planar graphs, unit disk

*** This research is supported in part by the NSF under the grant CCR-0000206.

graphs, etc ...) were studied as well [1,2,4,12,13]. In particular, if we consider the above problems on the class of planar graphs (the problems remain NP-complete), they become more tractable in terms of the above three complexity measures. All of the three problems on planar graphs have polynomial time approximation schemes [5,16], and are solvable in subexponential time [16]. Recent research in fixed parameter tractability shows that all the three problems admit parameterized algorithms whose running time is subexponential in the parameter [2]. Very Recently, Ellis et al. showed that the DOMINATING SET problem on graphs of constant genus is fixed parameter tractable [12].

This raises an interesting question: What are the graph structures that determine the computational complexity of these important NP-hard problems?

In this paper, we demonstrate how the genus of the underlying graph plays an important role in characterizing the parameterized complexity, the subexponential time computability, and the approximability of the VERTEX COVER, INDEPENDENT SET, and DOMINATING SET problems. Our research shows that in most cases, graph genus is the sole factor that determines the complexity of the above problems. More precisely, in most cases, there is a precise genus threshold that determines the computational complexity of the problems in terms of the three complexity measures. For instance, we show that under the widely-believed complexity assumption $W[2] \neq \text{FPT}$, DOMINATING SET is fixed parameter tractable if and only if the graph genus is $n^{o(1)}$. This result significantly extends both Alber et al. and Ellis et al.'s results for planar graphs and for constant genus graphs [1,12]. The proof is also simpler and more uniform. It is also shown that under the assumption $W[1] \neq \text{FPT}$, INDEPENDENT SET is fixed parameter tractable if and only if the graph genus is $o(n^2)$. For the subexponential time computability, we show that under the assumption that not all SNP problems are solvable in subexponential time, VERTEX COVER, INDEPENDENT SET, and DOMINATING SET are solvable in subexponential time if and only if the genus of the graph is $o(n)$. In terms of approximability, we show that INDEPENDENT SET has a PTAS on graphs of genus $o(n/\lg n)$, but has no PTAS on graphs of genus $\Omega(n)$ unless $P = NP$. It is also shown that, unless $P = NP$, the VERTEX COVER and DOMINATING SET problems on graphs of genus $n^{\Omega(1)}$ have no PTAS. A summary of our main results and the previous known results is given in Table 1. Finally, we point out that our techniques can be extended to derive similar results for other NP-hard graph problems. Due to lack of space, the proofs of some results in the paper will be omitted.

We give a quick review on the terminologies related to this paper. Let G be a graph. A set of vertices C in the graph G is a *vertex cover* for G if every edge in G is incident to at least one vertex in C . An *independent set* I in G is a subset of vertices in G such that no two vertices in I are adjacent. A *dominating set* D in G is a set of vertices in G such that every vertex in G is either in D or adjacent to a vertex in D .

A *surface* of genus g is a sphere with g handles in the 3-space [14]. A graph G *embedded* in a surface S is a continuous one-to-one mapping from the graph into the surface. The embedding is *cellular* if each component of $S - G$, which

Table 1. Comparison between our results and the previous results

Prob.	FPT		Subexp. Time		Approximability	
	Ours	Previous	Ours	Previous	Ours	Previous
VC	–	FPT [11,16]	$2^{o(n)}$ iff $g=o(n)$	$2^{O(\sqrt{n})}$ if $g=c$ [2,16]	APX-C if $g=n^{\Omega(1)}$	PTAS if $g=c$ [5,16]
IS	FPT iff $g=o(n^2)$	FPT if $g=0$ [2]	$2^{o(n)}$ iff $g=o(n)$	$2^{O(\sqrt{n})}$ if $g=c$ [2,16]	PTAS if $g=o(\frac{n}{\log n})$ APX-H if $g=\Omega(n)$	PTAS if $g=c$ [5,16]
DS	FPT iff $g=n^{o(1)}$	FPT if $g=c$ [12]	$2^{o(n)}$ iff $g=o(n)$	$2^{O(\sqrt{n})}$ if $g=c$ [2,16]	APX-H if $g=n^{\Omega(1)}$	PTAS if $g=c$ [5,16]

is called a *face*, is homeomorphic to an open disk [14]. In this paper, we only consider cellular graph embeddings. The *size* of a face is the number of edge sides along the boundary of the face. The (*minimum*) *genus* $\gamma_{\min}(G)$ of a graph G is the smallest integer g such that G has an embedding on a surface of genus g . For more detailed discussions on data structures and algorithms for graph embedding on surfaces, the readers are referred to [7].

2 Genus and Parameterized Complexity

A *parameterized problem* consists of instances of the form (x, k) , where x is the *problem description* and k is an integer called the *parameter*. For instance, the VERTEX COVER problem can be parameterized so that each instance of it is of the form (G, k) , where G is a graph and k is the parameter, asking whether the graph G has a vertex cover of k vertices. Similarly, we can define the parameterized versions for INDEPENDENT SET and DOMINATING SET. A parameterized problem Q is *fixed parameter tractable* if it can be solved by an algorithm of running time $O(f(k)n^c)$, where f is a function independent of $n = |x|$ and c is a fixed constant [11]. Denote by FPT the class of all fixed parameter tractable problems. An example of the FPT problems is the VERTEX COVER problem that can be solved in time $O(1.285^k + kn)$ [9]. On the other hand, a large class of computational problems seems to not belong to FPT [11]. A hierarchy of parameterized intractability, the W -hierarchy, has been introduced. The 0th level of the hierarchy is the class FPT, and the i th level is denoted by $W[i]$ for $i > 0$ [11]. The hardness and completeness under a parameterized complexity preserving reduction (the *FPT-reduction*) have been defined for each level $W[i]$ in the W -hierarchy [11]. In particular, INDEPENDENT SET is $W[1]$ -complete and DOMINATING SET is $W[2]$ -complete. It is widely believed that no $W[i]$ -hard problem is in the class FPT [11].

2.1 Genus and INDEPENDENT SET

We start by considering the parameterized complexity for the INDEPENDENT SET problem on graphs with genus constraints. A graph G is *p -colorable* if the vertices of G can be colored with p colors such that no two adjacent vertices are

colored with the same color. The *chromatic number* $\chi(G)$ of G is the smallest integer p such that G is p -colorable.

Theorem 1. *The INDEPENDENT SET problem on graphs of genus bounded by $g(n)$ is fixed parameter tractable if $g(n) = o(n^2)$.*

Proof. Since $g(n) = o(n^2)$, there is a nondecreasing and unbounded function $r(n)$ such that $g(n) \leq n^2/r(n)$.¹ Without loss of generality, we can assume that $r(n) \leq n^2$ since otherwise $g(n) = 0$ and the theorem follows from [2]. Let G be a graph of n vertices and genus $g' \leq g(n)$. By Heawood's Theorem [14], the chromatic number $\chi(G)$ of the graph G is bounded by $(7 + \sqrt{1 + 48g'})/2$. From the definition, the chromatic number $\chi(G)$ of G implies an independent set of at least $n/\chi(G)$ vertices in G . Thus, the size $\alpha(G)$ of a maximum independent set in the graph G is at least $2n/(7 + \sqrt{1 + 48g'})$. Since $g' \leq g(n) \leq n^2/r(n)$, we get (note that $r(n) \leq n^2$)

$$\alpha(G) \geq \frac{2n}{7 + \sqrt{1 + 48n^2/r(n)}} \geq \frac{2n\sqrt{r(n)}}{7n + \sqrt{n^2 + 48n^2}} = \frac{\sqrt{r(n)}}{7} \quad (1)$$

Now we are ready for describing our parameterized algorithm. Note that one difficulty we must overcome is estimating the genus of the input graph. The graph minimum genus problem is NP-complete [18] and there is no known effective approximation algorithm for the problem. Therefore, some special tricks have to be used for this purpose. Here we will make use of the approximation algorithm for the graph minimum genus problem proposed in [8], which on an input graph G constructs an embedding of G whose genus is bounded by $\max\{4\gamma_{\min}(G), \gamma_{\min}(G) + 4n\}$. Consider the algorithm given in Figure 1.

ALGORITHM. IS-FPT

Input: a graph G of n vertices and an integer k

Output: decide if G has an independent set of k vertices

1. let $r_1(n) = \min\{r(n)/4, nr(n)/(n + 4r(n))\}$;
2. construct an embedding $\pi(G)$ of G using the algorithm in [8];
3. **if** the genus of $\pi(G)$ is larger than $n^2/r_1(n)$ **then**
 Stop ("the genus of G is larger than $g(n)$ ");
4. **if** $k \leq \sqrt{r_1(n)}/7$ **then**
 Stop ("the graph G has an independent set of k vertices")
 else try all vertex subsets of k vertices to derive a conclusion.

Fig. 1. A parameterized algorithm for INDEPENDENT SET

We analyze the complexity of the algorithm **IS-FPT**. First note that by our assumption on the function $r(n)$, the function $r_1(n)$ is also nondecreasing and

¹ In this paper, we only consider "simple" complexity functions whose value can be feasibly computed. Thus, in our discussion, the computational time for computing the values of complexity functions as such $g(n)$ and $r(n)$ will be neglected.

unbounded. The embedding $\pi(G)$ of the graph G in step 2 can be constructed in linear time [8], and the genus of the embedding $\pi(G)$ can also be computed in linear time [7].

Since $r_1(n) = \min\{r(n)/4, nr(n)/(n+4r(n))\}$, if the genus $\gamma(\pi(G))$ of the embedding $\pi(G)$ is larger than $n^2/r_1(n)$, then $\gamma(\pi(G))$ is larger than both $4n^2/r(n)$ and $n^2/r(n) + 4n$. According to [8], the genus $\gamma(\pi(G))$ of the embedding $\pi(G)$ is bounded by $\max\{4\gamma_{\min}(G), \gamma_{\min}(G) + 4n\}$. Thus, in case $\gamma(\pi(G)) \leq 4\gamma_{\min}(G)$, we have $4\gamma_{\min}(G) > 4n^2/r(n)$, and in case $\gamma(\pi(G)) \leq \gamma_{\min}(G) + 4n$, we have $\gamma_{\min}(G) + 4n > n^2/r(n) + 4n$. Thus, in all cases, we will have $\gamma_{\min}(G) > n^2/r(n) \geq g(n)$. In consequence, the algorithm **IS-FPT** concludes correctly if it stops at step 3.

If the algorithm **IS-FPT** reaches step 4, we know that the minimum genus of the graph G is bounded by $n^2/r_1(n)$. By the analysis above and the relation in (1), the size of a maximum independent set in G is at least $\sqrt{r_1(n)}/7$. Thus, in case $k \leq \sqrt{r_1(n)}/7$, there must be an independent set in G with k vertices. On the other hand, if $k > \sqrt{r_1(n)}/7$ then $\bar{r}_1(49k^2) \geq n$, where $\bar{r}_1$ is the inverse function of the function $r_1(n)$ defined by $\bar{r}_1(p) = \min\{q \mid r_1(q) \geq p\}$. Since the function $r_1(n)$ is nondecreasing and unbounded, it is not difficult to see that the inverse function $\bar{r}_1(p)$ is also nondecreasing and unbounded. Since enumerating all vertex subsets of k vertices in the graph G can be done in $O(2^n)$ time, which is bounded by $O(2^{\bar{r}_1(49k^2)})$, we conclude that the total running time of the algorithm **IS-FPT** is bounded by $O(f(k) + n^2)$, where $f(k) = 2^{\bar{r}_1(49k^2)}$ is a function dependent only on k but not on n .

Thus, the algorithm **IS-FPT** solves the INDEPENDENT SET problem on graphs of genus bounded by $g(n)$ in time $O(f(k) + n^2)$, and the problem is fixed parameter tractable.

Remark. The algorithm **IS-FPT** does not have to know whether the input graph has its minimum genus bounded by $g(n)$. The point is, if the input graph has its minimum genus bounded by $g(n)$, then the algorithm **IS-FPT**, without needing to know this fact, will definitely and correctly decide whether it has an independent set of size k .

Theorem 2. *The INDEPENDENT SET problem on graphs of genus bounded by $g(n)$ is $W[1]$ -complete if $g(n) = \Omega(n^2)$.*

Combining Theorem 1 and Theorem 2, and noting that the genus of a graph of n vertices is always bounded by $(n-3)(n-4)/12$ [14], we have the following tight result.

Corollary 1. *Assuming $FPT \neq W[1]$, the INDEPENDENT SET problem on graphs of genus bounded by $g(n)$ is not fixed parameter tractable if and only if $g(n) = \Theta(n^2)$.*

2.2 Genus and DOMINATING SET

We now discuss how graph genus affects the parameterized complexity of the DOMINATING SET problem. Efficient algorithms for DOMINATING SET on graphs

of lower genus have been a recent focus in the study of parameterized computation. In particular, it is known that DOMINATING SET on planar graphs [1, 2] and on graphs of constant genus [12] is fixed parameter tractable. We will show a much stronger result: the DOMINATING SET problem on graphs of genus bounded by $g(n)$ is fixed-parameter tractable if and only if $g(n) = n^{o(1)}$.

For a given instance (G, k) of the DOMINATING SET problem, we apply the branch-and-bound search process to construct a dominating set D of k vertices in G . Initially, we have $D = \emptyset$, and all vertices of G are not yet dominated by vertices in D . In a more general form during the search process, we have included certain vertices in the dominating set D , and removed these vertices from the graph G . The remaining graph G' consists of white and black vertices, corresponding to the vertices that are dominated by vertices in D and the vertices that are still not yet dominated by vertices in D . The graph G' thus will be called a *BW-graph*. We call a set D' of vertices in the BW-graph G' a *B-dominating set* if every black vertex in G' is either in D' or is adjacent to a vertex in D' . Thus, our task is to construct a B-dominating set of k' vertices in the BW-graph G' , where k' plus the number of vertices in D is equal to k .

Certain reduction rules can be applied to a BW-graph G' :

- R1.** Remove from G' all edges between white vertices;
- R2.** Remove from G' all white vertices of degree 1;
- R3.** If all neighbors of a white vertex u_1 are neighbors of another white vertex u_2 , remove u_1 from G' .

It can be verified that there is a B-dominating set of k vertices in the graph before applying any of these rules if and only if there is a B-dominating set of k vertices in the graph after applying the rule [1,12]. A BW-graph G is called *reduced* if none of the above rules can be applied. According to rule **R1**, every edge in a reduced BW-graph either connects two black vertices or connects a black vertex and a white vertex (the edge will be called a *bb-edge* or a *bw-edge*, respectively).

Lemma 1. *Let G be a reduced BW-graph with n vertices, in which n_w are white and n_b are black, m edges, and minimum genus g , and suppose that G has neither multiple edges nor self-loops, then (a) $m \leq 9n_b + 18g - 18$; and (b) $n \leq 4n_b + 6g - 6$.*

Theorem 3. *The DOMINATING SET problem on graphs of genus bounded by $g(n)$ is fixed parameter tractable if $g(n) = n^{o(1)}$.*

Proof. Since $g(n) = n^{o(1)}$, we can write $g(n) \leq n^{1/r(n)}$ for some nondecreasing and unbounded function $r(n)$. For an instance (G, k) of the DOMINATING SET problem, where the graph G has n vertices and genus g' , we apply the algorithm **DS-FPT** in Figure 2.

Let $\bar{r}$ be the inverse function of the function $r(n)$ defined by $\bar{r}(p) = \min\{q \mid r(q) \geq p\}$. Then the function $\bar{r}$ is also nondecreasing and unbounded. In case $k \geq r(n)$, we have $\bar{r}(k) \geq n$. Thus, step 1 of the algorithm DS-FPT takes time $O(2^n) = O(2^{\bar{r}(k)})$.

ALGORITHM. DS-FPT

Input: a graph G of n vertices and an integer k

Output: decide if G has a dominating set of k vertices

1. **if** $k \geq r(n)$ **then**
 solve the problem by enumerating all subsets of k vertices in G ; Stop;
2. $k_0 = k$; $D = \emptyset$; $G_0 = G$; color all vertices of G_0 black;
3. **while** there is a black vertex u of degree $d \leq 19$ in G_0 **do**
 make a $(d + 1)$ -way branch each including either u or a neighbor of u in D ;
 remove the new vertex in D from G_0 and color its neighbors in G_0 white;
 apply rules R1-R3 to make G_0 a reduced BW-graph;
 $k_0 = k_0 - 1$;
4. **if** the graph G_0 has at most $78n^{1/k}$ vertices **then**
 find a B-dominating set of k_0 vertices in G_0 by enumerating all vertex subsets of k_0 vertices in G_0
 else Stop ("the graph G has genus larger than $g(n)$ ");

Fig. 2. A parameterized algorithm for DOMINATING SET

Now suppose $k < r(n)$, step 3 repeatedly branches at a black vertex of degree bounded by 19 in the reduced BW-graph G_0 . The search tree size $T(k)$ of step 3 thus satisfies the recurrence relation

$$T(k) \leq 20 \cdot T(k - 1)$$

which has a solution $T(k) = O(20^k)$.

At the end of step 3, all black vertices in the reduced BW-graph G_0 have degree at least 20. Suppose at this point, the number of edges, the number of vertices, and the number of black vertices in G_0 are m_0 , n_0 and n_b , respectively. Since $2m_0$ is equal to the sum of total vertex degrees in G_0 , we have $2m_0 \geq 20n_b$. By Lemma 1, we also have $m_0 \leq 9n_b + 18g' - 18$ (note that the genus of the reduced BW-graph G_0 cannot be larger than the genus g' of the original graph G). Combining these two relations, we get $n_b \leq 18g' - 18$. Now again by Lemma 1, we have $n_0 \leq 4n_b + 6g' - 6$. Thus

$$n_0 \leq 4n_b + 6g' - 6 \leq 78g' - 78 < 78g'$$

Thus, if $g' \leq g(n) \leq n^{1/r(n)} < n^{1/k}$ (note $k < r(n)$), then the number n_0 of vertices in the graph G_0 must be bounded by $78n^{1/k}$. In this case, step 4 solves the problem in time $O(n_{k_0+1}^{k_0+1}) = O((n^{1/k})^k) = O(n)$. On the other hand, if G_0 has more than $78n^{1/k}$ vertices, then again step 4 concludes correctly that the genus of the input graph G is larger than $g(n)$.

In conclusion, the algorithm **DS-FPT** solves the DOMINATING SET problem on graphs of genus bounded by $g(n)$ in time $O(2^{\bar{r}(k)} + 20^k + n)$, and the problem is fixed parameter tractable.

We point out that the techniques used in Theorem 3 are simpler, more uniform, and derive much stronger results than those given in [1,12]. Also, similarly to the algorithm **IS-FPT**, the algorithm **DS-FPT** does not have to know

whether the input graph has minimum genus bounded by $g(n)$. For any graph of minimum genus bounded by $g(n)$, the algorithm will definitely derive a correct conclusion.

Theorem 4. *The DOMINATING SET problem on graphs of genus bounded by $g(n)$ is $W[2]$ -complete if $g(n) = n^{\Omega(1)}$.*

Combining Theorem 3 and Theorem 4, we derive the following tight result.

Corollary 2. *Assuming $FPT \neq W[2]$, the DOMINATING SET problem on graphs of genus bounded by $g(n)$ is fixed parameter tractable if and only if $g(n) = n^{o(1)}$.*

3 Genus and Subexponential Time Complexity

We say that a problem can be solved in *sublinear exponential time* (or shortly *subexponential time*) if it can be solved in time $O(2^{o(n)})$. Lipton and Tarjan used their planar graph separator theorem to show that a class of NP-hard planar graph problems, including VERTEX COVER, INDEPENDENT SET, and DOMINATING SET, are solvable in subexponential time [16]. They also described how their results can be extended to graphs of constant genus [16]. Recently, deriving lower bounds on the precise complexity of NP-hard problems has been attracting more and more attention [6,15]. In particular, Impagliazzo, Paturi, and Zane introduced the concept of SERF-reduction and showed that many well-known NP-hard problems are SERF-complete for the class SNP [15,17]. This implies that if any of these problems is solvable in subexponential time, then so are all problems in the class SNP, which seems quite unlikely.

In this section, we demonstrate how graph genus affects the subexponential time computability for the problems VERTEX COVER, INDEPENDENT SET, and DOMINATING SET. Our algorithmic results in this section extend Lipton and Tarjan's results on planar graphs and graphs of constant genus [16], and our lower bound results refine Impagliazzo, Paturi, and Zane's results on general graphs [15].

Proposition 1 ([10]). *Let $G = (V, E)$ be a graph of n vertices and genus g . There is a linear time algorithm that partitions V into three sets A, B, C , such that C separates A and B , $|A|, |B| \leq n/2$, $|C| \leq c\sqrt{(g' + 1)n}$, where c is a fixed constant and $0 \leq g' \leq g$, and the graph induced by $A \cup B$ has genus bounded by $g - g'$.*

Theorem 5. *The problems VERTEX COVER, INDEPENDENT SET, and DOMINATING SET on graphs of genus bounded $g(n)$ are solvable in subexponential time if $g(n) = o(n)$.*

Proof. We first give a detailed description of our proof for the DOMINATING SET problem. Again, during the search for a minimum dominating set D in a graph G , we classify the vertices in G into five groups (instead of three groups as in Subsection 2.2):

- (1) dominating vertices, which have been included in D ;
- (2) dominated vertices, which should not be in D and are dominated by vertices in D ;
- (3) white vertices, which are not in D but dominated by vertices in D ;
- (4) black vertices, which are not in D and neither yet dominated by vertices in D ;
- (5) red vertices, which should not be in D and are not yet dominated by vertices in D .

During our search process, the dominating vertices and dominated vertices are removed from the graph. Thus, the remaining graph consists of only black, red, and white vertices. Such a graph G will be called a *BWR-graph*. We will use Proposition 5 to partition the vertices of G into the three vertex subsets A , B , and C . Then we consider all possible assignments on the vertices in the set C . Each vertex u in C has the following possibilities:

- u is a white vertex. Then either u is in D or u is not in D ;
- u is a red vertex. Then u must be dominated by either a vertex in C , or by a vertex in A , or by a vertex in B ;
- u is a black vertex. Then either u is in D , or u is not in D , thus must be dominated by a vertex in C , by a vertex in A , or by a vertex in B .

Thus an assignment to the vertices in C can be as follows: each white vertex is assigned either “in- D ” or “not-in- D ”, each red vertex is assigned either “in- A ” or “in- B ”, and each black vertex is assigned either “in- D ”, “in- A ”, or “in- B ”. After this assignment, a white vertex will become either a dominating vertex (if it is “in- D ”) or a dominated vertex (if it is “not-in- D ”), and thus will be removed from the graph; a red vertex adjacent to an “in- D ” vertex in C will become a dominated vertex and will be removed from the graph (in this case, the assignment to the red vertex is ignored); a red vertex not adjacent to any “in- D ” vertex in C will become a red vertex and will be added to the subgraph induced by either A or B (depending on whether it is an “in- A ” or “in- B ” vertex); a “in- D ” black vertex will become a dominating vertex and will be removed from the graph G ; a black vertex whose status is either “in- A ” or “in- B ” and is adjacent to a “in- D ” vertex in C will become a dominated vertex, and will be removed from the graph; finally, an “in- A ” black vertex (resp. an “in- B ” black vertex) not adjacent to any “in- D ” vertex in C will become a red vertex and will be added to the subgraph induced by A (resp. by B).

Since the set C separates the subgraphs induced by the sets A and B , it is not difficult to see that an assignment to the vertices in the set C will result in two separated BWR-subgraphs of G , one is induced by the set A plus certain vertices in C (we will call it the *A-subgraph*), and the other is induced by the set B plus some other vertices in C (we will call it the *B-subgraph*). Thus, the search process can be executed recursively on the *A-subgraph* and the *B-subgraph*.

We analyze the above algorithm. First note that the genus of a subgraph is always bounded by that of the original graph. Therefore, if the original graph has its genus bounded by $g(n)$, then all recursive calls of the algorithm are on

graphs of genus bounded by $g(n)$. Thus, according to Proposition 5, the number of vertices in the set C constructed in each recursive call of the algorithm must be bounded by $c\sqrt{(g(n)+1)n}$.

The algorithm enumerates all possible assignments to the vertices in C . Since each vertex in C can get at most 3 different statuses, the total number of assignments to the vertices in C is bounded by $3^{|C|} \leq 3^{c\sqrt{(g(n)+1)n}}$. For each such assignment, the algorithm recursively works on the induced A -subgraph and B -subgraph. Since $|A|, |B| \leq n/2$, and $|C| \leq c\sqrt{(g(n)+1)n}$, the total number of vertices in each of the A -subgraph and the B -subgraph is bounded by $n/2 + c\sqrt{(g(n)+1)n}$, which is bounded by $2n/3$ when n is larger than a fixed constant. Therefore, the time complexity $T(n)$ of the algorithm is given by the recurrence relation

$$T(n) \leq 3^{c\sqrt{(g(n)+1)n}} \cdot 2T(2n/3) \leq 3^{c\sqrt{(g(n)+1)n}+1} T(2n/3)$$

From this and the fact that $g(n) = o(n)$, we can easily derive that $T(n) = 2^{o(n)}$, thus proving that for graphs of genus bounded by $g(n) = o(n)$, the DOMINATING SET problem can be solved in subexponential time.

The discussion on VERTEX COVER and INDEPENDENT SET is similar, and thus omitted.

Theorem 6. *For any function $g(n) = \Omega(n)$, if one of VERTEX COVER, INDEPENDENT SET, and DOMINATING SET problems on graphs of genus bounded by $g(n)$ can be solved in subexponential time, then all problems in the class SNP can be solved in subexponential time.*

The class SNP contains many well-known NP-hard problems [15] including: k -SAT, k -COLORABILITY, k -SET COVER, VERTEX COVER, and INDEPENDENT SET. It is widely believed among researchers that it is quite unlikely that all problems in SNP are solvable in subexponential time. Based on this, and combining Theorem 5 and Theorem 6, we have the following tight results.

Corollary 3. *Assuming that not all the problems in SNP are solvable in subexponential time, the VERTEX COVER, INDEPENDENT SET, and DOMINATING SET problems on graphs of genus bounded by $g(n)$ are solvable in subexponential time if and only if $g(n) = o(n)$.*

4 Genus and Approximability

The reader is referred to [4,13] for the basic definitions and terminology of approximation algorithms.

Proposition 2 ([10]). *There is an $O(n \log g)$ time algorithm that for a given graph G of n vertices and genus g constructs a subset P of at most $c \cdot \sqrt{gn \log g}$ vertices, where c is a fixed constant, such that removing the vertices in P from G results in a planar graph.*

Theorem 7. *The INDEPENDENT SET problem on graphs of genus bounded by $g(n)$ has a PTAS if $g(n) = o(n/\log n)$.*

Proof. Let $g(n) \leq n/(r(n) \log n)$, where $r(n)$ is a nondecreasing and unbounded function. Our PTAS for the INDEPENDENT SET works as follows: for a given graph G of n vertices, we use the algorithm in Proposition 2 to construct the vertex subset P (this can be done in time $O(n \log n)$ even when the genus of G is larger than $g(n)$). If the number p_0 of vertices in P is larger than $c \cdot \sqrt{g(n)n \log g(n)}$, then we know that the input graph G has genus larger than $g(n)$ and we stop. Otherwise, the graph G_1 obtained by deleting the vertices in P from the graph G is a planar graph. We apply any known PTAS algorithm (e.g., those given in [5,16]) to construct an independent set I_1 for the graph G_1 . The set I_1 is clearly an independent set in the original graph G . Thus, we simply output I_1 as a solution to the graph G .

It is obvious that the above algorithm runs in polynomial time and is an approximation algorithm for the INDEPENDENT SET problem on graphs of genus bounded by $g(n)$. What left is to analyze the approximation ratio of the algorithm. First note that because $g(n) \leq n/(r(n) \log n)$, the number p_0 of vertices in P is bounded by $p_0 \leq c \cdot \sqrt{g(n)n \log g(n)} \leq cn/\sqrt{r(n)}$. Let n_1 be the number of vertices in the graph G_1 , then $n_1 = n - p_0$. Let α and α_1 be the sizes of a maximum independent set in the graphs G and G_1 , respectively. We have $\alpha_1 \leq \alpha \leq \alpha_1 + p_0$ (the second inequality is true because any maximum independent set in G minus the vertices in P makes an independent set in G_1). Moreover, because G_1 is a planar graph, by the Four-Color theorem [14], α_1 is at least $n_1/4$.

Let α'_1 be the number of vertices in the independent set I_1 . Since the independent set I_1 is constructed by a PTAS on the planar graph G_1 , we have $\alpha_1/\alpha'_1 \leq 1 + \epsilon$, where ϵ is the given error bound. Since the function $r(n)$ is nondecreasing and unbounded, there is a constant N_0 such that when $n \geq N_0$, we have

$$\frac{c}{4\sqrt{r(n)}} \leq \frac{1}{8} \quad \text{and} \quad \frac{8c(1+\epsilon)}{\sqrt{r(n)}} \leq \epsilon \quad (2)$$

From the first inequality, we get

$$\begin{aligned} \alpha'_1 &\geq \frac{\alpha_1}{1+\epsilon} \geq \frac{n_1}{4(1+\epsilon)} = \frac{n-p_0}{4(1+\epsilon)} \geq \frac{n-cn/\sqrt{r(n)}}{4(1+\epsilon)} \\ &= n \cdot \left(\frac{1}{4(1+\epsilon)} - \frac{c}{4(1+\epsilon)\sqrt{r(n)}} \right) \geq \frac{n}{8(1+\epsilon)} \end{aligned} \quad (3)$$

Since $\alpha \leq \alpha_1 + p_0 \leq (1+\epsilon)\alpha'_1 + cn/\sqrt{r(n)}$, combining this with (2) and (3), we get

$$\frac{\alpha}{\alpha'_1} \leq 1 + \epsilon + \frac{cn}{\alpha'_1 \sqrt{r(n)}} \leq 1 + \epsilon + \frac{8cn(1+\epsilon)}{n\sqrt{r(n)}} \leq 1 + 2\epsilon$$

This shows that our algorithm is a PTAS for the INDEPENDENT SET problem on graphs of genus bounded by $g(n)$.

Theorem 8. *Assuming $P \neq NP$, the INDEPENDENT SET problem on graphs of genus bounded by $g(n)$ has no PTAS if $g(n) = \Omega(n)$.*

Unfortunately, the analogous theorems to Theorem 7 do not hold for VERTEX COVER and DOMINATING SET.

Theorem 9. *Unless $P = NP$, VERTEX COVER and DOMINATING SET on graphs of genus bounded by $g(n)$ have no PTAS if $g(n) = n^{\Omega(1)}$.*

References

1. J. ALBER, H. FAN, M. FELLOWS, H. FERNAU, R. NIEDERMEIER, F. ROSAMOND, AND U. STEGE, Refined search tree technique for dominating set on planar graphs, *LNCS* **2136**, (2001), pp. 111–122.
2. J. ALBER, H. FERNAU, AND R. NIEDERMEIER, Parameterized complexity: exponential speedup for planar graph problems, *LNCS* **2076**, (2001), pp. 261–272.
3. S. ARORA, C. LUND, R. MOTWANI, M. SUDAN, AND M. SZEGEDY, Proof verification and hardness of approximation problems, *J. ACM* **45**, (1998), pp. 501–555.
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, Berlin Heidelberg, 1999.
5. B. BAKER, Approximation algorithms for NP-complete problems on planar graphs, *J. ACM* **41**, (1994), pp. 153–180.
6. L. CAI AND D. JUEDES, On the existence of subexponential-time parameterized algorithms, *JCSS*, to appear.
7. J. CHEN, Algorithmic graph embeddings, *TCS* **181**, (1987), pp. 247–266.
8. J. CHEN, S. KANCHI, AND A. KANEVSKY, A note on approximating graph genus, *Information Processing Letters* **61**, (1997), pp. 317–322.
9. J. CHEN, I. KANJ, AND W. JIA, Vertex cover: further observations and further improvements, *J. Algorithms* **41**, (2001), pp. 280–301.
10. H. DJIDJEV AND S. VENKATESAN, Planarization of graphs embedded on surfaces, *LNCS* **1017** (WG’95), (1995), pp. 62–72.
11. R. DOWNEY AND M. FELLOWS, *Parameterized Complexity*, Springer-Verlag, New York, 1999.
12. J. ELLIS, H. FAN, AND M. FELLOWS, The dominating set problem is fixed parameter tractable for graphs of bounded genus, *LNCS* **2368**, (2002), pp. 180–189.
13. M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.
14. J. GROSS AND T. TUCKER, *Topological Graph Theory*, Wiley-Interscience, New York, 1987.
15. R. IMPAGLIAZZO, R. Paturi, AND F. ZANE, Which problems have strongly exponential complexity? *JCSS* **63**, (2001), pp. 512–530.
16. R. LIPTON AND R. TARJAN, Applications of a planar separator theorem, *SIAM Journal on Computing* **9**, (1980), pp. 615–627.
17. C. PAPADIMITRIOU AND M. YANNAKAKIS, Optimization, approximation and complexity classes, *JCSS* **43**, (1991), pp. 425–440.
18. C. THOMASSEN, The graph genus problem is NP-complete, *J. Algorithms* **10**, (1989), pp. 568–576.

The Definition of a Temporal Clock Operator

Cindy Eisner¹, Dana Fisman^{1,2*}, John Havlicek³, Anthony McIsaac⁴, and
David Van Campenhout^{5**}

¹ IBM Haifa Research Laboratory

² Weizmann Institute of Science

³ Motorola, Inc.

⁴ STMicroelectronics, Ltd.

⁵ Verisity Design, Inc.

Abstract. Modern hardware designs are typically based on multiple clocks. While a singly-clocked hardware design is easily described in standard temporal logics, describing a multiply-clocked design is cumbersome. Thus it is desirable to have an easier way to formulate properties related to clocks in a temporal logic. We present a relatively simple solution built on top of the traditional LTL-based semantics, study the properties of the resulting logic, and compare it with previous solutions.

1 Introduction

Synchronous hardware designs are based on a notion of discrete time, in which the flip-flop (or latch) takes the system from the current state to the next state. The signal that causes the flip-flop (or latch) to transition is termed the *clock*. In a singly-clocked hardware design, the behavior of hardware in terms of the clock naturally maps to the notion of the next-time operator in temporal logics such as LTL[10] and CTL[2], so that the following LTL formula:

$$G(p \rightarrow X q) \tag{1}$$

can be interpreted as “globally, if p then *at the next clock cycle*, q ”. Mapping between a state of a model for the temporal logic and a clock cycle of hardware can then be dealt with by the tool which builds a model from the source code (written in some hardware description language, or HDL).

Modern hardware designs, however, are typically based on multiple clocks. In such a design, for instance, some flip-flops may be clocked with *clka*, while others are clocked with *clkb*. In this case, the mapping between states and clock cycles cannot be done automatically; rather, the formula itself must contain some

* The work of this author was supported in part by the John Von Neumann Minerva Center for the Verification of Reactive Systems.

** *E-mail addresses:* eisner@il.ibm.com (C. Eisner), dana@wisdom.weizmann.ac.il (D. Fisman), john.havlicek@motorola.com (J. Havlicek), anthony.mcisaac@st.com (A. McIsaac), dvc@verisity.com (D. Van Campenhout)

indication of which clock to use. For instance, a clocked version of Formula 1 might be:

$$(G(p \rightarrow X q))@clka \quad (2)$$

We would like to interpret Formula 2 as “globally, if p during a cycle of $clka$, then at the next cycle of $clka$, q ”. In LTL we can express this as:

$$G ((clka \wedge p) \rightarrow X[\neg clka \ W (clka \wedge q)]) \quad (3)$$

Thus, we would like to give semantics to a new operator $@$ such that Formula 2 is equivalent to Formula 3. The issue of defining what such a solution should be for LTL is the problem we explore in this paper.

We present a relatively simple solution built on top of the traditional LTL-based semantics. Our solution is based on the idea that the only role of the clock operator should be to define a projection of the path onto those states where the clock “ticks”¹. Thus, $\neg(f@clk)$ should be equivalent to $(\neg f)@clk$, that is, the clock operator should be its own dual. Achieving this introduces a problem for paths on which the clock never ticks. We solve this problem by introducing a propositional strength operator that extends the semantics from non-empty paths to empty paths in the same way that the strong next operator [8] extends the semantics from infinite to finite paths. We present the resulting logic $LTL^@$, and show that we meet the goal of the “projection view”, as well as other design goals presented below. To show that the clock and propositional strength operators add no expressive power to LTL, we provide a set of rewrite rules to translate an $LTL^@$ formula to an equivalent LTL formula.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 defines hardware clocks. Section 4 discusses design requirements for the clock operator. Section 5 presents the definition of $LTL^@$. In Section 6 we show that we have met the goals of Section 4. Section 7 discusses some additional properties of our logic. Section 8 concludes.

2 Related Work

Many modeling languages, such as Lustre [5] and Signal, incorporate the idea of a clock. However, in this paper we are interested in the addition of a clock operator to temporal logic. The work described in this paper is the result of discussions in the LRM sub-committee of the Accellera Formal Verification Technical Committee (FVTC). All four languages (Sugar2.0, ForSpec, Temporal e, CBV) examined by the committee enhance temporal logic with clock operators. Many of these languages distinguish between *strong* and *weak* clock operators, in a similar way as LTL distinguishes between strong and weak until.

¹ Actually, referring to a projection of the path is not precisely correct, as we allow access to states in between consecutive states of a projection in the event of a clock switch. However, the word “projection” conveys the intuitive function of the clock operator in the case that the formula is singly-clocked. Use of the word “projection” when describing the clocks of Sugar2.0 and ForSpec below is similarly imprecise.

Sugar2.0 supports both strong and weak versions of a clock operator. As originally proposed [3], a strongly clocked Sugar2.0 formula requires the clock to “tick long enough to ensure that the formula holds”, while a weakly clocked formula allows it to stop ticking before then.

In ForSpec [1], which also supports strong and weak clocks, a strongly clocked formula requires only that the clock tick at least once, after which the only role of the clock is to define the projection of the path onto those states where the clock ticks. A weakly clocked formula, on the other hand, holds if the clock never ticks; if it does tick, then the role of the clock is the same as for a strongly clocked formula.

In Temporal e [9], which also supports multiple clocks, clocks are not attributed with strength. This is consistent with the use of Temporal e in simulation, in which behaviors are always finite in duration. Support for reasoning about infinite length behaviors is limited in Temporal e.

In CBV [6], clocking and alignment of formulas are supported by separate and independent sampling and alignment operators. The sampling operator is self-dual and determines the projection in the singly-clocked case. It is similar to the clock operator of LTL[®]. The CBV alignment operators come in a strong/weak dual pair that take us to the first clock event, without affecting the sampling clock. The composition of the sampling operator with a strong/weak alignment operator on the same clock is provided by the CBV synchronization operators, which behave like the ForSpec strong/weak clock operators.

Clocked Temporal Logic [7], confusingly termed CTL by its authors, is another temporal logic that deals with multiple clocks. However, in their solution a clock is a pre-determined subset of the states on a path, and their approach is to associate a clock with each atomic proposition, rather than to clock formulas and sub-formulas.

Wang, Mok and Emerson have defined APTL [11], which enhances temporal logic with multiple real-time clocks. In this work, we are concerned with hardware clocks, which determine the granularity of time in a synchronous system, rather than with clocks in the sense of [11] that measure real time in an asynchronous system. Thus, for example [11] assumes the clock ticks infinitely often, while we are trying to face the problems arising when such an assumption is not adopted.

3 Hardware Clocks

A hardware clock is any signal connected to the *clock input* of a flip-flop or latch. A flip-flop or latch is a memory element, which passes on some function of its inputs to its outputs, but only when its clock input is active. At all other times, it remembers its previous input. A flip-flop responds only to a change in its clock input, while a latch will function continuously as long as the clock input is active.

There are many types of flip-flops and latches, each of which passes on different functions of its inputs to its outputs. Furthermore, real flip-flops and latches work in the real world, where time is continuous, and the amount of time during

which a signal is asserted makes a difference. For the purposes of this paper, it is sufficient to examine one kind of flip-flop, working in an abstract world where time is discrete, defined as follows.

Definition 1 (Abstract flip-flop). *An abstract flip-flop is a hardware device with two inputs, d and c , and one output, o . Its functionality is described by the formula $o' = (c \wedge d) \vee (\neg c \wedge o)$, where o' is the value of o at the next point in time.²*

4 Issues in Defining the Clock Operator

We begin by trying to set the design requirements for the clock operator. What is the intuition it should capture? What are the problems involved?

The projection view. When only a single clock is involved we would like that a clocked formula $f@clk$ hold on a path π if and only if the unclocked formula f holds on a path π' where π' is π projected onto those states where clk holds.

Non-accumulation of clocks. In many hardware designs, large chunks of the design work on some main clock, while small pieces work on a secondary clock. Rather than require the user to specify a clock for each sub-formula, we would like to allow clocking of an entire formula on a main clock, and pieces of it on a secondary clock, in such a way that the outer clock (which is applied to the entire formula) does not affect the inner clock (which is applied to one or more sub-formulas). That is, we want a nested clock operator to have the effect of “changing the projection”, rather than further projecting the projected path.

Finite and empty paths. The introduction of clocks requires us to deal with finite paths, since the projection of an infinite path may be finite. For LTL, this means that the single *next operator* X no longer suffices. To see why, consider an atomic proposition p and a path where the clock stops ticking. On the last state of the path, do we want $(X p)@clk$ to hold or not? Whatever we do, assuming we want to preserve the duality $\neg(X p) = X(\neg p)$ under clocks, and thus obtain a definition under which $\neg((X p)@clk)$ is equivalent to $(X(\neg p))@clk$, the result is unsatisfactory. For instance, if $(X p)@clk$ holds when the clock stops ticking, then $\neg((X p)@clk)$ does not. Letting $p = \neg q$, we get that $(X q)@clk$ does not hold if the clock stops ticking, which is a contradiction.

Thus, the addition of clocks to LTL-based semantics introduces problems similar to those of defining LTL semantics for finite paths. In particular, it requires us to make a decision as to the semantics of the next operator on the last clock tick of a path, with the result that the next operator is not dual to itself. Instead, we end up with two next operators, strong and weak, which are dual to each other [8].

² The value of the flip-flop's output is not defined at the first point in time.

Not only may the projection of an infinite path be finite, it may be empty as well. For LTL, this means that the duality problem exists not only for the next operator, but also for atomic propositions. Whatever choice we make for the semantics of $p@clk$ (where p is an atomic proposition) on an empty path, we cannot achieve the duality $\neg(p@clk) = (\neg p)@clk$ without adding something to the logic.

A natural solution for the semantics of a formula over a path where the clock does not tick is to take the strength from the temporal operator. Under this approach, for example, a clocked strong next does not hold on a path with no ticks, while a clocked weak next does hold on such a path. This solution breaks down in the case of a formula with no temporal operators. One way to deal with this is to make a decision as to the semantics of the clock operator on a path with no ticks, giving two clock operators which are dual to each other, rather than a single clock operator that is dual to itself. Below we discuss this issue in more detail.

Avoiding the problems of existing distinctions between strong and weak clocks. Three of the languages considered by the FVTC make a distinction between strong and weak clocks. However, each has significant drawbacks that we would like to avoid.

In Sugar2.0 as originally proposed [3], a strongly clocked formula requires the clock to “tick long enough to ensure that the formula holds”, while a weakly clocked formula allows it to stop ticking before then. Thus, for instance, the formula $(F p)@clk!$ (where $@$ is the clock operator, clk is the clock, and the $!$ indicates that it is strong) requires there to be enough ticks of clk so that p eventually holds, whereas the formula $(F p)@clk$ (which is a weakly clocked formula, because there is no $!$) allows the case where p never occurs, if it “is the fault of the clock”, i.e., if the clock ticks a finite number of times. Negation switches the clock strength, so that $\neg(f@clk) = (\neg f)@clk!$ and we get that $(G q)@clk!$ holds if the clock ticks an infinite number of times and q holds at every tick, while $(G q)@clk$ holds if q holds at every tick, no matter how many there are. Although initially pleasing, this semantics has the disadvantage that the formula $(F p) \wedge (G q)$ cannot be satisfactorily clocked for a finite path, because $((F p) \wedge (G q))@clk!$ does not hold on any finite path, while $((F p) \wedge (G q))@clk$ makes no requirement on p on such a path. Since our intent is to define a semantics that can be used in simulation (where every path is finite) as well as in model checking, this is unacceptable.

In ForSpec, a strongly clocked formula requires only that the clock tick at least once, after which the only role of the clock is to define the projection of the path onto those states where the clock ticks. A weakly clocked formula, on the other hand, holds if the clock never ticks; if it does tick, then the role of the clock is the same as for a strongly clocked formula. Thus, the only difference between strong and weak clocks in ForSpec is on paths whose projection is empty. This leads to the strange situation that a liveness formula may hold on some path π , but not on an extension of that path, $\pi\pi'$. For instance, if p is an atomic

proposition, then $(F p)@clk$ holds if there are no ticks of clk , but does not hold if there is just one tick, at which p does not hold.

In CBV, there is a self-dual clock operator, the sampling operator, according to which all temporal advances are aligned to the clock. However, the sampling operator causes no initial alignment. Therefore, sampled booleans are evaluated immediately; sampled next-times align to the next strictly future tick of the clock; and so forth. As a result, the projection defined by the CBV sampling operator includes the first state of a path, regardless of whether it is a tick of the clock. The CBV alignment and synchronization operators come in strong/weak dual pairs. The latter behave like the ForSpec strong/weak clock operators and therefore suffer from the same disadvantages.

Under the solutions described above, the clock or synchronization operator is given the role of determining the semantics in case the path is empty. As a result, the operator cannot be its own dual, resulting in two kinds of clocks. Our goal is to define a logic where the only role of the clock operator is to determine a projection. Thus, we seek a solution which solves the problem of an empty path in such a way that the clock operator is its own dual, eliminating the need for two kinds of clocks.

Equivalence and substitution. We would like the logic to adhere to an equivalence lemma as well as a substitution lemma. Loosely speaking, an equivalence lemma requires that two equivalent LTL formulas remain equivalent after the application of the clock operator. The substitution lemma guarantees that substituting sub-formula g for an equivalent sub-formula h does not change the truth value of the original formula.

Motivating example. We would like our original motivating example from the introduction to hold.

Goals

To summarize, our goals composed in light of the discussion above, are as follows:

1. When singly-clocked, the semantics should be that of the projection view.
2. Clocks should not accumulate.
3. The clock operator should be its own dual.
4. There should be a clocked version of $(F p) \wedge (G q)$ that is meaningful on paths with a finite number of clock ticks.
5. For any atomic proposition p , if $(F p)@clk$ holds on a path, it should hold on any extension of that path.
6. For any clock c , two equivalent LTL formulas should remain equivalent when clocked with c .
7. Substituting sub-formula g for an equivalent sub-formula h should not change the truth value of the original formula.
8. The truth value of LTL° Formula 2 should be the same as the truth value of LTL Formula 3 for every path.

5 The Definition of LTL°

We solve the problem of finite paths introduced by clocks in LTL -based semantics by supplying both strong and weak versions of the next operator ($X!$ and X).

We solve the problem of empty paths by introducing a new, propositional strength operator. Thus, if p is an atomic proposition, then $p!$ is as well. While p is a weak atomic proposition, and so holds on an empty path, $p!$ is a strong atomic proposition, and does not hold on such a path. The intuition behind this is that the role of the strength of a temporal operator is to tell us how far a finite path is required to extend. For strong until, as in $[f \text{ U } g]$, we require that g hold somewhere on the path. For strong next, as in $X! f$, we require that there be a next state. Intuitively then, we get that a strong proposition, as in $p!$, requires that there be a current state.

Without clocks, there is never such a thing as not having a current state, so the problem of an empty path does not come up in traditional temporal logics. But for a clocked semantics, there may indeed not be a first state. In such a situation, putting the responsibility on the atomic proposition gives a natural extension to the idea of the formula itself telling us how far a finite path must extend. This leaves us with the desired situation that the sole responsibility of the clock operator will be to “light up” the states that are relevant for the current clock context, which is the intuitive notion of a clock.

5.1 Syntax

The syntax of LTL° is defined below, where we use the term *boolean expression* to refer to any application of the standard boolean operators to atomic propositions.

Definition 2 (Formulas of LTL°).

- If p is an atomic proposition, then p and $p!$ are LTL° formulas.
- If clk is a boolean expression and f , f_1 , and f_2 are LTL° formulas, then the following are LTL° formulas: $\neg f$, $f_1 \wedge f_2$, $X! f$, $[f_1 \text{ U } f_2]$, $f @ clk$.

Additional operators are derived from the basic operators defined above:³

- $f_1 \vee f_2 \stackrel{\text{def}}{=} \neg(\neg f_1 \wedge \neg f_2)$
- $f_1 \rightarrow f_2 \stackrel{\text{def}}{=} \neg f_1 \vee f_2$
- $F f \stackrel{\text{def}}{=} [T \text{ U } f]$
- $X f \stackrel{\text{def}}{=} \neg X! \neg f$
- $G f \stackrel{\text{def}}{=} \neg F \neg f$
- $[f_1 \text{ W } f_2] \stackrel{\text{def}}{=} [f_1 \text{ U } f_2] \vee G f_1$

LTL is the subset of LTL° consisting of the formulas that have no clock operator and no sub-formulas of the form $p!$, for some atomic proposition p .

³ Where T is an atomic proposition that holds on every state. In the sequel, we also use F , which is an atomic proposition that does not hold for any state.

5.2 Semantics

We define the semantics of LTL° formulas over words⁴ from the alphabet 2^P . A letter is a subset of the set of atomic propositions P such that T belongs to the subset and F does not. We will denote a letter from 2^P by ℓ and an empty, finite, or infinite word from 2^P by w . We denote the length of word w as $|w|$. An empty word $w = \epsilon$ has length 0, a finite word $w = (\ell_0 \ell_1 \ell_2 \dots \ell_n)$ has length $n + 1$, and an infinite word has length ∞ . We denote the i^{th} letter of w by w^i . We denote by $w^{i..}$ the suffix of w starting at w^i . That is, $w^{i..} = (w^i w^{i+1} \dots w^n)$ or $w^{i..} = (w^i w^{i+1} \dots)$. We denote by $w^{i..j}$ the finite sequence of letters starting from w^i and ending in w^j . That is, $w^{i..j} = (w^i w^{i+1} \dots w^j)$.

We first present the semantics of LTL° minus the clock operator over infinite, finite, and empty words (*unlocked semantics*). We then present the semantics of LTL° over infinite, finite, and empty words (*clocked semantics*). Later, we relate the two.

Unlocked semantics. We now present a semantics for LTL° minus the clock operator. The semantics is defined with respect to an infinite, finite, or empty word. The notation $w \models f$ means that formula f holds along the word w . The semantics is defined as follows, where p denotes an atomic proposition, f , f_1 , and f_2 denote formulas, and j and k denote natural numbers (i.e., non-negative integers).

- $w \models p \iff |w| = 0$ or $p \in w^0$
- $w \models p! \iff |w| > 0$ and $p \in w^0$
- $w \models \neg f \iff w \not\models f$
- $w \models f_1 \wedge f_2 \iff w \models f_1$ and $w \models f_2$
- $w \models X! f \iff |w| > 1$ and $w^{1..} \models f$
- $w \models [f_1 \cup f_2] \iff$ there exists $k < |w|$ such that $w^{k..} \models f_2$, and for every $j < k$ $w^{j..} \models f_1$

Clocked semantics. We define the semantics of an LTL° formula with respect to an infinite, finite, or empty word w and a context c , where c is a boolean expression over P . For word w and boolean expression b , we say that $w^i \models b$ iff $w^{i..i} \models b$. Second, we say that a finite word w is a *clock tick* of clock c if c holds at the last letter of w and does not hold at any previous letter of w . Formally,

Definition 3 (is a clock tick of). We say that finite word w is a clock tick of c iff $|w| > 0$ and $w^{|w|-1} \models c$ and for every natural number $i < |w| - 1$, $w^i \not\models c$.

The notation $w \models^c f$ means that formula f holds along the word w in the context of clock c . The semantics of an LTL° formula is defined as follows, where p denotes an atomic proposition, c , and c_1 denote boolean expressions, f , f_1 , and f_2 denote LTL° formulas, and j and k denote natural numbers.

⁴ Relating the semantics over words to semantics over models is done in the standard way. Due to lack of space, we omit the details.

- $w \models^c p \iff$ for all $j < |w|$ such that $w^{0..j}$ is a clock tick of c , $p \in w^j$
- $w \models^c p! \iff$ there exists $j < |w|$ such that $w^{0..j}$ is a clock tick of c and $p \in w^j$
- $w \models^c \neg f \iff w \not\models^c f$
- $w \models^c f_1 \wedge f_2 \iff w \models^c f_1$ and $w \models^c f_2$
- $w \models^c X! f \iff$ there exist $j < k < |w|$ such that $w^{0..j}$ is a clock tick of c and $w^{j+1..k}$ is a clock tick of c and $w^{k..} \models^c f$
- $w \models^c [f_1 \cup f_2] \iff$ there exists $k < |w|$ such that $w^k \models c$ and $w^{k..} \models^c f_2$ and for every $j < k$ such that $w^j \models c$, $w^{j..} \models^c f_1$
- $w \models^c f@c_1 \iff w \models^{c_1} f$

In LTL° , every formula is evaluated in the context of a clock. The projection view requires that propositions are evaluated not at the first state of a path, but at the first state where the context clock ticks (if there is such a state). To be consistent with this, if the clock does not tick in the first state of a path, a formula Xf or $X!f$ must be evaluated in terms of the value of f at the second tick of the clock after the initial state.

6 Meeting the Goals

In this section, we analyze the logic LTL° with respect to the goals of Section 4. Due to lack of space all proofs are omitted; they can be found in the full version of the paper. The following definitions are needed for the sequel.

Definition 4 (Projection). *The projection of word w onto clock c , denoted $w|_c$, is the word obtained from w after leaving only the letters which satisfy c .*

Definition 5 (Unclocked equivalent). *Two LTL° formulas f and g with no clock operator are unclocked equivalent ($f \equiv g$) if for all words w , $w \models f$ if and only if $w \models g$.*

Definition 6 (Clocked equivalent). *Two LTL° formulas f and g are clocked equivalent ($f \overset{\circ}{\equiv} g$) if for all words w and all contexts c , $w \models^c f$ if and only if $w \models^c g$.*

Goal 1. The following theorem states that when a single clock is applied to a formula, the projection view is obtained.

Theorem 1 *Let f be an LTL° formula with no clock operator, c a boolean expression and w an infinite, finite, or empty word.*

$$w \models^c f \quad \text{if and only if} \quad w|_c \models f$$

It follows immediately that the clocked semantics is a generalization of the unlocked semantics - that is, that the clocked semantics reduces to the unlocked semantics when the context is $\top$.

Corollary 1 *Let f be an LTL° formula with no clock operator, and w a word.*

$$w \models^\top f \quad \text{if and only if} \quad w \models f$$

Goal 2. Looking at the semantics for $f@c_1$ in context c it is easy to see that $f@c_1@c_2 \equiv^\circ f@c_1$, and therefore clocks do not accumulate.

Goal 3. The following claim states that this goal is met.

$$\text{Claim. } (\neg f)@b \equiv^\circ \neg(f@b)$$

Goal 4. The clocked version of $(F p) \wedge (G q)$ is $((F p) \wedge (G q))@c$, and holds if p holds for some state and q holds for all states on the projected path.

Goal 5. The following claim states that Goal 5 is met.

Claim. Let b , clk and c be boolean expressions, w a finite word, and w' an infinite or finite word.

$$w \models^c (F b)@clk \implies ww' \models^c (F b)@clk$$

Goal 6. The following claim states that Goal 6 is met.

Claim. Let f and g be LTL° formulas with no clock operators, and let b be a boolean expression.

$$f \equiv g \implies f@b \equiv^\circ g@b$$

Note that if f and g are unlocked formulas then for some boolean expression c it may be that $f@c \equiv^\circ g@c$, even though $f \not\equiv g$. For example, let $f = (\neg c) \rightarrow \top$ and let $g = (\neg c) \rightarrow \text{F}$. Then $f@c \equiv^\circ g@c$, but $f \not\equiv g$.

Goal 7. We use the notation $\varphi[\psi \leftarrow \psi']$ to denote the formula obtained from φ by replacing sub-formula ψ with ψ' . The following claim states that this goal is met.

Claim. Let g be a sub-formula of f , and let $g' \equiv^\circ g$. Then $f \equiv^\circ f[g \leftarrow g']$.

Goal 8. The following claim states that this goal is met.

Claim. For every word w ,

$$w \models^\top (G(p \rightarrow X q))@clk \iff w \models G((clk \wedge p) \rightarrow X[\neg clk \text{ W } (clk \wedge q)])$$

7 Discussion

Looking backwards. In LTL, the evaluation of formula $G(p \rightarrow f)$, where p is a boolean expression, depends only on the evaluations of f starting at those points where p holds. In particular, satisfaction of $G(p \rightarrow f)$ on w is independent of the initial segment of w before the first occurrence of p . We might hope that satisfaction of $G(p@clkp \rightarrow f@clkf)$ on w will be independent of the initial segment of w before the first occurrence of p at a tick of $clkp$. This is not the case. For instance, consider the following formula:

$$G(p@clkp \rightarrow q@clkq) \quad (4)$$

which is a clocked version of the simple invariant $G(p \rightarrow q)$, where both p and q are boolean expressions. Formula 4 can be rewritten as

$$G([\neg clkp \text{ W } (clkp \wedge p)] \rightarrow [\neg clkq \text{ W } (clkq \wedge q)]) \quad (5)$$

by the rewrite rules in Theorem 2 below. The result is a dimension of temporality not present in the original, unlocked formula. For instance, for the behavior of p shown in Figure 1, Formula 5 requires that q hold at time 4 (because $[\neg clkp \text{ W } (clkp \wedge p)]$ holds at time 3, and in order for $[\neg clkq \text{ W } (clkq \wedge q)]$ to hold

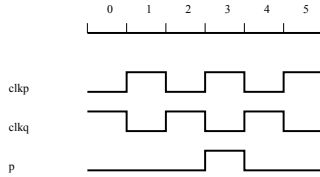


Fig. 1. Behavior of p illustrating a problem with Formula 5

at time 3, we need q to hold at time 4). Not only does Formula 5 require that q hold at time 4 for the behavior of p shown in Figure 1, it also requires that q hold at time 2 (because $[\neg clkp \text{ W } (clkp \wedge p)]$ holds at time 2, and in order for $[\neg clkq \text{ W } (clkq \wedge q)]$ to hold at time 2, we need q to hold at time 2). Thus, the direction of the additional dimension of temporality may be backwards as well as forwards.

To avoid the “looking backward” phenomenon the semantics of a boolean expression under clock operators should be non-temporal. For instance, we could define $p@clk = p$ and $p!@clk = p!$, or alternatively, $p@clk = clk \rightarrow p$ and $p!@clk = clk \wedge p!$. The disadvantage of these definitions is that the projection view is not preserved (because on a path such that p holds at the first clock but does not hold at the first state, $p@clk$ and/or $p!@clk$ do/does not hold).

We note that Formula 4 has the same backwards-looking feature in other semantics with strong and weak clocks [3,1], so the phenomenon does not arise purely from the design decisions we have taken here. Furthermore, if the multi-clocked version is taken as $(G(p \rightarrow (q@clkq)))@clkp$, then the phenomenon does

not arise. Many properties of practical interest are of this form, for example a property asserting that the data is not corrupted between input and output interfaces clocked on different clocks:

$$(G((receive \wedge (data\_in = d)) \rightarrow (\neg send \cup (send \wedge (data\_out = d))))@clk\_out))@clk\_in \quad (6)$$

$[f \cup g]$ as a fixed point. In standard LTL, $[f \cup g]$ can be defined as a least solution of the equation $S = g \vee (f \wedge X! S)$. In LTL° , there is a fixed point characterization if f and g are themselves unclocked, because $[f \cup g] \equiv (T! \wedge g) \vee (f \wedge X![f \cup g])$ (the conjunction with $T!$ is required in order to ensure equivalence on empty paths as well as the non-empty paths on which standard LTL formulas are interpreted). Thus by the claim of Goal 6 $[f \cup g]@c \equiv ((T! \wedge g) \vee (f \wedge X![f \cup g]))@c$ for any clock c and any formulas f and g containing no clock operators, and hence by the semantics, the truth value of $[f \cup g]$ under context c is the same as the truth value of $(T! \wedge g) \vee (f \wedge X![f \cup g])$ under context c , for any context c . If f and g contain clock operators, this equivalence no longer holds. Let p, q and d be atomic propositions, and let $f = q@d$. Consider a word w such that $w^0 \models d \wedge q$ and for all $i > 0$, $w^i \not\models d \wedge q$, and $w^0 \not\models c$. Then $w \models^c f$ hence $w \models^c (T! \wedge f) \vee (p \wedge X![p \cup f])$. However, since $w^0 \not\models c$, and there is no state other than w^0 where $d \wedge q$ holds, $w \not\models^c [p \cup f]$. Note that while of theoretical interest, the lack of a fixed point characterization of $[f \cup g]$ is not an obstacle to model checking, since any LTL° formula can be translated to an equivalent LTL formula by the rewrite rules presented below.

Xf and $X!f$ on states where the clock does not hold. As mentioned earlier, another property of our definition is that on states where the clock does not hold, the next operators take us two clock cycles into the future, instead of the one clock cycle that we might expect. Further consideration shows that this is a direct result of the projection view: since $p@clk$ must mean that p holds at the next clock, it is clear that an application of a next operator (as in $(Xp)@clk$ or $(X!p)@clk$) must mean that p holds at the one after that. This behavior of a clocked next operator is a consideration only in multi-clocked formulas, since in a singly-clocked formula, we are never “at” a state where the clock does not hold (except perhaps at the initial state).

Expressive power. The clock operator provides a concise way to express what would otherwise be cumbersome, but it does not add expressive power. Theorem 2 below states that the truth value of any LTL° formula under context clk is the same as that of the LTL formula $f' = \mathcal{T}^{clk}(f)$, where $\mathcal{T}^{clk}(f)$ is defined as follows:

- $\mathcal{T}^{clk}(p) = [\neg clk \ W \ (clk \wedge p)]$
- $\mathcal{T}^{clk}(p!) = [\neg clk \ \cup \ (clk \wedge p)]$
- $\mathcal{T}^{clk}(\neg f) = \neg \mathcal{T}^{clk}(f)$
- $\mathcal{T}^{clk}(f_1 \wedge f_2) = \mathcal{T}^{clk}(f_1) \wedge \mathcal{T}^{clk}(f_2)$

- $\mathcal{T}^{clk}(X! f) = [\neg clk \cup (clk \wedge X![\neg clk \cup (clk \wedge \mathcal{T}^{clk}(f))])]$
- $\mathcal{T}^{clk}([f_1 \cup f_2]) = [(clk \rightarrow \mathcal{T}^{clk}(f_1)) \cup (clk \wedge \mathcal{T}^{clk}(f_2))]$
- $\mathcal{T}^{clk}(f @ clk_1) = \mathcal{T}^{clk_1}(f)$

Theorem 2 *Let f be any LTL[@] formula, c a boolean expression, and w a word.*

$$w \models^c f \quad \text{if and only if} \quad w \models \mathcal{T}^c(f)$$

Clearly $\mathcal{T}^{clk}()$ defines a recursive procedure whose application starting with $clk = \top$ results in an LTL formula with the same truth value in context $\top$. Note that while we can rewrite a formula f into an LTL formula f' with the same truth value, we cannot use formulas f and f' interchangeably. For example, $p!@clk1$ translates to $[\neg clk1 \cup (clk1 \wedge p)]$, but these two are not clocked equivalent (because clocking each of them with $clk2$ will give different results).

8 Conclusion and Future Work

We have given a relatively simple definition of multiple clocking for LTL augmented with a clock operator that we believe captures the intuition behind hardware clocks, and have presented a set of rewrite rules that can be used as an implementation of the clock operator. In our definition, the only role of the clock operator is to define a projection of the path, and it is its own dual.

Our semantics, based on strong and weak propositions, achieves goals not achieved by semantics based on strong and weak clocks. In particular, it gives the projection view for singly-clocked formulas and a uniform treatment of empty and non-empty paths, including the interpretation of the operators G and F . It does not provide an easy solution to the question of how to define U as a fixed point operator for multi-clocked formulas. Future work should seek a way to resolve these issues without losing the advantages.

It may be noted that in the strong/weak clock semantics, alignment is always applied immediately after the setting of a clock context; while in the strong/weak proposition semantics, it is always applied immediately before an atomic proposition. Allowing more flexibility in where alignment (and strength) is applied may be a useful avenue for investigation.

Acknowledgements. We would like to thank Sharon Barner, Shoham Ben-David, Alan Hartman and Emmanuel Zarpas for their help with the formal definition of multiple clocks. We would also like to thank Mike Gordon, whose work on studying the formal semantics of Sugar2.0 with HOL [4] greatly contributed to our understanding of the problems discussed in this paper. Finally, thank you to Shoham Ben-David, Avigail Orni and Sitvanit Ruah for careful review and important comments.

References

1. R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification language. In J.-P. Katoen and P. Stevens, editors, *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2280 of *Lecture Notes in Computer Science*. Springer, 2002.
2. E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logics of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.
3. C. Eisner and D. Fisman. Sugar 2.0 proposal presented to the Accellera Formal Verification Technical Committee, March 2002. At http://www.haifa.il.ibm.com/projects/verification/sugar/Sugar_2.0-Accellera.ps.
4. M. J. C. Gordon. Using HOL to study Sugar 2.0 semantics. In *Proc. 15th International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, NASA Conference Proceedings CP-2002-211736, 2002.
5. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991.
6. J. Havlicek, N. Levi, H. Miller, and K. Shultz. Extended CBV statement semantics, partial proposal presented to the Accellera Formal Verification Technical Committee, April 2002. At http://www.eda.org/vfv/hm/att-0772/01-ecbv_statement_semantics.ps.gz.
7. C. Liu and M. Orgun. Executing specifications of distributed computations with Chronologic(MC). In *Proceedings of the 1996 ACM Symposium on Applied Computing (SAC), February 17-19, 1996, Philadelphia, PA, USA*. ACM, 1996.
8. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*, pages 272–273. Springer-Verlag, New York, 1995.
9. M. Morley. Semantics of temporal e. In T. F. Melham and F. G. Moller, editors, *Proc. Banff'99 Higher Order Workshop (Formal Methods in Computation)*, 1999. University of Glasgow, Dept. of Computing Science Technical Report.
10. A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.
11. F. Wang, A. K. Mok, and E. A. Emerson. Distributed real-time system specification and verification in APTL. *ACM Transactions on Software Engineering and Methodology*, 2(4):346–378, Oct. 1993.

Minimal Classical Logic and Control Operators

Zena M. Ariola¹ and Hugo Herbelin²

¹ University of Oregon, Eugene, OR 97403, USA
ariola@cs.uoregon.edu

² INRIA-Futurs, Parc Club Orsay Université, 91893 Orsay Cedex, France
Hugo.Herbelin@inria.fr

Abstract. We give an analysis of various classical axioms and characterize a notion of minimal classical logic that enforces Peirce’s law without enforcing Ex Falso Quodlibet. We show that a “natural” implementation of this logic is Parigot’s classical natural deduction. We then move on to the computational side and emphasize that Parigot’s $\lambda\mu$ corresponds to minimal classical logic. A continuation constant must be added to $\lambda\mu$ to get full classical logic. We then map the extended $\lambda\mu$ to a new theory of control, $\lambda\mathcal{C}^-top$, which extends Felleisen’s reduction theory. $\lambda\mathcal{C}^-top$ allows one to distinguish between aborting and throwing to a continuation. It is also in correspondence with the proofs of a refinement of Prawitz’s natural deduction.

1 Introduction

Traditionally, classical logic is defined by extending intuitionistic logic with either Pierce’s law, excluded middle or the double negation law. We show that these laws are not equivalent and define *minimal classical logic*, which validates Peirce’s law but not Ex Falso Quodlibet (EFQ), *i.e.* the law $\perp \rightarrow A$. The notion is interesting from a computational point of view since it corresponds to a calculus with a notion of control (such as `callcc`) which however does not allow one to abort a computation. We point out that closed typed terms of Parigot’s $\lambda\mu$ [Par92] correspond to tautologies of minimal classical logic and not of (full) classical logic. We define a new calculus called $\lambda\mu\text{-top}$. Tautologies of classical natural deduction correspond to closed typed $\lambda\mu\text{-top}$ terms. We show the correspondence of $\lambda\mu\text{-top}$ with a new theory of control, $\lambda\mathcal{C}^-top$. The calculus $\lambda\mathcal{C}^-top$ is interesting in its own right, since it extends Felleisen’s theory of control ($\lambda\mathcal{C}$) [FH92]. The study of $\lambda\mathcal{C}^-top$ leads to the development of a refinement of Prawitz’s natural deduction [Pra65] in which one can distinguish between aborting a computation and throwing to a continuation (aborting corresponds to throwing to the top-level continuation). This logic provides a solution to the mismatch between the operational and proof-theoretical interpretation of Felleisen’s $\lambda\mathcal{C}$ reduction theory.

We devote Section 2 to the definition of the various logics considered in this paper. Sections 3 through 5 explain their computational counterparts. We discuss related work in Section 6 and conclude in Section 7.

$\frac{}{\Gamma, A \vdash_M A} Ax$	$\frac{\Gamma, A \vdash_M B}{\Gamma \vdash_M A \rightarrow B} \rightarrow_i$	$\frac{\Gamma \vdash_M A \rightarrow B \quad \Gamma \vdash_M A}{\Gamma \vdash_M B} \rightarrow_e$

Fig. 1. Minimal Natural Deduction

2 Minimal, Intuitionistic, and Classical Logic

In this paper, we restrict our attention to propositional logic. We assume a set of *formulas*, denoted by roman uppercase letters A, B , *etc.*, which are built from an infinite set of *propositional atoms* (ranged over by X, Y , *etc.*), a distinguished formula $\perp$ denoting *false*, and *implication* written $\rightarrow$. We define *negation* as $\neg A \equiv A \rightarrow \perp$. A *named formula* is a pair of a formula and a name taken from an infinite set of *names*. We write A^x, B^α , *etc.* for named formulas. A *context* is a set of named formulas¹. We use Greek uppercase letters Γ, Δ , *etc.* for contexts. We generally omit the names, unless there is an ambiguity. We will consider *sequents* of the form $\Gamma \vdash A$, $\Gamma \vdash, \Gamma \vdash; \Delta$, and $\Gamma \vdash A; \Delta$. The formulas in Γ are the *hypotheses* and the formulas on the right-hand side of the symbol $\vdash$ are the *conclusions*. In each case, the intuitive meaning is that the conjunction of the hypotheses implies the disjunction of the conclusions. A sequent with no conclusion means the negation of the conjunction of the hypotheses. As initially shown by Gentzen [Gen69] in his sequent calculus LK, classical logic can be obtained by considering sequents with several conclusions. Parigot extended this approach to natural deduction [Par92]. We will see that using sequents with several conclusions allows for a uniform presentation of different logics.

In the rest of the paper, we successively recall the definitions of minimal, intuitionistic and classical logic. We state simple facts about various classical axioms from which the definition of minimal classical logic emerges. Although we use natural deduction to formalize the various logics, we could have used sequent calculi instead (then the Curry-Howard correspondence would be with Herbelin's calculus [Her94]). If S is a schematic axiom or rule, we denote by $S, \Gamma \vdash A$ the fact that $\Gamma \vdash A$ is derivable using an arbitrary number of instances of S .

Minimal Logic. *Minimal natural deduction* implements minimal logic [Joh36]. It is defined by the set of (schematic) inference rules given in Figure 1. In minimal logic, $\perp$ is neutral and has no specific rule associated to it.

Normal proofs are an important tool for reasoning about provability in natural deduction. We say that an occurrence of $\rightarrow_e$ (also called Modus Ponens) is *normal* if its left premise is an axiom or another normal instance of Modus

¹ If interested only in provability, one could have defined contexts just as sets of formulas (not as sets of named formulas). But to assign terms to proofs, one needs to be able to distinguish between different occurrences of the same formula. This is the role of names. Otherwise, *e.g.* the two distinct normal proofs of $A, A \vdash A$ (representable by the λ -terms $\lambda x.\lambda y.x$ and $\lambda x.\lambda y.y$) would have been identified.

$\frac{}{\Gamma, A \vdash_I A} Ax \quad \frac{\Gamma \vdash_I}{\Gamma \vdash_I A} Activate$		
$\frac{\Gamma \vdash_I \perp}{\Gamma \vdash_I} \perp_e$	$\frac{\Gamma, A \vdash_I B}{\Gamma \vdash_I A \rightarrow B} \rightarrow_i$	$\frac{\Gamma \vdash_I A \rightarrow B \quad \Gamma \vdash_I A}{\Gamma \vdash_I B} \rightarrow_e$

Fig. 2. Intuitionistic Natural Deduction

Ponens. We say that a proof in minimal logic is *normal* if any occurrence of Modus Ponens in the proof is normal. As is well-known, a provable statement can be proved with a normal proof.

Theorem 1 (Prawitz). *If $\Gamma \vdash_M A$ is provable then there is a normal proof of $\Gamma \vdash_M A$.*

Intuitionistic logic. *Intuitionistic natural deduction* is described in Figure 2. The rule $\perp_e$ introduces a sequent with no conclusion, thus allowing the application of a weakening rule named *Activate*. Obviously, this presentation of intuitionistic logic is equivalent to minimal logic extended with the schematic axiom $\perp \rightarrow A$.

Proposition 1. $\Gamma \vdash_I A \text{ iff } EFQ, \Gamma \vdash_M A$.

In propositional or first-order predicate logic, there is no formula $\perp$ with the desired property, as stated by the following lemma which expresses that (propositional) intuitionistic logic is strictly stronger than minimal logic.

Proposition 2. $\not\vdash_M EFQ$.

In contrast, in second-order logic, a formula having the property of $\perp$ is $\forall X.X$. However, the rule $\perp_e$ is still not valid for $\forall X.X$.

Classical axioms. We now give an analysis in minimal logic of different axiom schemes² leading to classical logic.

$(\neg A \rightarrow A) \rightarrow A$	Weak Peirce's law ($PL_{\perp}$)
$\neg A \vee A$	Excluded middle (EM)
$((A \rightarrow B) \rightarrow A) \rightarrow A$	Peirce's law (PL)
$(A \rightarrow B) \vee A$	Generalized excluded-middle (GEM)
$\neg\neg A \rightarrow A$	Double negation law (DN)

We classify the axioms in three categories: we call $PL_{\perp}$ and EM *weak classical axioms*, PL and GEM *minimal classical axioms*, and DN a *full classical axiom*. The main results of this section are that none of the classical axioms are indeed derivable in minimal logic and that the weak classical axioms are weaker in

² To reason about excluded-middle, we enrich the set of formulas with disjunction and the usual inference rules.

$\frac{}{\Gamma, A \vdash_{MC} A; \Delta} Ax$	$\frac{\Gamma \vdash_{MC}; A, \Delta}{\Gamma \vdash_{MC} A; \Delta} Activate$	$\frac{\Gamma \vdash_{MC} A; A, \Delta}{\Gamma \vdash_{MC}; A, \Delta} Passivate$
$\frac{\Gamma, A \vdash_{MC} B; \Delta}{\Gamma \vdash_{MC} A \rightarrow B; \Delta} \rightarrow_i$	$\frac{\Gamma \vdash_{MC} A \rightarrow B; \Delta \quad \Gamma \vdash_{MC} A; \Delta}{\Gamma \vdash_{MC} B; \Delta} \rightarrow_e$	

Fig. 3. Minimal Classical Natural Deduction

minimal logic than the minimal classical axioms, which themselves are weaker than DN. Together with EFQ, weak and minimal classical axioms are however equivalent to DN.

Proposition 3. *In minimal logic, we have*

1. *neither $PL_{\perp}$, PL , EM , GEM nor DN is derivable.*
2. *$PL_{\perp}$ and EM are equivalent (as schemes).*
3. *GEM and PL are equivalent (as schemes).*
4. *GEM and PL imply EM and $PL_{\perp}$ but not conversely.*
5. *DN implies GEM and PL but not conversely.*
6. *DN , $EM+EFQ$, $GEM+EFQ$, $PL_{\perp}+EFQ$ and $PL+EFQ$ are all equivalent.*

The previous result suggests that there is space for a classical logic which does validate Peirce's law (or GEM) but not EFQ . Let us call this logic *minimal classical logic*. In contrast, EM and $PL_{\perp}$ without EFQ are weaker than PL , and their addition to minimal logic seems uninteresting. We will investigate a weaker form of EFQ at the end of this section.

Minimal Classical Logic. An axiom-free implementation of minimal classical logic is actually Parigot's classical natural deduction [Par92] (with no special rule for $\perp$). The inference rules are shown in Figure 3. Parigot's convention is to have two kinds of sequents, one with only named formulas on the right, written $\Gamma \vdash; \Delta$, and one with exactly one unnamed formula on the right, written $\Gamma \vdash A; \Delta$. We now state that minimal Parigot classical natural deduction is equivalent to minimal logic extended with Peirce's law, *i.e.* it implements minimal classical logic³.

Proposition 4. $\Gamma \vdash_{MC} A$ *iff* $PL, \Gamma \vdash_M A$

Thanks to Proposition 3(4), we have, as a Corollary,

Corollary 1. *Minimal Parigot's classical natural deduction does not prove DN .*

Note however that $\vdash_{MC} \neg\neg A \rightarrow A; \perp$ is provable.

We now define the notion of *normal proof* for minimal Parigot classical natural deduction. We say that an occurrence of the rule *Passivate* is *normal* if its

³ The proof involves replacing each instance of *Activate* on A by a number of instances of PL which is equal to the number of instances of *Passivate* on A .

premise is not an *Activate* rule. We say that a proof in minimal classical natural deduction is *normal* if any occurrence of Modus Ponens in the proof is normal (this is the same definition as for minimal non-classical natural deduction) and if any occurrence of *Passivate* is normal also.

Theorem 2 (Parigot). *If $\Gamma \vdash_{MC} A; \Delta$ is provable then there is a normal proof of $\Gamma \vdash_{MC} A; \Delta$*

$\frac{}{\Gamma, A \vdash_C A; \Delta} Ax$	$\frac{\Gamma \vdash_C A; \Delta}{\Gamma \vdash_C A; \Delta} Activate$	$\frac{\Gamma \vdash_C A; A, \Delta}{\Gamma \vdash_C A; \Delta} Passivate$
$\frac{\Gamma \vdash_C \perp; \Delta}{\Gamma \vdash_C \Delta} \perp_e$	$\frac{\Gamma, A \vdash_C B; \Delta}{\Gamma \vdash_C A \rightarrow B; \Delta} \rightarrow_i$	$\frac{\Gamma \vdash_C A \rightarrow B; \Delta \quad \Gamma \vdash_C A; \Delta}{\Gamma \vdash_C B; \Delta} \rightarrow_e$

Fig. 4. Classical Natural Deduction

Classical Logic. To obtain full classical logic from minimal Parigot's classical natural deduction⁴ and thus derive DN, we explicitly add the elimination rule for $\perp$. The (full) *Parigot's classical natural deduction* is described in Figure 4. From Propositions 1, 3 and 4, we directly have:

Proposition 5. $\Gamma \vdash_C A$ iff PL, $\Gamma \vdash_I A$ iff DN, $\Gamma \vdash_M A$ iff EFQ, $\Gamma \vdash_{MC} A$.

We define *normal proofs* for classical natural deduction as for minimal classical natural deduction where the rule $\perp_e$ is normal if its premise is not an *Activate* rule (i.e. $\perp_e$ is considered at the same level as *Passivate*). Parigot's normalisation proof for minimal classical natural deduction applies also for full classical natural deduction.

Theorem 3 (Parigot). *If $\Gamma \vdash_C A; \Delta$ is provable then there is a normal proof of $\Gamma \vdash_C A; \Delta$.*

As expected, full classical logic is conservative over minimal classical logic for formulas not mentioning the $\perp$ formula, as stated by the following consequence of Theorem 3.

Proposition 6. *If $\perp$ does not occur in A then $\vdash_C A$ iff $\vdash_{MC} A$.*

⁴ Parigot's original formulation of classical natural deduction [Par92] does not include the $\perp_e$ -rule but gives direct rules for negation which are easily derivable from the elimination rule for $\perp$.

Remark 1. Minimal classical natural deduction without the *Passivate* rule yields minimal logic, since the context Δ is inert and can only remain empty in a derivation for which the end sequent has the form $\Gamma \vdash A$; (even the *Activate* rule cannot be applied). Similarly, classical natural deduction without the *Passivate* rule yields intuitionistic logic. As a consequence, minimal and intuitionistic natural deduction can both be seen as subsystems of classical natural deduction.

$\frac{}{\Gamma, A \vdash_{RAA} A} Ax$	$\frac{\Gamma \vdash_{RAA} \perp^c}{\Gamma \vdash_{RAA} A} Activate$	$\frac{\Gamma, \neg_c A \vdash_{RAA} \perp^c}{\Gamma \vdash_{RAA} A} RAA_c$
$\frac{\Gamma \vdash_{RAA} \perp}{\Gamma \vdash_{RAA} \perp^c} \perp^c$	$\frac{\Gamma, A \vdash_{RAA} B}{\Gamma \vdash_{RAA} A \rightarrow B} \rightarrow_i$	$\frac{\Gamma \vdash_{RAA} A \rightarrow B \quad \Gamma \vdash_{RAA} A}{\Gamma \vdash_{RAA} B} \rightarrow_e$

Fig. 5. Natural Deduction with RAA_c

Minimal Prawitz Classical Logic. Prawitz defines classical logic as minimal logic plus the Reductio Ad Absurdum rule (RAA) [Pra65]: from $\Gamma, \neg A \vdash \perp$ deduce $\Gamma \vdash A$. This rule implies EFQ (as DN implies EFQ) and hence yields full classical logic. In here we are interested in exploring the possibility of defining minimal classical logic from minimal logic and RAA but without deriving EFQ. Equivalently, we would like to devise a restricted version of EFQ that would allow one to prove PL from $PL_{\perp}$. This alternative formulation of (minimal) classical logic is obtained by distinguishing two different notions of $\perp$: $\perp$ for commands (written as $\perp^c$) and $\perp$ for terms (see Figure 5 where $\neg_c A$ stands for $A \rightarrow \perp^c$). If the context Δ is the set of formulas $A_1, \dots, A_n$, then we write $\neg_c \Delta$ for the set $\neg_c A_1, \dots, \neg_c A_n$. Sequents are of the form $\Gamma, \neg_c \Delta \vdash A$ or $\Gamma, \neg_c \Delta \vdash \perp^c$ and $\perp^c$ is not allowed to occur in Γ , Δ and A . The minimal subset does not contain the $\perp^c$ rule and is denoted by $\vdash_{MRAA}$.

Proposition 7. *Given a formula A and contexts Γ and Δ , all with no occurrences of $\perp^c$, we have*

1. $\Gamma \vdash_{MC} A; \Delta \text{ iff } \Gamma, \neg_c \Delta \vdash_{MRAA} A$.
2. $\Gamma \vdash_C A; \Delta \text{ iff } \Gamma, \neg_c \Delta \vdash_{RAA} A$.

3 Computational Content of Minimal Logic + Double Negation

To reason about Scheme programs, Felleisen *et al.* [FH92] introduced the $\lambda\mathcal{C}$ calculus. $\mathcal{C}$ provides *abortive continuations*: the invocation of a continuation reinstates the captured context *in place* of the current one. Griffin was the first to observe that $\mathcal{C}$ is typable with $\neg\neg A \rightarrow A$. This extended the Curry-Howard

$$\begin{array}{c}
M ::= x \mid \lambda x.M \mid MM \mid (CM) \\
\\
\frac{}{\Gamma, x : A \vdash x : A} Ax \qquad \frac{\Gamma \vdash M : \neg\neg A}{\Gamma \vdash \mathcal{C}(M) : A} DN \\
\\
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \rightarrow B} \rightarrow_i \qquad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash M' : A}{\Gamma \vdash MM' : B} \rightarrow_e
\end{array}$$

Fig. 6. The $\lambda\mathcal{C}$ calculus

isomorphism to classical logic [Gri90]. The typing system for $\lambda\mathcal{C}$ is given in Figure 6.

Proposition 8 (Griffin). *A formula A is provable in classical logic iff there exists a closed $\lambda\mathcal{C}$ term M such that $\vdash M : A$ is provable.*

Felleisen also developed the $\lambda\mathcal{K}$ calculus which axiomatizes the `callcc` (i.e. call-with-current-continuation) control operator. In contrast to $\mathcal{C}$, $\mathcal{K}$ leaves the current context intact as explicitly described in its usual encoding: $\mathcal{K}(M) = \mathcal{C}(\lambda k.k(Mk))$. $\mathcal{K}$ is not as powerful as $\mathcal{C}$ [Fel90]. In order to define $\mathcal{C}$ we need the abort primitive $\mathcal{A}$ (of type EFQ): $\mathcal{C}(M) = \mathcal{K}(\lambda k.\mathcal{A}(Mk))$. An alternative encoding, $\mathcal{K}(M) = \mathcal{C}(\lambda k.k(M\lambda x.\mathcal{A}(kx)))$, shows that $\mathcal{K}$ can be typed with PL. From Proposition 4, we have:

Proposition 9. *A formula A is provable in minimal classical logic iff there exists a closed $\lambda\mathcal{K}$ term M such that $\vdash M : A$ is provable.*

The call-by-value and call-by-name reduction semantics of $\lambda\mathcal{C}$ are presented in Figure 7. An important point to clarify is the presence of the abort operations in the right-hand sides of the reduction rules. As far as evaluation is concerned, they are not necessary. They are important in order to obtain a satisfying correspondence between the operational and reduction semantics. For example, the term $\mathcal{C}(\lambda k.(k \lambda x.x)N)$ evaluates to $\lambda x.x$. However, the absence of the abort from the reduction rules makes impossible to get rid of the control context $\lambda f.f N$. The abort steps signal that k is not a normal function but is an abortive continuation. As we explain in Section 5, these abort steps are different from the abort used in defining $\mathcal{C}$ in terms of $\mathcal{K}$. The aborts in the reduction rules correspond to throwing to a user defined continuation (i.e. a *Passivate* step), whereas the abort in the definition of $\mathcal{C}$ corresponds to throwing to the predefined top-level continuation (i.e. a $\perp_e$ step).

Remark 2. Parigot in [Par92] criticized Griffin's work because the proposed $\mathcal{C}$ -typing did not fit the operational semantics. Actually, the only rule that breaks subject reduction is the top-level computation rule ($CM \mapsto M(\lambda x.\mathcal{A}(x))$) which forces a conversion from $\perp$ to the top-level type. To solve the problem, instead of reducing M , Griffin proposed to reduce $\mathcal{C}(\lambda\alpha.\alpha M)$, where αM is of type $\perp$. As detailed in the next section, the classical version of Parigot's $\lambda\mu$ requires a similar intervention; a free continuation constant is needed which we call *top*.

$$\begin{array}{l}
\lambda_n\text{-}\mathcal{C} \quad \left\{ \begin{array}{l} \beta : (\lambda x.M)N \rightarrow M[x := N] \\ \mathcal{C}_L : (\mathcal{C}M)N \rightarrow \mathcal{C}(\lambda k.M(\lambda f.\mathcal{A}(k(fN)))) \\ \mathcal{C}_{top} : \mathcal{C}M \rightarrow \mathcal{C}(\lambda k.M(\lambda f.\mathcal{A}(kf))) \\ \mathcal{C}_{idem} : \mathcal{C}(\lambda k.\mathcal{C}M) \rightarrow \mathcal{C}(\lambda k.M(\lambda x.\mathcal{A}(x))) \\ \mathcal{C}_{elim} : \mathcal{C}(\lambda k.kM) \rightarrow M \quad k \notin FV(M) \end{array} \right. \\
\\
\lambda_v\text{-}\mathcal{C} \quad \left\{ \begin{array}{l} \beta : (\lambda x.M)V \rightarrow M[x := V] \\ \mathcal{C}_L : (\mathcal{C}M)N \rightarrow \mathcal{C}(\lambda k.M(\lambda f.\mathcal{A}(k(fN)))) \\ \mathcal{C}_R : V(\mathcal{C}M) \rightarrow \mathcal{C}(\lambda k.M(\lambda x.\mathcal{A}(k(Vx)))) \\ \mathcal{C}_{top} : \mathcal{C}M \rightarrow \mathcal{C}(\lambda k.M(\lambda f.\mathcal{A}(kf))) \\ \mathcal{C}_{idem} : \mathcal{C}(\lambda k.\mathcal{C}M) \rightarrow \mathcal{C}(\lambda k.M(\lambda x.\mathcal{A}(x))) \end{array} \right. \\
V ::= x \mid \lambda x.M
\end{array}$$

Fig. 7. $\lambda_n\text{-}\mathcal{C}$ and $\lambda_v\text{-}\mathcal{C}$ reduction rules

4 Computational Content of Classical Natural Deduction

Figure 8 describes Parigot’s $\lambda\mu$ calculus [Par92] which is a term assignment for his classical natural deduction. The *Passivate* rule reads as follows: given a term producing a value of type A , if α is a continuation variable waiting for something of type A (*i.e.* A cont), then by invoking the continuation variable we leave the current context. Terms of the form $[\alpha]t$ are called *commands*. The *Activate* rule reads as follows: given a command (*i.e.* no formula is focused) we can select which result to get by capturing the associated continuation. If A^α is not present in the precondition then the rule corresponds to weakening. Note that the rule $\perp_e$ differs from Parigot’s version. In [Par92], the elimination rule for $\perp$ is interpreted by an unnamed term $[\gamma]t$, where γ is any continuation variable (not always the same for every instance of the rule). In contrast, the rule is here systematically associated to the same primitive continuation variable, top , considered as a constant. This was also observed by Streicher *et al.* [SR98]. Parigot would represent DN as the term $\lambda y.\mu\alpha.[\gamma](y\lambda x.\mu\delta.[\alpha]x)$ whereas our representation is $\lambda y.\mu\alpha.[top](y\lambda x.\mu\delta.[\alpha]x)$. We use $\lambda\mu\text{-}top$ to denote the whole calculus with $\perp_e$ and $\lambda\mu$ to denote the calculus without $\perp_e$. The need for an

$$\begin{array}{c}
t :: x \mid \lambda x.t \mid tt \mid \mu\alpha.c \\
c ::= [\beta]t \mid [top]t \\
\\
\frac{}{\Gamma, A^x \vdash x : A; \Delta} Ax \quad \frac{c : \Gamma \vdash; A^\alpha, \Delta}{\Gamma \vdash \mu\alpha.c : A; \Delta} Activate \quad \frac{\Gamma \vdash t : A; A^\alpha, \Delta}{[\alpha]t : \Gamma \vdash; A^\alpha, \Delta} Passivate \\
\\
\frac{\Gamma \vdash t : \perp; \Delta}{[top]t : \Gamma \vdash; \Delta} \perp_e \quad \frac{\Gamma \vdash t : A \rightarrow B; \Delta \quad \Gamma \vdash s : A; \Delta}{\Gamma \vdash ts : B; \Delta} \rightarrow_e \quad \frac{\Gamma, A^x \vdash t : B; \Delta}{\Gamma \vdash \lambda x.t : A \rightarrow B; \Delta} \rightarrow_i
\end{array}$$

Fig. 8. $\lambda\mu$ and $\lambda\mu\text{-}top$ calculi

extra continuation constant to interpret the elimination of $\perp$ can be emphasized by the following statement:

Proposition 10. *A formula A is provable in minimal classical logic (resp. classical logic) iff there exists a closed $\lambda\mu$ term (resp. $\lambda\mu$ -top term) t such that $\vdash t : A$ is provable.*

$\lambda\mu_n$ and $\lambda\mu_n\text{-top}$	$\left\{ \begin{array}{ll} \text{Logical rule:} & (\lambda x.t)s \rightarrow t[x := s] \\ \text{Structural rule:} & (\mu\alpha.t)s \rightarrow (\mu\alpha.t[[\alpha](ws)/[\alpha]w]) \\ \text{Renaming rule:} & \mu\alpha.[\beta]\mu\gamma.u \rightarrow \mu\alpha.u[\beta/\gamma] \\ \text{Simplification rule:} & \mu\alpha.[\alpha]u \rightarrow u \quad \alpha \notin FV(u) \end{array} \right.$
$\lambda\mu_v$ and $\lambda\mu_v\text{-top}$ ($v ::= x \mid \lambda x.t$)	$\left\{ \begin{array}{ll} \text{Logical rule:} & (\lambda x.t)v \rightarrow t[x := v] \\ \text{Left structural rule:} & (\mu\alpha.t)s \rightarrow (\mu\alpha.t[[\alpha](ws)/[\alpha]w]) \\ \text{Right structural rule:} & v(\mu\alpha.t) \rightarrow (\mu\alpha.t[[\alpha](vw)/[\alpha]w]) \\ \text{Renaming rule:} & \mu\alpha.[\beta]\mu\gamma.u \rightarrow \mu\alpha.u[\beta/\gamma] \\ \text{Simplification rule:} & \mu\alpha.[\alpha]u \rightarrow u \quad \alpha \notin FV(u) \end{array} \right.$

Fig. 9. Call-by-name and call-by-value $\lambda\mu$ and $\lambda\mu$ -top reduction rules

We write $\lambda\mu_n$ and $\lambda\mu_v$ (resp. $\lambda\mu_n\text{-top}$ and $\lambda\mu_v\text{-top}$) for the $\lambda\mu$ calculus (resp. $\lambda\mu$ -top calculus) equipped with call-by-name and call-by-value reduction rules, respectively. The reduction rules are given in Figure 9 (substitutions $[[\alpha](ws)/[\alpha]w]$ and $[[\alpha](sw)/[\alpha]w]$ are defined as in [Par92]). Note that the rules are the same for the $\lambda\mu$ and $\lambda\mu\text{-top}$ calculi. $\lambda\mu_n$ is Parigot's original calculus, while our presentation of $\lambda\mu_v$ is similar to Ong and Stewart [OS97]. Both sets of reduction rules are well-typed and enjoy subject reduction.

Instead of showing a correspondence between the $\lambda\mu\text{-top}$ calculi and the $\lambda\mathcal{C}$ calculi as in [dG94], we have searched for an isomorphic calculus. This turns out to be interesting in its own right since it extends the expressive power of Felleisen $\lambda\mathcal{C}$ and provides a term assignment for Prawitz's classical logic.

5 Computational Content of Prawitz's Classical Deduction

We consider a restricted form of $\lambda\mathcal{C}$, called $\lambda\mathcal{C}^-\text{-top}$. Its typing system is given in Figure 10. In $\lambda\mathcal{C}^-\text{-top}$, we distinguish between capturing a continuation and expressing where to go next. We assume the existence of a top-level continuation called *top*. The control operator $\mathcal{C}^-$ can only be applied to a lambda abstraction. Moreover, the body of a $\mathcal{C}^-$ -lambda abstraction is always of the form kM for a continuation variable k . In $\lambda\mathcal{C}^-\text{-top}$, $\mathcal{K}$ and $\mathcal{C}$ are expressed as $\mathcal{C}^-(\lambda k.k M)$ and $\mathcal{C}^-(\lambda k.top M)$, respectively. In $\lambda\mathcal{C}^-\text{-top}$, it is possible to distinguish between aborting a computation and throwing to a continuation. For example, one would write $\mathcal{C}^-(\lambda d.top M)$ to abort the computation M and $\mathcal{C}^-(\lambda d.k M)$ to invoke

continuation k with M (d not free in M). Variables and continuation variables are kept distinct. The translation from $\lambda\text{-}\mathcal{C}$ to $\lambda\text{-}\mathcal{C}^- \text{-top}$ is given in Figure 11. The call-by-name and call-by-value $\lambda\text{-}\mathcal{C}^- \text{-top}$ reduction rules are given in Figure 12. Note that one does not need the $\mathcal{C}_{top}$ -rule, whose action is to wrap up an application of a continuation with a throw operation. $\mathcal{C}_{idem}^-$ is a generalization of $\mathcal{C}_{idem}$, which is obtained by instantiating the continuation variable k' to top (i.e. the continuation $\lambda x.\mathcal{A}(x): \mathcal{C}^-(\lambda k.top \mathcal{C}(\lambda q.M)) \rightarrow \mathcal{C}^-(\lambda k.M[top/q])$). $\mathcal{C}_{idem}^-$ is similar to the rule proposed by Barbanera *et al.* [BB93]: $M(\mathcal{C}N) \rightarrow N(\lambda a.(Ma))$, where M has type $\neg A$. Felleisen proposed in [FH92] the following additional rules for $\lambda_v\text{-}\mathcal{C}$: $\mathcal{C}_E: E[\mathcal{C}M] \rightarrow \mathcal{C}(\lambda k.M(\lambda x.\mathcal{A}(k E[x])))$ (where E stands for a call-by-value evaluation context) and $\mathcal{C}_{elim}: \mathcal{C}(\lambda k.k M) \rightarrow M$, where k is not free in M . The first rule is a generalization of $\mathcal{C}_L$, $\mathcal{C}_R$, and $\mathcal{C}_{top}$ which adds expressive power to the calculus. The second rule, which also appears in [Hof95], leads to better simulation of evaluation. However, both rules destroy confluence of $\lambda_v\text{-}\mathcal{C}$. Felleisen left unresolved the problem of finding an extended theory that would include $\mathcal{C}_E$ or $\mathcal{C}_{elim}$ and still satisfy the classical properties of reduction theories. $\mathcal{C}_{elim}$ is already present in our calculi and $\mathcal{C}_E$ is derivable. Thus one may consider our calculi as a solution.

$$\begin{array}{c}
M ::= x \mid MM \mid \lambda x.M \mid \mathcal{C}^-(\lambda k.N) \\
N ::= k'M \mid topM \\
\\
\frac{}{\Gamma, x : A \vdash x : A} Ax \quad \frac{\Gamma \vdash N : \perp^c}{\Gamma \vdash \mathcal{C}^-(\lambda q.N) : A} Activate \quad \frac{\Gamma, k : \neg_c A \vdash N : \perp^c}{\Gamma \vdash \mathcal{C}^-(\lambda k.N) : A} RAA_c \\
\\
\frac{\Gamma \vdash M : \perp}{\Gamma \vdash topM : \perp^c} \perp_e^c \quad \frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash M' : A}{\Gamma \vdash MM' : B} \rightarrow_e \quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \rightarrow B} \rightarrow_i
\end{array}$$

Fig. 10. $\lambda\text{-}\mathcal{C}^-$ and $\lambda\text{-}\mathcal{C}^- \text{-top}$ calculi

$$\overline{\overline{x}} = x \quad \overline{\overline{\lambda x.M}} = \lambda x.\overline{\overline{M}} \quad \overline{\overline{MN}} = \overline{\overline{M}} \overline{\overline{N}} \quad \overline{\overline{\mathcal{C}M}} = \mathcal{C}^-(\lambda k.top(\overline{\overline{M}}(\lambda x.\mathcal{C}^-(\lambda \delta.kx))))$$

Fig. 11. Translation from $\lambda\text{-}\mathcal{C}$ to $\lambda\text{-}\mathcal{C}^- \text{-top}$

- Proposition 11.** 1. $\lambda_v\text{-}\mathcal{C}^- \text{-top}$ and $\lambda_n\text{-}\mathcal{C}^- \text{-top}$ are confluent and strongly normalizing.
2. Subject reduction: Given $\lambda_v\text{-}\mathcal{C}^- \text{-top}$ ($\lambda_n\text{-}\mathcal{C}^- \text{-top}$) terms M, N , if $\Gamma \vdash M : A$ and $M \twoheadrightarrow N$ then $\Gamma \vdash N : A$.

Soundness and completeness properties for $\lambda_v\text{-}\mathcal{C}^- \text{-top}$ with respect to $\lambda_v\text{-}\mathcal{C}$ are stated below, where $\simeq_c$ denotes operational equivalence as defined in [FH92].

A $\lambda_v\mathcal{C}^-$ -top term M is translated into a $\lambda_v\mathcal{C}$ term $\overline{M}$ by simply replacing $\mathcal{C}^-$ with $\mathcal{C}$ and by erasing the references to the top continuation.

Proposition 12. 1. Given $\lambda_v\mathcal{C}$ terms M and N , if $M \twoheadrightarrow N$ then $\overline{M} \twoheadrightarrow \overline{N}$.
2. Given $\lambda_v\mathcal{C}^-$ -top terms M and N , if $M \twoheadrightarrow N$ then $\overline{M} \simeq_c \overline{N}$.

Relation between the $\lambda\mu$ -top and the $\lambda\mathcal{C}^-$ -top calculi. The $\lambda\mathcal{C}^-$ -top calculus has been designed in such a way that it is in one-to-one correspondence with the $\lambda\mu$ -top calculus. The correspondence is given by $\overline{\lambda x.t} = \lambda x.\overline{t}$, $\overline{ts} = \overline{t}\overline{s}$, $\overline{\mu\alpha.[\gamma]t} = \mathcal{C}^-(\lambda\alpha.\gamma\overline{t})$. This correspondence extends to the reduction rules (Figure 12 matches Figure 9), as expressed by the following statement:

Proposition 13. Let t, s be $\lambda\mu$ -top-terms, then $t \rightarrow_{\lambda\mu\text{-top}} s$ iff $\overline{t} \rightarrow_{\lambda_n\mathcal{C}^- \text{-top}} \overline{s}$ and $t \rightarrow_{\lambda_{\mu\nu}\text{-top}} s$ iff $\overline{t} \rightarrow_{\lambda_v\mathcal{C}^- \text{-top}} \overline{s}$.

$\lambda_n\mathcal{C}^-$ and $\lambda_n\mathcal{C}^- \text{-top}$	$\begin{cases} \beta : & (\lambda x.M)N & \rightarrow M[x := N] \\ \mathcal{C}_L^- : & \mathcal{C}^-(\lambda k.M)N & \rightarrow \mathcal{C}^-(\lambda k.M[k(PN)/kP]) \\ \mathcal{C}_{idem}^- : & \mathcal{C}^-(\lambda k.k'\mathcal{C}^-(\lambda q.N)) & \rightarrow \mathcal{C}^-(\lambda k.N[k'/q]) \\ \mathcal{C}_{elim}^- : & \mathcal{C}^-(\lambda k.kM) & \rightarrow M \quad k \notin FV(M) \end{cases}$
$\lambda_v\mathcal{C}^-$ and $\lambda_v\mathcal{C}^- \text{-top}$ ($V ::= x \mid \lambda x.M$)	$\begin{cases} \beta : & (\lambda x.M)V & \rightarrow M[x := V] \\ \mathcal{C}_{elim}^- : & \mathcal{C}^-(\lambda k.kM) & \rightarrow M \quad k \notin FV(M) \\ \mathcal{C}_L^- : & \mathcal{C}^-(\lambda k.M)N & \rightarrow \mathcal{C}^-(\lambda k.M[k(PN)/kP]) \\ \mathcal{C}_R^- : & V\mathcal{C}^-(\lambda k.M) & \rightarrow \mathcal{C}^-(\lambda k.M[k(VP)/kP]) \\ \mathcal{C}_{idem}^- : & \mathcal{C}^-(\lambda k.k'\mathcal{C}^-(\lambda q.N)) & \rightarrow \mathcal{C}^-(\lambda k.N[k'/q]) \end{cases}$

Fig. 12. Call-by-name and call-by-value $\lambda\mathcal{C}^-$ and $\lambda\mathcal{C}^- \text{-top}$ reduction rules

Remark 3. Reducing the term corresponding to $\mathcal{C}(\lambda k.kIx)1$ we have:

$$\begin{aligned} (\mu\alpha.[\text{top}](\lambda k.kIx)(\lambda f.\mu\delta.[\alpha]f))1 &\rightarrow (\mu\alpha.[\text{top}]((\lambda f.\mu\delta.[\alpha]f)Ix))1 \rightarrow (\mu\alpha.[\text{top}]((\mu\delta.[\alpha]I)x))1 \\ &\rightarrow (\mu\alpha.[\text{top}](\mu\delta.[\alpha]I))1 \rightarrow (\mu\alpha.[\text{top}](\mu\delta.[\alpha](I1))) \rightarrow (\mu\alpha.[\alpha](I1)) \rightarrow (\mu\alpha.[\alpha]1) \rightarrow 1 \end{aligned}$$

This reduction sequence is better than the corresponding sequence in $\lambda\mathcal{C}$.

Proposition 14. A formula A is provable in Prawitz's classical logic iff there exists a closed $\lambda\mathcal{C}^- \text{-top}$ term M such that $\vdash M : A$ is provable.

We define a subset of $\lambda\mathcal{C}^- \text{-top}$, which does not allow one to abort a computation, i.e. terms of the form $\mathcal{C}^-(\lambda k.\text{top}M)$ are not allowed. We call this subset, which is isomorphic to $\lambda\mu$, $\lambda\mathcal{C}^-$.

Proposition 15. A formula A is provable in minimal Prawitz classical logic iff there exists a closed $\lambda\mathcal{C}^-$ term M such that $\vdash M : A$ is provable.

Remark 4. The $\lambda\mathcal{C}^-$ term representing PL is $\lambda y.\mathcal{C}^-(\lambda k.k(y(\lambda x.\mathcal{C}^-(\lambda q.kx))))$, which can be written in ML as:

```
- fun PL y = callcc (fn k => (y (fn x => throw k x)));
val PL = fn : (('a -> 'b) -> 'a) -> 'a
```

Notice how the throw construct corresponds to a weakening step. By Propositions 7, 9 and 15, $\lambda\text{-}\mathcal{C}^-$ is equivalent to $\lambda\text{-}\mathcal{K}$, assuming `callcc` is typed with PL, say `callccpl`. However, it might not be at all obvious how to use a continuation in different contexts, since we do not have weakening available. Consider for example the following ML term (with `callcc` and `throw` typable as in [DHM91]):

```
- callcc (fn k => if (throw k 1) then 7 else (throw k 99));
```

We use the continuation in both boolean and integer contexts. How can we write the above expression without making use of weakening or throw? The proof of Proposition 4 gives the answer:

```
- callcc_pl (fn k => callcc_pl (fn q => if q 1 then 7 else k 99));
```

We define a subset of $\lambda\text{-}\mathcal{C}^-$, called $\lambda\text{-}\mathcal{A}^-$, in which expressions of the form $\mathcal{C}^-(\lambda d.qM)$ are only allowed when d is not free in M and q is *top*, that is, we only allow throwing to the top-level continuation.

Proposition 16. *A formula A is provable in intuitionistic logic iff there exists a closed $\lambda\text{-}\mathcal{A}^-$ term M such that $\vdash M : A$ is provable.*

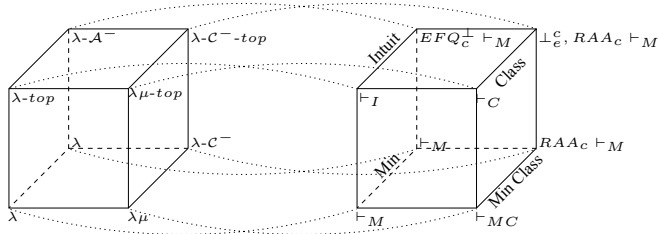
6 Related Work

The relation between Parigot $\lambda\mu$ and $\lambda\text{-}\mathcal{C}$ has been investigated by de Groote [dG94], who only considers the $\lambda\mu$ structural rule but not renaming and simplification. As for $\lambda\text{-}\mathcal{C}$, he only considers $\mathcal{C}_L$ and $\mathcal{C}_{top}$. However, these rules are not the original rules of Felleisen, since they do not contain abort. For example, $\mathcal{C}_{top}$ is $\mathcal{C}M \rightarrow \mathcal{C}(\lambda k.M(\lambda f.kf))$ which is in fact a reduction rule for $\lambda\text{-}\mathcal{F}$ [Fel88]. This work fails in relating $\lambda\mu$ to $\lambda\text{-}\mathcal{C}$ in an untyped framework, since it does not express continuations as abortive functions. It says in fact that $\mathcal{F}$ behaves as $\mathcal{C}$ in the simply-typed case. Ong and Stewart [OS97] also do not consider the abort step in Felleisen's rules. This could be justified because in a simply-typed setting these steps are of type $\perp \rightarrow \perp$. Therefore, it seems we have a mismatch. While the aborts are essential in the reduction semantics, they are irrelevant in the corresponding proof. We are the first to provide a proof theoretic justification for those abort steps, they correspond to the step $\perp \rightarrow \perp_c$. In addition to Ong and Stewart, Py [Py98] and Bierman [Bie98] have pointed out the peculiarity of having an open $\lambda\mu$ term corresponding to a tautology. Their solution is to abolish the distinction between commands and terms. A command is a term returning $\perp$. The body of a μ -abstraction is not restricted to a command, but can be of the form $\mu\alpha.t$, where t is of type $\perp$. Thus, one has $\lambda y.\mu\alpha.(y \lambda x.[\alpha]x) : \neg\neg A \rightarrow A$. We would then represent the term $\mathcal{C}(\lambda k.(kI)x)$ (where I is $\lambda x.x$) as $\mu\alpha.(\alpha I)x$. Whereas $\mathcal{C}(\lambda k.kIx)$ would reduce to $\mathcal{C}(\lambda k.kI)$ according to $\lambda_n\text{-}\mathcal{C}$ and to I in $\lambda\mu_n\text{-}top$, it would be in normal form in their calculus. Thus, their work in relating $\lambda\mu$ to $\lambda\text{-}\mathcal{C}$ only applies to typed $\lambda\text{-}\mathcal{C}$, whereas our work also applies to the untyped case. Crolard [Cro99] studied the relation between Parigot's $\lambda\mu$ and a calculus with a `catch` and `throw` mechanism. He showed that contraction corresponds to the `catch` operator ($\mu\alpha.[\alpha]t = \text{catch } \alpha t$) and weakening corresponds to the `throw` operator ($\mu\delta.[\alpha]t = \text{throw } \alpha t$ for δ not free in t). He only considers

terms of the form $\mu\alpha.[\alpha]t$ and $\mu\beta.[\alpha]t$, where β does not occur free in t . This property is not preserved by the renaming rule, therefore reduction is restricted. We do not require such restrictions on reduction. We can simulate Ong and Stewart's $\lambda\mu$ and Crolard's calculus via this simple translation: $\mu\alpha.t$ becomes $\mu\alpha.[top]t$ and $[\beta]t$ becomes $\mu\delta.[\beta]t$, where δ is not free in t .

7 Conclusions

Our analysis of the logical strengths of EFQ, PL (or EM) and DN has led naturally to a restricted form of classical logic called *minimal classical logic*. Depending on whether EFQ, PL, or both are assumed in minimal logic, we get intuitionistic, minimal classical, or classical logic. Depending on whether or not we admit *Passivate* (RAA_c ⁵) and $\perp_e$ ($\perp_e^c$) in full classical natural deduction (on top of minimal natural deduction), we get the correspondences with the λ -calculus considered in this paper, as summarized above⁶. Among these systems, $\lambda\mathcal{C}^-$ -top is a confluent extension of Felleisen's theory of control.



We also have some preliminary results regarding $\mathcal{F}$ [Fel88] which provides *functional continuations*, meaning that the invocation of a continuation reinstates the captured context *on top* of the current one. When a continuation is applied it acts like an ordinary function. We conjecture that $\mathcal{F}$ is still typable with DN. The difference with $\mathcal{C}$ is that, for $\mathcal{F}$, the $\perp$ type is equated to the top-level type. Therefore, we do not need the throw construct. As before, one could define a calculus $\lambda\mathcal{F}^-$ -top with similar restrictions as for $\lambda\mathcal{C}^-$ -top, but without requiring the use of a throw construct to invoke a continuation. What is interesting is that the reduction rules for call-by-value and call-by-name $\lambda\mathcal{F}^-$ -top would be the same as those given in Figure 12 with $\mathcal{F}^-$ replacing $\mathcal{C}^-$. In [Fel88], Felleisen also introduced the notion of a *prompt*, written $\#$, with the reduction rules $\#_{\mathcal{F}} : \#(\mathcal{F}M) \rightarrow \#(M(\lambda x.x))$ and $\#_v : (\#V) \rightarrow V$. We can define prompt as $\#M = \mathcal{F}^-(\lambda top.top M)$. However, one would need one more reduction rule $(\mathcal{F}^-(\lambda top.M) \rightarrow \mathcal{F}^-(\lambda top.M[\lambda x.x/top]))$ and a proviso (the bound variable k cannot be *top*) on the lifting rules $\mathcal{F}_L^-$ and $\mathcal{F}_R^-$. To see why we need the proviso, consider the term $\mathcal{F}^-(\lambda top.3 * (top2)) + 1$ which reduces to

⁵ with restrictions on the use of $\perp_e$

⁶ λ -top is the subset of $\lambda\mu$ -top in which expressions of the form $\mu\delta.[\alpha]t$ are only allowed when δ is not free in t and α is *top*. $EFQ_c^\perp$ stands for $\perp_e^c$ and the rule $\Gamma \vdash \perp_e$ implies $\Gamma \vdash A$ (i.e. the restriction of RAA_c when $\neg_c A$ is not used in the proof).

7. If we had allowed the left lifting rule we would have obtained 9. We can also extend $\lambda\mathcal{C}^-$ -top with prompt, however, we do not need to extend the system with any additional reduction rules. With $\mathcal{C}^-$ and *prompt* one could also define **shift** and **reset** [DF89,DF90] as $(\text{shift } M) = \mathcal{C}^-(\lambda k.M(\lambda x.\#(kx)))$. This was also observed by Filinsky in [Fil94]. We plan to investigate these additional calculi and to extend our analysis to other control operators. The reader may refer to [Que93] for a complete list of them.

Acknowledgements. We thank Matthias Felleisen for numerous discussions about his theory of control. Miley Semmelroth helped us to improve the presentation of the paper. We also thank the anonymous referees for their comments. The first author has been supported by NSF grant 0204389.

References

- [BB93] F. Barbanera and S. Berardi. Extracting constructive content from classical logic via control-like reductions. In *TLCA'93, LNCS 664*, pages 45–59. 1993.
- [Bie98] G.M. Bierman. A computational interpretation of the lambda-mu calculus. In *MFCS'98, LNCS 1450*, pages 336–345, 1998.
- [Cro99] T. Crolard. A confluent lambda-calculus with a catch/throw mechanism. *Journal of Functional Programming*, 9(6):625–647, 1999.
- [DF89] Olivier Danvy and Andrzej Filinski. A functional abstraction of typed contexts. Technical Report 89/12, 1989.
- [DF90] O. Danvy and A. Filinski. Abstracting control. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming, Nice*, pages 151–160, New York, NY, 1990. ACM.
- [dG94] P. de Groote. On the relation between the lambda-mu calculus and the syntactic theory of sequential control. In *LPAR'94*, pages 31–43. 1994.
- [DHM91] B. F. Duba, R. Harper, and D. MacQueen. Typing first-class continuations in ML. In *POPL'91*, pages 163–173, 1991.
- [Fel88] M. Felleisen. The theory of practice of first-class prompt. In *POPL '88*, pages 180–190, 1988.
- [Fel90] M. Felleisen. On the expressive power of programming languages. In *ESOP '90, LNCS 432*, pages 134–151. 1990.
- [FH92] M. Felleisen and R. Hieb. A revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, 1992.
- [Fil94] Andrzej Filinski. Representing monads. In *Conf. Record 21st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'94, Portland, OR, USA, 17–21 Jan. 1994*, pages 446–457, New York, 1994. ACM Press.
- [Gen69] G. Gentzen. Investigations into logical deduction. In M.E. Szabo, editor, *Collected papers of Gerhard Gentzen*, pages 68–131. North-Holland, 1969.
- [Gri90] T. G. Griffin. The formulae-as-types notion of control. In *POPL'90*, pages 47–57, 1990.
- [Her94] H. Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In *CSL'94, LNCS 933*, 1994.

- [Hof95] M. Hofmann. Sound and complete axiomatization of call-by-value control operators. *Mathematical Structures in Computer Science*, 5:461–482, 1995.
- [Joh36] I. Johansson. Der Minimalkalkül, ein Reduzierter Intuitionistischer Formalismus. *Compositio Mathematica*, 4:119–136, 1936.
- [OS97] C.-H. Luke Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *POPL'97*, pages 215–227. 1997.
- [Par92] M. Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR '92*, pages 190–201, 1992.
- [Pra65] D. Prawitz. *Natural Deduction, a Proof-Theoretical Study*. Almqvist and Wiksell, Stockholm, 1965.
- [Py98] W. Py. *Confluence en $\lambda\mu$ -calcul*. PhD thesis, Université de Savoie, 1998.
- [Que93] Christian Queinnec. A library of high-level control operators. *ACM SIG-PLAN Lisp Pointers*, 6(4):11–26, 1993.
- [SR98] T. Streicher and B. Reus. Classical logic: Continuation semantics and abstract machines. *Journal of Functional Programming*, 8(6):543–572, 1998.

Counterexample-Guided Control^{*}

Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar

EECS Department, University of California, Berkeley
{tah,jhala,rupak}@eecs.berkeley.edu

Abstract. A major hurdle in the algorithmic verification and control of systems is the need to find suitable abstract models, which omit enough details to overcome the state-explosion problem, but retain enough details to exhibit satisfaction or controllability with respect to the specification. The paradigm of counterexample-guided abstraction refinement suggests a fully automatic way of finding suitable abstract models: one starts with a coarse abstraction, attempts to verify or control the abstract model, and if this attempt fails and the abstract counterexample does not correspond to a concrete counterexample, then one uses the spurious counterexample to guide the refinement of the abstract model. We present a counterexample-guided refinement algorithm for solving ω -regular control objectives. The main difficulty is that in control, unlike in verification, counterexamples are strategies in a game between system and controller. In the case that the controller has no choices, our scheme subsumes known counterexample-guided refinement algorithms for the verification of ω -regular specifications. Our algorithm is useful in all situations where ω -regular games need to be solved, such as supervisory control, sequential and program synthesis, and modular verification. The algorithm is fully symbolic, and therefore applicable also to infinite-state systems.

1 Introduction

The key to the success of algorithmic methods for the verification (analysis) and control (synthesis) of complex systems is *abstraction*. Useful abstractions have two desirable properties. First, the abstraction should be *sound*, meaning that if a property (e.g., safety, controllability) is proved for the abstract model of a system, then the property holds also for the concrete system. Second, the abstraction should be *effective*, meaning that the abstract model is not too fine and can be handled by the tools at hand; for example, in order to use conventional model checkers, the abstraction must be both finite-state and of manageable size. Recent research has focused on a third desirable property of abstractions. A sound and effective abstraction (provided it exists) should be found *automatically*; otherwise, the labor-intensive process of constructing suitable abstract models often

^{*} This research was supported in part by the DARPA SEC grant F33615-C-98-3614, the ONR grant N00014-02-1-0671, and the NSF grants CCR-9988172, CCR-0085949, and CCR-0225610.

negates the benefits of automatic methods for verification and control. The most successful paradigm in automatic abstraction is the method of *counterexample-guided abstraction refinement* [5,6,9]. According to that paradigm, one starts with a very coarse abstract model, which is effective but may not be informative, meaning that it may not exhibit the desired property even if the concrete system does. Then the abstract model is refined iteratively as follows: first, if the abstract model does not exhibit the desired property, then an abstract counterexample is constructed automatically; second, it can be checked automatically if the abstract counterexample corresponds to a concrete counterexample; if this is not the case, then, third, the abstract model is refined automatically in order to eliminate the spurious counterexample.

The method of counterexample-guided abstraction refinement has been developed for the *verification* of linear-time properties [9], and universal branching-time properties [10]. It has been applied successfully in both hardware [9] and software verification [6,18]. We develop the method of counterexample-guided abstraction refinement for the *control* of linear-time objectives. In verification, a counterexample to the satisfaction of a linear-time property is a *trace* that violates the property: for safety properties, a finite trace; for general ω -regular properties, an infinite, periodic (lasso-shaped) trace. In control, counterexamples are considerably more complicated: a counterexample to the controllability of a system with respect to a linear-time objective is a *tree* that represents a strategy of the system for violating the property no matter what the controller does. For safety objectives, finite trees are sufficient as counterexamples; for general ω -regular objectives on finite abstract models, infinite trees are necessary, but they can be finitely represented as graphs with cycles, because finite-state strategies are as powerful as infinite-state strategies [17].

In somewhat more detail, our method proceeds as follows. Given a two-player game structure (player 1 “controller” vs. player 2 “system”), we wish to check if player 1 has a strategy to achieve a given ω -regular winning condition. Solutions to this problem have applications in supervisory control [22], sequential hardware synthesis and program synthesis [8,7,21], modular verification [2,4,14], receptiveness checking [3,15], interface compatibility checking [12], and schedulability analysis [1]. We automatically construct an abstraction of the given game structure that is as coarse as possible and as fine as necessary in order for player 1 to have a winning strategy. We start with a very coarse abstract game structure and refine it iteratively. First, we check if player 1 has a winning strategy in the abstract game; if so, then the concrete system can be controlled; otherwise, we construct an abstract player-2 strategy that spoils against all abstract player-1 strategies. Second, we check if the abstract player-2 strategy corresponds to a spoiling strategy for player 2 in the concrete game; if so, then the concrete system cannot be controlled; otherwise, we refine the abstract game in order to eliminate the abstract player-2 strategy. In this way, we automatically synthesize “maximally abstract” controllers, which distinguish two states of the controlled system only if they need to be distinguished in order to achieve the control objective. It should be noted that ω -regular verification problems are but special cases

of ω -regular control problems, where player 1 (the controller) has no choice of moves. Our method, therefore, includes as a special case counterexample-guided abstraction refinement for linear-time *verification*.

Furthermore, our method is fully *symbolic*: while traditional symbolic verification computes fixpoints on the iteration of a transition-precondition operator on regions (symbolic state sets), and traditional symbolic control computes fixpoints on the iteration of a more general, game-precondition operator *Cpre* (controllable *Pre*) [4,20], our counterexample-guided abstraction refinement also computes fixpoints on the iteration of *Cpre* and two additional region operators, called *Focus* and *Shatter*. The *Focus* operator is used to check if an abstract counterexample is genuine or spurious. The *Shatter* operator, which is used to refine an abstract model guided by a spurious counterexample, splits an abstract state into several states. Our top-level algorithm calls only these three system-specific operators: *Cpre*, *Focus*, and *Shatter*. It is therefore applicable not only to finite-state systems but also to infinite-state systems, such as hybrid systems, on which these three operators are computable (termination can be studied as an orthogonal issue along the lines of [13]; clearly, our abstraction-based algorithms terminate in all cases in which the standard, *Cpre*-based algorithms terminate, such as in the control of timed automata [20], and they may terminate in more cases).

In a previous paper, we improved the naive iteration of the “abstract-verify-refine” loop by integrating the construction of the abstract model and the verification process [18]. The improvement is called *lazy abstraction*, because the abstract model is constructed on demand during verification, which results in nonuniform abstractions, where some areas of the state space are abstracted more coarsely than others, and thus guarantees an abstract model that is as small as possible. The lazy-abstraction paradigm can be applied also to the algorithm presented here, which subsumes both verification and control. The details of this, however, need to be omitted for space reasons.

2 Games and Abstraction

Two-player games. Let Λ be a set of labels, and Φ a set of propositions. A (*two-player*) *game structure* $\mathcal{G} = (V_1, V_2, \delta, P)$ consists of two (possibly infinite) disjoint sets V_1 and V_2 of player-1 and player-2 states (let $V = V_1 \cup V_2$ denote the set of all states), a labeled transition relation $\delta \subseteq V \times \Lambda \times V$, and a function $P: V \rightarrow 2^\Phi$ that maps every state to a set of propositions. For every state $v \in V$, we call $L(v) = \{l \in \Lambda \mid \exists w. (v, l, w) \in \delta\}$ the set of *available moves*. In the sequel, i ranges over the set $\{1, 2\}$ of players. Intuitively, at state $v \in V_i$, player i chooses a move $l \in L(v)$, and the game proceeds nondeterministically to some state w satisfying $\delta(v, l, w)$.¹ We require that every player-2 state $v \in V_2$ has an available move, that is, $L(v) \neq \emptyset$. For a move $l \in \Lambda$,

¹ Even if the transition relation is deterministic, abstractions of the game may be nondeterministic.

let $Avl(l) = \{v \in V \mid l \in L(v)\}$ be the set of states in which move l is available. We extend the transition relation to sets via the operators Apr , $Epre$: $2^V \times \Lambda \rightarrow 2^V$ by defining $Apr(X, l) = \{v \in V \mid \forall w. \delta(v, l, w) \Rightarrow w \in X\}$ and $Epre(X, l) = \{v \in V \mid \exists w. \delta(v, l, w) \wedge w \in X\}$. For a proposition $p \in \Phi$, let $[p] = \{v \in V \mid p \in P(v)\}$ and $[\bar{p}] = V \setminus [p]$ be the sets of states in which p is true and false, respectively. We assume that Φ contains a special proposition $init$, which specifies a set $[init] \subseteq V$ containing the initial states. A *run* of the game structure $\mathcal{G}$ is a finite or infinite sequence $v_0 v_1 v_2 \dots$ of states $v_j \in V$ such that for all $j \geq 0$, if v_j is not the last state of the run, then there is a move $l_j \in \Lambda$ with $\delta(v_j, l_j, v_{j+1})$. A *strategy of player i* is a partial function $f_i: V^* \cdot V_i \rightarrow \Lambda$ such that for every state sequence $u \in V^*$ and every state $v \in V_i$, if $L(v) \neq \emptyset$, then $f_i(u \cdot v)$ is defined and $f_i(u \cdot v) \in L(v)$. Intuitively, a player- i strategy suggests, when possible, a move for player i given a sequence of states that end in a player- i state. Given two strategies f_1 and f_2 of players 1 and 2, the *possible outcomes* $\Omega_{f_1, f_2}(v)$ from a state $v \in V$ are runs: a run $v_0 v_1 v_2 \dots$ belongs to $\Omega_{f_1, f_2}(v)$ iff $v = v_0$ and for all $j \geq 0$, either $L(v_j) = \emptyset$ and v_j is the last state of the run, or $v_j \in V_i$ and $\delta(v_j, f_i(v_0 \dots v_j), v_{j+1})$. Note that the last state of a finite outcome is always a player-1 state.

Winning conditions. A *game* $(\mathcal{G}, \Gamma)$ consists of a game structure $\mathcal{G}$ and an objective Γ for player 1. We focus on safety games, and briefly discuss games with more general ω -regular objectives at the very end of the paper. A *safety game* has an objective of the form $\Box \overline{err}$, where $err \in \Phi$ is a proposition which specifies a set $[err] \subseteq V$ of error states. Intuitively, the goal of player 1 is to keep the game in states in which err is false, and the goal of player 2 is to drive the game into a state in which err is true. Moreover, in all games we consider, whenever a dead-end state is encountered, player 1 loses. Formally, a run $v_0 v_1 v_2 \dots$ is *winning for player 1* if it is infinite and for all $j \geq 0$, we have $v_j \notin [err]$. Let Π_1 denote the set of runs that are winning for player 1. In general, an *objective* for player 1 is a set $\Gamma \subseteq (2^\Phi)^\omega$ of infinite words over the alphabet 2^Φ , and Π_1 contains all infinite runs $v_0 v_1 \dots$ such that $P(v_0), P(v_1), \dots \in \Gamma$. The game starts from any initial state. A strategy f_1 is *winning for player 1* if for all strategies f_2 of player 2 and all states $v \in [init]$, we have $\Omega_{f_1, f_2}(v) \subseteq \Pi_1$; that is, all possible outcomes are winning for player 1. Dually, a strategy f_2 is *spoiling for player 2* if for all strategies f_1 of player 1, there is a state $v \in [init]$ such that $\Omega_{f_1, f_2}(v) \not\subseteq \Pi_1$. Note that in our setting, nondeterminism is always on the side of player 2. If the objective Γ is ω -regular, then either player 1 has a winning strategy or player 2 has a spoiling strategy [17]. We say that player 1 *wins* the game if there is a player-1 winning strategy.

Example 1 [EXSAFETY] Figure 1(a) shows an example of a safety game. The white states are player 1 states, and the black ones are player 2 states. The labels on the edges denote moves. The objective is $\Box \bar{p}$, that is, player 1 seeks to avoid the error states $[p]$. The player 1 states 1, 2, and 3 are the initial states, i.e., we wish player 1 to win from all three states. Note that in fact player 1 does win from the states 1, 2, and 3: at state 1, she plays the move C ; at 2, she plays A ;

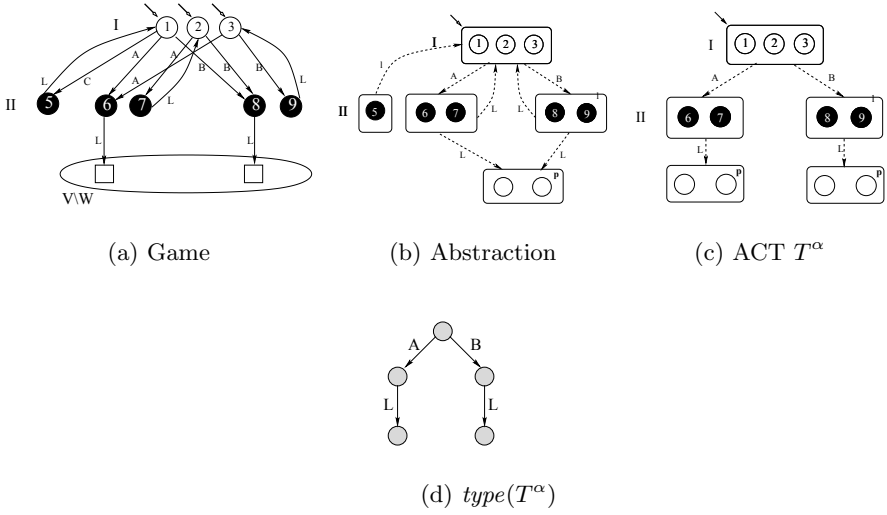


Fig. 1. Example EXSAFETY

and at 3, she plays B . In each case, the only move L available to player 2 brings the game back to the original state. This ensures that the game never reaches a state in $[p]$. ■

The (*player-1*) *controllable predecessor* operator $Cpre_1: 2^V \rightarrow 2^V$ denotes, for a set $X \subseteq V$ of states, the states from which player 1 can force the game into X in one step. Player 1 can force the game into X from a state $v \in V_1$ iff there is *some* available move l such that all l -successors of v are in X , and player 1 can force the game into X from a state $v \in V_2$ iff for *all* available moves l , all l -successors of v are in X . Formally:

$$Cpre_1(X) = (V_1 \cap \bigcup_{l \in A} (Avl(l) \cap Apre(X, l))) \cup (V_2 \cap \bigcap_{l \in A} Apre(X, l))$$

In particular, the set of states from which player 1 can keep the game away from *err* states is the greatest fixpoint $\nu X. [\overline{err}] \cap Cpre_1(X)$. Hence player 1 wins the safety game with objective $\Box \overline{err}$ iff $[init] \subseteq (\nu X. [\overline{err}] \cap Cpre_1(X))$.

Abstractions of games. Since solving a game may be expensive, we wish to construct sound abstractions of the game with smaller state spaces. Soundness means that if player 1 wins the abstract game, then she wins also the original, concrete game. To ensure soundness, we restrict the power of player 1 and increase the power of player 2 [19]. Therefore, we abstract the player-1 states so that *fewer* moves are available, and the player-2 states so that *more*

moves are available. An *abstraction* $\mathcal{G}^\alpha$ for the game structure $\mathcal{G}$ is a game structure $(V_1^\alpha, V_2^\alpha, \delta^\alpha, P^\alpha)$ and a concretization function $\llbracket \cdot \rrbracket: V^\alpha \rightarrow 2^V$ (where $V^\alpha = V_1^\alpha \cup V_2^\alpha$ is the abstract state space) such that conditions (1)–(3) hold. (1) The abstraction preserves the player structure and propositions: for $i \in \{1, 2\}$ and all $v^\alpha \in V_i^\alpha$, we have $\llbracket v^\alpha \rrbracket \subseteq V_i$; for all $v^\alpha \in V^\alpha$, if $v, v' \in \llbracket v^\alpha \rrbracket$, then $P(v) = P(v')$ and $P^\alpha(v^\alpha) = P(v)$. (2) The abstract states cover the concrete state space: $\bigcup_{v^\alpha \in V^\alpha} \llbracket v^\alpha \rrbracket = V$. (3) For each player-1 abstract state $v^\alpha \in V_1^\alpha$, define $L^\alpha(v^\alpha) = \bigcap_{v \in \llbracket v^\alpha \rrbracket} L(v)$, and for each player-2 abstract state $v^\alpha \in V_2^\alpha$, define $L^\alpha(v^\alpha) = \bigcup_{v \in \llbracket v^\alpha \rrbracket} L(v)$. Then, for all $v^\alpha, w^\alpha \in V^\alpha$ and all $l \in \Lambda$, we have $\delta^\alpha(v^\alpha, l, w^\alpha)$ iff $l \in L^\alpha(v^\alpha)$ and there are states $v \in \llbracket v^\alpha \rrbracket$ and $w \in \llbracket w^\alpha \rrbracket$ with $\delta(v, l, w)$. Note that the abstract state space V^α and the concretization function $\llbracket \cdot \rrbracket$ uniquely determine the abstraction $\mathcal{G}^\alpha$. Intuitively, each abstract state $v^\alpha \in V^\alpha$ represents a set $\llbracket v^\alpha \rrbracket \subseteq V$ of concrete states. We will use only abstractions with finite state spaces. The controllable predecessor operator on the abstract game structure $\mathcal{G}^\alpha$ is denoted $Cpre_1^\alpha$.

Proposition 1 [Soundness of abstraction] *Let $\mathcal{G}^\alpha$ be an abstraction for a game structure $\mathcal{G}$, and let Γ be an objective for player 1. If player 1 wins the abstract game $(\mathcal{G}^\alpha, \Gamma)$, then player 1 also wins the concrete game $(\mathcal{G}, \Gamma)$.*

Example 2 [EXSAFETY] Figure 1(b) shows one particular abstraction for the game structure from Figure 1(a). The boxes denote abstract states with the states they represent drawn inside them. The dashed arrows are the abstract transitions. Note that from the starting player 1 box, the move C is not available, because it is not available at states 2 and 3. as not all the states in the box can do it. In the abstract game, player 2 has a spoiling strategy: after player 1 plays either move A or move B , player 2 can play move L and take the game to the error set $[p]$. ■

3 Counterexample-Guided Abstraction Refinement

A *counterexample* to the claim that player 1 can win a game is a spoiling strategy for player 2. A counterexample for an abstract game $(\mathcal{G}^\alpha, \Gamma)$ may be either *genuine*, meaning that it corresponds to a counterexample for the concrete game $(\mathcal{G}, \Gamma)$, or *spurious*, meaning that it arises due to the coarseness of the abstraction. In the sequel, we check whether or not an abstract counterexample is genuine for a fixed safety game $(\mathcal{G}, \Box err)$ and abstraction $\mathcal{G}^\alpha$. Moreover, if the counterexample is spurious, then we refine the abstraction in order to rule out that particular counterexample.

Abstract counterexample trees. Our abstract games are finite-state, and for safety games, memoryless spoiling strategies suffice for player 2. Finite trees are therefore a natural representation of counterexamples. We work with rooted, directed, finite trees with labels on both nodes and edges. Each node is labeled

by an abstract state $v^\alpha \in V^\alpha$ or a concrete state $v \in V$, and possibly a set $r \subseteq V$ of concrete states. We write $\mathbf{n}:v^\alpha$ for node $\mathbf{n}$ labeled with v^α , and $\mathbf{n}:v^\alpha:r$ if $\mathbf{n}$ is labeled with both v^α and r . Each edge is labeled with a move $l \in \Lambda$. If $\mathbf{n} \xrightarrow{l} \mathbf{n}'$ is an edge labeled by l , then $\mathbf{n}'$ is called an l -child of $\mathbf{n}$. A leaf is a node without children. For two trees S and T , we write $S \preceq T$ iff S is a connected subgraph of T which contains the root of T . The *type* of a labeled tree T results from T by removing all node labels (but keeping all edge labels). Furthermore, $\text{Subtypes}(T) = \{\text{type}(S) \mid S \preceq T\}$. An *abstract counterexample tree* (ACT) T^α is a finite tree whose nodes are labeled by abstract states such that conditions (1)–(4) hold. (1) If the root is labeled by v^α , then $\llbracket v^\alpha \rrbracket \subseteq [\text{init}]$. (2) If $\mathbf{n}':w^\alpha$ is an l -child of $\mathbf{n}:v^\alpha$, then $(v^\alpha, l, w^\alpha) \in \delta^\alpha$. (3) If node $\mathbf{n}:v^\alpha$ is a nonleaf player-1 node (that is, $v^\alpha \in V_1^\alpha$), then for *each* move $l \in L^\alpha(v^\alpha)$, the node $\mathbf{n}$ has at least one l -child. Note that if node $\mathbf{n}:v^\alpha$ is a nonleaf player-2 node ($v^\alpha \in V_2^\alpha$), then for *some* move $l \in L^\alpha(v^\alpha)$, the node $\mathbf{n}$ has at least one l -child. (4) If a leaf is labeled by v^α , then either $v^\alpha \in V_1^\alpha$ and $L^\alpha(v^\alpha) = \emptyset$, or $\llbracket v^\alpha \rrbracket \subseteq [\text{err}]$. Intuitively, T^α corresponds to a *set* of spoiling strategies for player 2 in the abstract safety game.

Example 3 [EXSAFETY] Figure 1(c) shows an ACT T^α for the abstract game of Figure 1(b), and Figure 1(d) shows the type of T^α . After player 1 plays either move A or move B , player 2 plays L to take the game to the error set. ■

Concretizing abstract counterexamples. A *concrete counterexample tree* (CCT) S is a finite tree whose nodes are labeled by concrete states such that conditions (1)–(4) hold. (1) If the root is labeled by v , then $v \in [\text{init}]$. (2) If $\mathbf{n}':w$ is an l -child of $\mathbf{n}:v$, then $(v, l, w) \in \delta$. (3) If node $\mathbf{n}:v$ is a nonleaf player-1 node ($v \in V_1$), then for each move $l \in L(v)$, the node $\mathbf{n}$ has at least one l -child. (4) If a leaf is labeled by v , then either $v \in V_1$ and $L(v) = \emptyset$, or $v \in [\text{err}]$. The CCT S *realizes* the ACT T^α if $\text{type}(S) \in \text{Subtypes}(T^\alpha)$ and for each node $\mathbf{n}:w$ of S and corresponding node $\mathbf{n}:v^\alpha$ of T^α , we have $w \in \llbracket v^\alpha \rrbracket$. The ACT T^α is *genuine* if there is a CCT that realizes T^α , and otherwise T^α is *spurious*. To determine if the ACT T^α is genuine, we annotate every node $\mathbf{n}:v^\alpha$ of T^α , in addition, with a set $r \subseteq \llbracket v^\alpha \rrbracket$ of concrete states; that is, $\mathbf{n}:v^\alpha:r$. The result is called an *annotated ACT*. The set r represents an overapproximation for the set of states that can be part of a CCT with a type in $\text{Subtypes}(T^\alpha)$. Initially, $r = \llbracket v^\alpha \rrbracket$. The overapproximation r is sharpened repeatedly by application of a symbolic operator called *Focus*. For a node $\mathbf{n}$ of T^α , let $C(\mathbf{n}) = \{l \in \Lambda \mid \mathbf{n} \text{ has an } l\text{-child}\}$ be the set of moves that label the outgoing edges of $\mathbf{n}$. For each move $l \in C(\mathbf{n})$, let $\{\mathbf{n}_{l,j}:v_{l,j}^\alpha:r_{l,j}\}$ be the set of l -children of $\mathbf{n}$ (indexed by j). The operator $\text{Focus}(\mathbf{n}:v^\alpha:r)$ returns a subset of r :

$$\text{Focus}(\mathbf{n}:v^\alpha:r) = \begin{cases} r & \text{if } \mathbf{n} \text{ leaf and } L^\alpha(v^\alpha) \neq \emptyset \\ r \cap \left(\bigcap_{l \in C(\mathbf{n})} \text{Epre}(\bigcup_j r_{l,j}, l) \right) \cap \left(\bigcap_{l \notin C(\mathbf{n})} \overline{\text{Avl}(l)} \right) & \text{if } \mathbf{n} \text{ other player-1 node} \\ r \cap \left(\bigcup_{l \in C(\mathbf{n})} \text{Epre}(\bigcup_j r_{l,j}, l) \right) & \text{if } \mathbf{n} \text{ player-2 node} \end{cases}$$

Algorithm 1 $\text{AnalyzeCounterex}(T^\alpha)$

Input: an abstract counterexample tree T^α with root $\mathbf{n}_0$.

Output: if T^α is spurious, then SPURIOUS and an annotation of T^α ; otherwise GENUINE.

for each node $\mathbf{n}:v^\alpha$ of T^α **do** annotate $\mathbf{n}:v^\alpha$ by $\llbracket v^\alpha \rrbracket$

while there is some node $\mathbf{n}:v^\alpha:r$ with $r \neq \text{Focus}(\mathbf{n}:v^\alpha:r)$ **do**

 replace the annotation r of $\mathbf{n}:v^\alpha:r$ by $\text{Focus}(\mathbf{n}:v^\alpha:r)$

if $r_0 = \emptyset$ for the annotated root $\mathbf{n}_0 : \_ : r_0$ **then return** (SPURIOUS, T^α with annotations)

end while

return GENUINE

An application of $\text{Focus}(\mathbf{n}:v^\alpha:r)$ sharpens the set r by determining which of the states in r actually have successors that can be part of a spoiling strategy for player 2 in the concrete game. For leaves $\mathbf{n}:v^\alpha:r$ with $L^\alpha(v^\alpha) \neq \emptyset$, it must be that every state in r is an error state, and so can be part of a CCT. For all other player-1 nodes $\mathbf{n}:v^\alpha:r$, a state $v \in r$ can be part of a CCT only if (i) all moves available at v are contained in $C(\mathbf{n})$ and (ii) for every available move l , there is an l -child from which player 2 has a spoiling strategy; that is, for every available move l , the state v must have a successor in the union of all l -children's overapproximations. For player-2 nodes $\mathbf{n}:v^\alpha:r$, a state $v \in r$ can be part of a CCT only if there is some child from which player 2 has a spoiling strategy; that is, the state v must have a successor in the union of all children's overapproximations.

The procedure AnalyzeCounterex (Algorithm 1) iterates the Focus operator on the nodes of a given ACT T^α until there is no change. Let $\text{Focus}^*(\mathbf{n})$ denote the fixpoint value of the annotation for node $\mathbf{n}$ of T^α . For the root $\mathbf{n}_0$ of T^α , if $\text{Focus}^*(\mathbf{n}_0)$ is empty, then T^α is spurious. Otherwise, consider the annotated ACT that results from T^α by annotating each node $\mathbf{n}$ with $\text{Focus}^*(\mathbf{n})$, and removing all nodes $\mathbf{n}$ for which $\text{Focus}^*(\mathbf{n})$ is empty. This annotated ACT has a type in $\text{Subtypes}(T^\alpha)$, and moreover, its annotations contain exactly the states that can be part of a CCT that realizes T^α . Consequently, if $\text{Focus}^*(\mathbf{n}_0)$ is nonempty, then T^α is genuine, and the result of the procedure AnalyzeCounterex is a representation of the CCTs that realize T^α . The nondeterminism in the while loop of AnalyzeCounterex can be efficiently resolved by focusing each node after focusing all of its children. Since T^α is a finite tree, in this bottom-up way, each node is focused exactly once. Indeed, for finite-state game structures and nonsymbolic representations of ACTs, where all node annotations are stored as lists of concrete states, algorithm AnalyzeCounterex can be implemented in linear time.

Proposition 2 [Counterexample checking] *An ACT T^α for a safety game is spurious iff the procedure $\text{AnalyzeCounterex}(T^\alpha)$ returns SPURIOUS. Checking if an ACT for a safety game is spurious can be done in time linear in the size of the tree.*

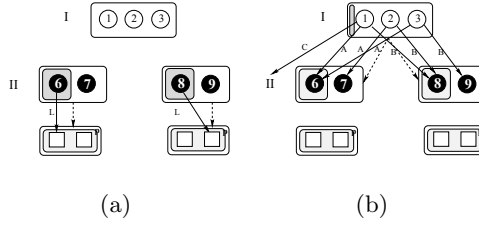


Fig. 2. Focusing T^α

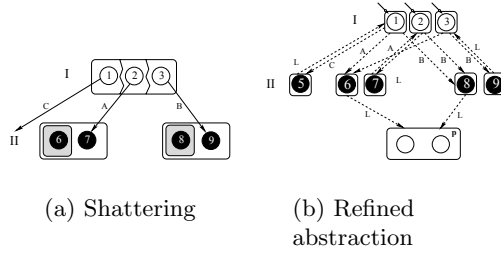


Fig. 3. Abstraction refinement

Example 4 [EXSAFETY] Figure 2 shows the result of running *AnalyzeCounterex* on the ACT T^α from Figure 1(c). The shaded parts of the boxes denote the states that may be a part of a CCT. The dashed arrows indicate abstract transitions, and the solid arrows concrete transitions. Figure 2(a) shows the result of focusing the player 2 nodes. All states in the leaves are error states, and therefore in the shaded boxes. Only states 6 and 8 can go to the error region from the two abstract player-2 states; hence only they are in the focused regions indicated by shaded boxes. Figure 2(b) shows the result of a subsequent application of *Focus* to the root. No state in the root can play only moves *A* and *B* and subsequently go to states from which player 2 can spoil. Hence none of these states can serve as the root of a CCT whose type is in $\text{Subtypes}(T^\alpha)$. Since the focused region of the root is empty, we conclude that the ACT T^α is spurious. ■

Abstraction refinement. If we find an ACT T^α to be spurious, then we must refine the abstraction $\mathcal{G}^\alpha$ in order to rule out T^α . Consider a state $\mathbf{n}:v^\alpha$ of T^α . Abstraction refinement may split the abstract state v^α into several states $v_1^\alpha, \dots, v_m^\alpha$, with $\llbracket v_k^\alpha \rrbracket = r_k$ for $1 \leq k \leq m$, such that $r_1 \cup \dots \cup r_m = \llbracket v^\alpha \rrbracket$. For this purpose, we define a symbolic *Shatter* operator, which takes, for a node $\mathbf{n}:v^\alpha:r$ of the annotated version of T^α generated by the procedure *AnalyzeCounterex*, the triple $(\mathbf{n}, \llbracket v^\alpha \rrbracket, r)$, and returns the set $\{r_1, \dots, r_m\}$. The set r_1 is the “good” set r (the annotation), from which player 2 does indeed have a spoiling strategy

Algorithm 2 RefineAbstraction($\mathcal{G}^\alpha, T^\alpha$)

Input: an abstraction $\mathcal{G}^\alpha$ and an abstract counterexample tree T^α .

Output: if T^α is spurious, then SPURIOUS and a refined abstraction; otherwise GENUINE.

if AnalyzeCounterex(T^α) = (SPURIOUS, S^α) **then**

$R := \{\llbracket v^\alpha \rrbracket \mid v^\alpha \in V^\alpha\}$

for each annotated node $\mathbf{n}:v^\alpha:r$ of S^α **do** $R := R \cup \text{Shatter}(\mathbf{n}, \llbracket v^\alpha \rrbracket, r)$

return (SPURIOUS, $\text{Abstraction}(R)$)

else return GENUINE

of a type in $\text{Subtypes}(T^\alpha)$. The sets $r_2, \dots, r_m$ are “bad” subsets of $\llbracket v^\alpha \rrbracket \setminus r$, from which no such spoiling strategy exists. Each “bad” set r_k , for $2 \leq k \leq m$, is small enough that there is a simple single reason for the absence of a spoiling strategy. For player-1 nodes $\mathbf{n}$, a set r_k may be “bad” because every state $v \in r_k$ either (i) has a move available which is not in $C(\mathbf{n})$, or (ii) has a move l available such that none of the l -successors of v is in a “good” set, from which player 2 can spoil. For player-2 nodes $\mathbf{n}$, there is a single “bad” set, which contains the states that have no successor in a “good” set. Formally, the operator $\text{Shatter}(\mathbf{n}, q, r)$ is defined to take a node $\mathbf{n}$ of the ACT T^α , and two sets $q, r \subseteq V$ of concrete states such that $r \subseteq q$, and it returns a collection $R \subseteq 2^V$ of state sets $r_k \subseteq q$. For each move $l \in C(\mathbf{n})$, let $\{\mathbf{n}_{l,j}:v_{l,j}^\alpha:r_{l,j}\}$ again be the set of l -children of $\mathbf{n}$. Then:

$$\text{Shatter}(\mathbf{n}, q, r) = \begin{cases} \{r\} \cup \{(q \setminus r) \cap \text{Avl}(l) \mid l \notin C(\mathbf{n})\} \\ \quad \cup \{(q \setminus r) \cap \overline{\text{Epre}(\cup_j r_{l,j}, l)} \mid l \in C(\mathbf{n})\} & \text{if } \mathbf{n} \text{ is a player-1 node} \\ \{r, (q \setminus r)\} & \text{if } \mathbf{n} \text{ is a player-2 node} \end{cases}$$

Note that $\bigcup \text{Shatter}(\mathbf{n}, q, \text{Focus}(\mathbf{n}:v^\alpha:q)) = q$. The refinement of the given abstraction $\mathcal{G}^\alpha$ is achieved by the procedure **RefineAbstraction** (Algorithm 2). Given a collection $R \subseteq 2^V$ of state sets, define the equivalence relation $\equiv_R \subseteq V \times V$ by $v_1 \equiv_R v_2$ if for all sets $r \in R$, we have $v_1 \in r$ precisely when $v_2 \in r$. Let $\text{Closure}(R)$ denote the equivalence classes of $\equiv_R$. Given $V \subseteq \bigcup R$, the set $\text{Closure}(R) \subseteq 2^V$ of sets of concrete states uniquely specifies an abstraction for $\mathcal{G}$, denoted $\text{Abstraction}(R)$, which contains for each set $r \in \text{Closure}(R)$ an abstract state w_r^α with $\llbracket w_r^\alpha \rrbracket = r$ (from this the other components of the abstraction are determined). In particular, let $R_1 = \bigcup_{(\mathbf{n}:v^\alpha) \in T^\alpha} \text{Shatter}(\mathbf{n}, \llbracket v^\alpha \rrbracket, \text{Focus}^*(\mathbf{n}))$ and $R_2 = \{\llbracket v^\alpha \rrbracket \mid v^\alpha \in V^\alpha\}$. Our refined abstraction is $\text{Abstraction}(R_1 \cup R_2)$. The new abstraction returned by the procedure **RefineAbstraction**($\mathcal{G}^\alpha, T^\alpha$) rules out ACTs that are similar to the spurious ACT T^α . Given two ACTs T^α and S^α , we say that T^α *subsumes* S^α if $\text{type}(S^\alpha) \in \text{Subtypes}(T^\alpha)$ and for each node $\mathbf{n}:w^\alpha$ of S^α and corresponding node $\mathbf{n}:v^\alpha$ of T^α , we have $\llbracket w^\alpha \rrbracket \subseteq \llbracket v^\alpha \rrbracket$.

Proposition 3 [Abstraction refinement] *If T^α is a spurious ACT for the abstraction $\mathcal{G}^\alpha$ of a safety game, then the abstraction returned by the procedure **RefineAbstraction**($\mathcal{G}^\alpha, T^\alpha$) has no ACT that is subsumed by T^α .*

Algorithm 3 CxSafetyControl($\mathcal{G}, \square \overline{err}$)**Input:** a game structure $\mathcal{G}$ and a safety objective $\square \overline{err}$.**Output:** either CONTROLLABLE and a player-1 winning strategy,
or UNCONTROLLABLE and a player-2 spoiling strategy represented as ACT. $\mathcal{G}^\alpha := \text{InitialAbstraction}(\mathcal{G}, \square \overline{err})$ **repeat** $(winner, T^\alpha) := \text{ModelCheck}(\mathcal{G}^\alpha, \square \overline{err})$ **if** $winner = 2$ **and** $\text{RefineAbstraction}(\mathcal{G}^\alpha, T^\alpha) = (\text{SPURIOUS}, \mathcal{H}^\alpha)$ **then** $\mathcal{G}^\alpha := \mathcal{H}^\alpha$; $winner := \perp$ **until** $winner \neq \perp$ **if** $winner = 1$ **then return** (CONTROLLABLE, T^α)**return** (UNCONTROLLABLE, T^α)

Example 5 [EXSAFETY] Figure 3 shows the effect of the *Shatter* operator on the root of the ACT T^α from Figure 1(c), and the resulting refined abstract game for which T^α is no longer an ACT. For all nonroot nodes, shattering is trivial, namely, into the focused region and its complement. We break up the states in the root into (i) state 1, which can play the move C not available to the abstract state, (ii) state 2, which can proceed by move A to a state from which the abstract player-2 spoiling strategy fails (i.e., a state not inside a shaded box), and (iii) state 3, which can proceed by move B to a state from which the abstract player-2 spoiling strategy fails. ■

4 Counterexample-Guided Controller Synthesis

Safety control. Given a game structure $\mathcal{G}$ and a safety objective $\square \overline{err}$, we wish to determine if player 1 wins, and if so, construct a winning strategy (“synthesize a controller”). Our algorithm, which generalizes the “abstract-verify-refine” loop of [5,6,9], proceeds as follows:

Step 1 (“abstraction”) We first construct an initial abstract game $(\mathcal{G}^\alpha, \square \overline{err})$.

This could be the trivial abstraction induced by the two propositions *init* and *err*, which has at most 8 abstract states (at most 4 for each player, depending on which of the two propositions are true).

Step 2 (“model checking”) We symbolically model check the abstract game to find if player 1 can win, by iterating the $Cpre_1^\alpha$ operator. If so, then the model checker provides a winning player-1 strategy for the abstract game, from which a winning player-1 strategy in the concrete game can be constructed [13]. If not, then the model checker symbolically produces an ACT [11]. As the abstract state space is finite, the model checking is guaranteed to terminate.

Step 3 (“counterexample-guided abstraction refinement”) If model checking returns an ACT T^α , then we use the procedure $\text{AnalyzeCounterex}(T^\alpha)$ to check if the ACT is genuine. If so, then player 2 has a spoiling strategy in the concrete game, and the system is not controllable. If the ACT is spurious, then

we use the procedure $\text{RefineAbstraction}(\mathcal{G}^\alpha, T^\alpha)$ to refine the abstraction $\mathcal{G}^\alpha$, so that T^α (and similar counterexamples) cannot arise on subsequent invocations of the model checker. This step uses the operators *Focus* and *Shatter*, which are defined in terms of *Epre* and can therefore be implemented symbolically. Since T^α is a finite tree, also this step is guaranteed to terminate.

Goto step 2. The process is iterated until we find either a player-1 winning strategy in step 2, or a genuine counterexample in step 3.

The procedure is summarized in Algorithm 3. The function $\text{InitialAbstraction}(\mathcal{G}, \square \overline{err})$ returns a trivial abstraction for $\mathcal{G}$, which preserves *init* and *err*. The function $\text{ModelCheck}(\mathcal{G}^\alpha, \square \overline{err})$ returns a pair $(1, T^\alpha)$ if player 1 can win the abstract game, where T^α is a (memoryless) winning strategy for player 1, and otherwise it returns $(2, T^\alpha)$, where T^α is an ACT. From the soundness of abstraction, we get the soundness of the algorithm.

Proposition 4 [Partial correctness of CxSafetyControl] *If the procedure $\text{CxSafetyControl}(\mathcal{G}, \square \overline{err})$ returns CONTROLLABLE, then player 1 wins the safety game $(\mathcal{G}, \square \overline{err})$. If the procedure returns UNCONTROLLABLE, then player 1 does not win the game.*

In general, the procedure CxSafetyControl may not terminate for infinite-state games (it does terminate for finite-state games). However, one can prove sufficient conditions for termination provided certain state equivalences on the game structure have finite index [13]. For example, for timed games [3,20], where in the course of the procedure CxSafetyControl , the abstract state space always consists of blocks of clock regions, termination is guaranteed. Verification is the special case of control where all states are player-2 states. Hence our algorithm works also for verification, which is illustrated by the following example.

Example 6 [Safety verification] Consider the transition system EXVERIF shown in Figure 4(a). All states are player-2 states. The initial states are 1 and 2, and we wish to check the safety property $\square \bar{p}$, that the states 5 and 6 are not visited by any run. It is easy to see that the system satisfies this property. Figure 4(b) shows an abstraction for EXVERIF. This is a standard existential abstraction for transition systems. In verification, counterexamples are traces (trees without branches). Figure 4(c) shows a trace τ^α , which is an ACT for the abstraction (b). Figure 5 shows the result of running the algorithm AnalyzeCounterex on τ^α . Figure 5(a) shows the effect of applying *Focus* to the second abstract state in τ^α . All concrete states in the third abstract state are error states; hence they are all shaded. Only state 4 can go to one of the error states; hence it is the only state in the focused region of the second abstract state. Figure 5(b) shows the second application of *Focus*, to the root of the trace. As neither 1 nor 2 have 4 as a successor, the focused region of the root is empty. This implies that the counterexample is spurious. Figure 6(a) shows the effect of *Shatter* on the abstract trace τ^α . Since the shaded box of the second abstract state is $\{4\}$, this abstract state gets shattered into $\{3\}$ and $\{4\}$. No other abstract state is shattered. Figure 6(b) shows the refined abstraction, which is free of counterexamples. ■

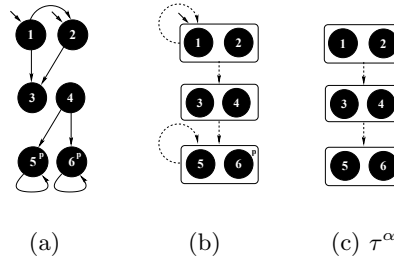


Fig. 4. Example EXVERIF

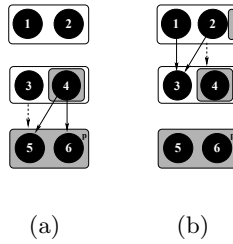


Fig. 5. Focusing τ^α

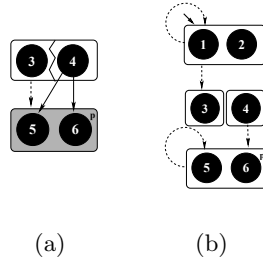


Fig. 6. Refinement

Omega-regular objectives. Counterexample-guided abstraction refinement can be generalized to games with arbitrary ω -regular objectives. To begin with, we must implement a symbolic model checker for solving ω -regular games: given a finite-state game structure $\mathcal{G}^\alpha$ and an ω -regular objective Γ , one can construct a fixpoint formula over the $Cpre_1^\alpha$ operator which characterizes the set of states from which player 1 can win [13]. Moreover, from the fixpoint computation, one

Algorithm 4 CombinedAnalyzeRefine($\mathcal{G}^\alpha, K^\alpha$)

Input: an abstraction $\mathcal{G}^\alpha$, and an abstract counterexample graph K^α with root $\mathbf{n}_0$.
Output: if K^α is spurious, then SPURIOUS and a refined abstraction; otherwise GENUINE.
for each node $\mathbf{n}:v^\alpha$ of K^α **do** annotate $\mathbf{n}:v^\alpha$ by $\llbracket v^\alpha \rrbracket$
 $R := \{\llbracket v^\alpha \rrbracket \mid v^\alpha \in V^\alpha\}$
while there is some node $\mathbf{n}:v^\alpha:r$ with $r \neq \text{Focus}(\mathbf{n}:v^\alpha:r)$ **do**
 $r' := \text{Focus}(\mathbf{n}:v^\alpha:r)$
 $R := R \cup \text{Shatter}(\mathbf{n}, r, r')$
 replace the annotation r of $\mathbf{n}:v^\alpha:r$ by r'
 if $r_0 = \emptyset$ for the annotated root $\mathbf{n}_0: \_ : r_0$ **then return** (SPURIOUS, $\text{Abstraction}(R)$)
end while
return GENUINE

can symbolically construct either a winning strategy for player 1 or a spoiling strategy for player 2 [13,20]. Counterexamples for finite-state ω -regular games are spoiling strategies with finite memory [17], which can be represented as finite graphs. Hence we generalize ACTs from trees to graphs as follows: an *abstract counterexample graph* (ACG) K^α is a rooted, directed, finite graph whose nodes are labeled by abstract states such that conditions (1)–(3) from the definition of ACT hold, and (4) if a leaf (a node with outdegree 0) is labeled by v^α , then $v^\alpha \in V_1^\alpha$ and $L^\alpha(v^\alpha) = \emptyset$. The definition of concrete counterexamples and of the operator *Subtypes* are generalized from trees to graphs in a similar, straightforward way, giving rise to the notion of whether an ACG is *genuine* or *spurious*. So suppose that the function $\text{ModelCheck}(\mathcal{G}^\alpha, \Gamma)$ returns a pair $(1, K^\alpha)$ if player 1 can win the abstract game, where K^α is a (finite-memory) winning strategy for player 1, and otherwise returns $(2, K^\alpha)$, where K^α is an ACG. In the latter case we must now check whether or not K^α is spurious, and if so, then refine the abstraction $\mathcal{G}^\alpha$.

While in the safety case, we analyzed counterexamples (Algorithm 1) before we refined the abstraction (Algorithm 2), for general ω -regular objectives, we combine both procedures (Algorithm 4). The algorithm CombinedAnalyzeRefine computes the fixpoint of the *Focus* operator on a given ACG K^α , and simultaneously refines the given abstraction $\mathcal{G}^\alpha$ by shattering an abstract state with each application of *Focus*. In contrast to the case of trees, for general graphs we cannot apply a bottom-up strategy for focusing. Indeed, in the presence of cycles, the computation of Focus^* may require focusing a node several times before a fixpoint is reached, and CombinedAnalyzeRefine is not guaranteed to terminate (it does terminate for finite-state games). It is easy to see that the procedures AnalyzeCounterex and RefineAbstraction are a special case of CombinedAnalyzeRefine for the case that each node needs to be focused only once. In this case, all shattering can be delayed until focusing is complete, and thus repeated shattering while refocusing the same abstract state can be avoided. Suppose that the procedure CxControl is obtained from CxSafetyControl (Algorithm 3) by replacing the safety objective $\Box \overline{err}$ with an arbitrary ω -

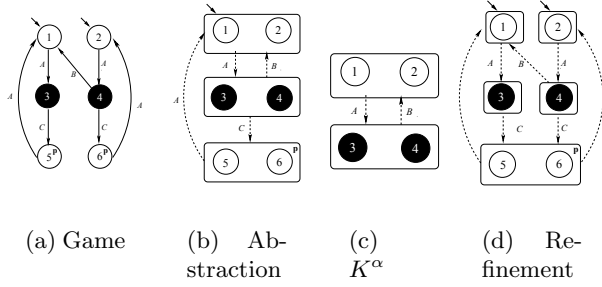
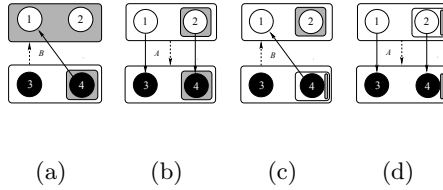


Fig. 7. Example ExBüchi

Fig. 8. CombinedAnalyzeRefine on K^α

regular objective Γ , and by calling the function `CombinedAnalyzeRefine` in place of `RefineAbstraction`. Then we have the following result.

Theorem 1. [Partial correctness of `CxControl`] *Let $\mathcal{G}$ be a game structure, and let Γ be an ω -regular objective. If the procedure `CxControl`($\mathcal{G}, \Gamma$) returns `CONTROLLABLE`, then player 1 wins the game; if the procedure returns `UNCONTROLLABLE`, then player 1 does not win.*

Example 7 [Büchi game] Figure 7(a) shows an example of a Büchi game. We wish to check if player 1 can force the game into a p -state infinitely often, i.e., the objective is $\Box\Diamond p$. Figure 7(b) shows an abstraction for the game. Figure 7(c) shows the result of solving the abstract game, namely, an ACG K^α that has player 2 force a loop not containing a p -state. Figure 8 shows how the ACG is analyzed and discovered to be spurious. Figure 8(a) shows the effect of *Focus* on the lower node of K^α . As only the state 4 has a move into $\{1, 2\}$, the shaded box for the lower node is $\{4\}$. Consequently, the abstract state $\{3, 4\}$ is shattered into $\{3\}$ and $\{4\}$. Figure 8(b) shows the effect of *Focus* on the upper node of K^α . Only state 2 has an A -successor in the shaded box of the lower node; hence the focused region for the upper node becomes $\{2\}$, and the upper node gets shattered into $\{1\}$ and $\{2\}$. In Figure 8(c) we again apply *Focus* to the lower node. Since no state has a B -move to the focused region of the upper node, the focused region of the lower node becomes empty. Figure 8(d) illustrates that

after another *Focus* on the upper node, its focused region becomes empty as well. Figure 7(d) shows the resulting refined abstraction; it is easy to see that player 2 has no spoiling strategy. ■

In [10], the authors consider counterexample-guided abstraction refinement for model checking universal CTL formulas. In this case (and for some more expressive logics considered in [10]), counterexamples are tree-like, and our algorithms for analyzing counterexamples and refining abstractions apply also (indeed, since in this case counterexamples are models of existential CTL formulas, abstract counterexample trees contain only player-2 nodes). More generally, the model-checking problem for the μ -calculus can be reduced to the problem of solving parity games [16]. Via this reduction, our method provides also a counterexample-guided abstraction refinement procedure for model checking the μ -calculus.

References

1. K. Altisen, G. Gössler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *RTSS: Real-Time Systems Symposium*, pages 154–163. IEEE, 1999.
2. R. Alur, L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. Automating modular verification. In *CONCUR: Concurrency Theory*, LNCS 1664, pages 82–97. Springer, 1999.
3. R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In *CONCUR: Concurrency Theory*, pages 74–88. LNCS 1243, Springer, 2001.
4. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.
5. R. Alur, A. Itai, R.P. Kurshan, and M. Yannakakis. Timing verification by successive approximation. *Information and Computation*, 118:142–157, 1995.
6. T. Ball and S.K. Rajamani. The SLAM project: debugging system software via static analysis. In *POPL: Principles of Programming Languages*, pages 1–3. ACM, 2002.
7. J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138:295–311, 1969.
8. A. Church. Logic, arithmetic, and automata. In *International Congress of Mathematicians*, pages 23–35. Institut Mittag-Leffler, 1962.
9. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV: Computer-Aided Verification*, LNCS 1855, pages 154–169. Springer, 2000.
10. E.M. Clarke, S. Jha, Y. Lu, and H. Veith. Tree-like counterexamples in model checking. In *LICS: Logic in Computer Science*, pages 19–29. IEEE, 2002.
11. E.M. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *DAC: Design Automation Conference*, pages 427–432. ACM/IEEE, 1995.
12. L. de Alfaro and T.A. Henzinger. Interface automata. In *FSE: Foundations of Software Engineering*, pages 109–120. ACM, 2001.
13. L. de Alfaro, T.A. Henzinger, and R. Majumdar. Symbolic algorithms for infinite-state games. In *CONCUR: Concurrency Theory*, pages 536–550. LNCS 2154, Springer, 2001.

14. L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. Detecting errors before reaching them. In *CAV: Computer-Aided Verification*, LNCS 1855, pages 186–201. Springer, 2000.
15. D.L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits*. MIT Press, 1989.
16. E.A. Emerson, C.S. Jutla, and A.P. Sistla. On model checking fragments of μ -calculus. In *CAV: Computer-Aided Verification*, LNCS 697, pages 385–396. Springer, 1993.
17. Y. Gurevich and L. Harrington. Trees, automata, and games. In *STOC: Symposium on Theory of Computing*, pages 60–65. ACM, 1982.
18. T.A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *POPL: Principles of Programming Languages*, pages 58–70. ACM, 2002.
19. T.A. Henzinger, R. Majumdar, F.Y.C. Mang, and J.-F. Raskin. Abstract interpretation of game properties. In *SAS: Static-Analysis Symposium*, pages 220–239. LNCS 1824, Springer, 2000.
20. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS: Theoretical Aspects of Computer Science*, LNCS 900, pages 229–242. Springer, 1995.
21. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL: Principles of Programming Languages*, pages 179–190. ACM, 1989.
22. P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25:206–230, 1987.

Axiomatic Criteria for Quotients and Subobjects for Higher-Order Data Types

Jo Hannay

Department of Software Engineering,
Simula Research Laboratory, Pb. 134, NO-1325 Lysaker, Norway
johannay@simula.no

Abstract. Axiomatic criteria are given for the existence of higher-order maps over subobjects and quotients. These criteria are applied in showing the soundness of a method for proving specification refinement up to observational equivalence. This generalises the method to handle data types with higher-order operations, using standard simulation relations. We also give a direct setoid-based model satisfying the criteria. The setting is the second-order polymorphic lambda calculus and the assumption of relational parametricity.

1 Introduction

As a motivating framework for the results in this paper, we use specification refinement. We address specifications for data types whose operations may be higher order.

A *stepwise specification refinement* process transforms an abstract specification into one or more concrete specifications or program modules. If each step is proven correct, the resulting modules will be correct according to the initial abstract specification. This then describes a software development technique for producing small-scale certified components. Theoretical aspects to this idea have been researched thoroughly in the field of algebraic specification, see *e.g.*, [31,6].

When data types have higher-order operations, taking functions as arguments, several things in the refinement methodology break down. Most well-known perhaps, is the lack of correspondence between observational equivalence and the existence of simulation relations for data types, together with the lack of composability. The view is that standard notions of simulation relation are not adequate, and several remedies have been proposed; *pre-logical relations* [18,17], *lax logical relations* [28,20], *L-relations* [19], and *abstraction barrier-observing simulation relations* [11,12,13]. The latter, developed for System F in a logic [27] asserting *relational parametricity* [30], are directly motivated by the information-hiding mechanism in data types. Relational parametricity is in this context the logical assertion of the Basic Lemma [25,18] for simulation relations.

In this paper, we address a further issue. A general proof strategy for proving specification refinement up to observational equivalence is formalised in [4, 3]. For data types with first-order operations, the strategy is expressed in the

setting of System F and relational parametricity by axiomatising the existence of subobjects and quotients [29,36,9,12]. The axioms are sound w.r.t. the parametric *per* model of [1] which is a model for the logic in [27]. At higher order, more work is required, because in order to validate the axioms, one has to find a model which has higher-order operations over subobjects and quotients. Our solution is the core technical issue of this paper. First, we use a setoid-based semantics based on work on the syntactic level in [16]. Then we present general axiomatic criteria for the existence of higher-order functions over subobjects and quotients, and the setoid model is then an instance of this general schema. We think the axiomatic criteria are of general interest outside refinement issues. The results also answer the speculation in [36] about the soundness of similar axioms postulating quotients and subobjects at higher order.

Since simulation relations express observational equivalence, they play an integral part in the above proof strategy. At higher order, it is still possible to use standard simulation relations, because the strategy relies on establishing observational equivalence from the existence of simulation relations. In this paper, we exploit this fact and devise the axiomatic criteria for standard simulation relations. For the strategy to be complete however, one must utilise one of the above alternative notions of simulation relation, since there may not exist a standard simulation relation even in the presence of observational equivalence. To this end, abstraction barrier-observing (*abo*) simulation relations were used in [10,12], together with *abo*-relational parametricity, and a special *abo*-semantics. That approach does indeed yield higher-order operations over quotients and subobjects, but to devise general axiomatic criteria for the existence of higher-order functions over subobjects and quotients with alternative notions of simulation relations, is ongoing research.

2 Syntax

We review relevant formal aspects. For full accounts, see [2,25,8,27,1].

The second-order lambda-calculus F_2 , or System F, has abstract syntax

$$\begin{aligned} \text{(types)} \quad T &::= X \mid (T \rightarrow T) \mid (\forall X. T) \\ \text{(terms)} \quad t &::= x \mid (\lambda x:T. t) \mid (tt) \mid (\Lambda X. t) \mid (tT) \end{aligned}$$

where X and x range over type and term variables respectively. This provides polymorphic functionals and encodings of self-iterating inductive types [5], *e.g.*, $\text{Nat} \stackrel{\text{def}}{=} \forall X. X \rightarrow (X \rightarrow X) \rightarrow X$, with constructors, destructors and conditionals. Products $U_1 \times \cdots \times U_n$ encode as inductive types.

We use the logic for parametric polymorphism due to [27]; a second-order logic augmented with relation symbols, relation definition, and the axiomatic assertion of relational parametricity. See also [22,34]. Formulae now include relational statements as basic predicates and quantifiables,

$$\phi ::= (t =_A u) \mid t \, R \, u \mid \cdots \mid \forall R \subset A \times B . \phi \mid \exists R \subset A \times B . \phi$$

where R ranges over relation variables. Relation definition is accommodated by the syntax,

$$\Gamma \triangleright (x:A, y:B) . \phi \quad \subset A \times B$$

where ϕ is a formula. For example $\mathbf{eq}_A \stackrel{\text{def}}{=} (x:A, y:A).(x =_A y)$.

We write $\alpha[\xi]$ to indicate possible occurrences of variable ξ in type, term or formula α , and write $\alpha[\beta/\xi]$ for the substitution $\alpha[\beta/\xi]$, following the appropriate rules regarding capture.

We get the *arrow-type relation* $\rho \rightarrow \rho' \subset (A \rightarrow A') \times (B \rightarrow B')$ from $\rho \subset A \times B$ and $\rho' \subset A' \times B'$ by

$$(\rho \rightarrow \rho') \stackrel{\text{def}}{=} (f:A \rightarrow A', g:B \rightarrow B') . (\forall x:A. \forall y:B . (x \rho y \Rightarrow (fx) \rho' (gy)))$$

The *universal-type relation* $\forall(Y, Z, R \subset Y \times Z) \rho[R] \subset (\forall Y. A[Y]) \times (\forall Z. B[Z])$ is defined from $\rho[R] \subset A[Y] \times B[Z]$, where Y, Z and $R \subset Y \times Z$ are free, by

$$\forall(Y, Z, R \subset Y \times Z) \rho[R] \stackrel{\text{def}}{=} (y: \forall Y. A[Y], z: \forall Z. B[Z]) . (\forall Y. \forall Z. \forall R \subset Y \times Z . ((yY) \rho[R] (zZ)))$$

For n -ary $\mathbf{X}, \mathbf{A}, \mathbf{B}, \boldsymbol{\rho}$, where $\rho_i \subset A_i \times B_i$, we get $T[\boldsymbol{\rho}] \subset T[\mathbf{A}] \times T[\mathbf{B}]$, the *action of $T[\mathbf{X}]$ on $\boldsymbol{\rho}$* , by

$$\begin{aligned} T[\mathbf{X}] = X_i : & \quad T[\boldsymbol{\rho}] = \rho_i \\ T[\mathbf{X}] = T'[\mathbf{X}] \rightarrow T''[\mathbf{X}] : & \quad T[\boldsymbol{\rho}] = T'[\boldsymbol{\rho}] \rightarrow T''[\boldsymbol{\rho}] \\ T[\mathbf{X}] = \forall X'. T'[\mathbf{X}, X'] : & \quad T[\boldsymbol{\rho}] = \forall(Y, Z, R \subset Y \times Z) T'[\boldsymbol{\rho}, R] \end{aligned}$$

The proof system is intuitionistic natural deduction, augmented with inference rules for relation symbols in the obvious way. There are standard axioms for equational reasoning implying extensionality for arrow and universal types.

Parametric polymorphism prompts all instances of a polymorphic functional to exhibit a uniform behaviour [33,1,30]. We adopt *relational parametricity* [30, 21]; a polymorphic functional instantiated at two related domains, should give related instances. This is asserted by the schema

$$\text{PARAM} : \forall \mathbf{Z}. \forall u. (\forall X. U[X, \mathbf{Z}]) . u \ (\forall X. U[X, \mathbf{eq}_{\mathbf{Z}}]) \ u$$

The logic with PARAM is sound; we have the parametric *per*-model of [1] and the syntactic models of [14]. Relational parametricity yields the fundamental *Identity Extension Lemma*:

$$\forall \mathbf{Z}. \forall u, v. T[\mathbf{Z}] . (u \ T[\mathbf{eq}_{\mathbf{Z}}] \ v \Leftrightarrow (u =_{T[\mathbf{Z}]} v))$$

Constructs such as products, sums, initial and final (co-)algebras are encodable in System F [5]. With PARAM, these become provably universal constructions.

3 Specification Refinement

A specification determines a collection of data types realising the specification. A signature provides the desired namespace, and a set of formulae give properties to be fulfilled. Depending on refinement stage, these range from abstract to

concrete implementational. A data type consists of a data representation and operations. In the logic, these are respectively a type A , and a term $\mathbf{a} : T[A]$, where $T[X]$ plays the role of a signature. For instance, using a labeled product notation, $T_{\text{STACK}_{\text{Nat}}}[X] \stackrel{\text{def}}{=} (\text{empty} : X, \text{push} : \text{Nat} \rightarrow X \rightarrow X, \text{pop} : X \rightarrow X, \text{top} : X \rightarrow \text{Nat})$. Each $f_i : T_i[X]$ is a *profile* of the signature. Abstract properties are *e.g.*, $\forall x : \text{Nat}, s : X . \mathbf{r}.\text{pop}(\mathbf{r}.\text{push } x \ s) = s \wedge \mathbf{r}.\text{top}(\mathbf{r}.\text{push } x \ s) = x$. A data type realising this stack specification, consists *e.g.*, of inductive type List_{Nat} and $\mathbf{l}$, where $\mathbf{l}.\text{empty} = \text{nil}$, $\mathbf{l}.\text{push} = \text{cons}$, $\mathbf{l}.\text{pop} = \lambda l : \text{List}_{\text{Nat}}. (\text{cond } \text{List}_{\text{Nat}} (\text{isnil } l) \text{ nil } (\text{cdr } l))$, and $\mathbf{l}.\text{top} = \lambda l : \text{List}_{\text{Nat}}. (\text{cond } \text{Nat} (\text{isnil } l) 0 (\text{car } l))$. For encapsulation, data types would be given as packages of existential type, but our technical results are on the component level, so we omit this.

To each refinement stage, a set Obs of *observable types* is associated, containing inductive types, and also parameters. Two data types are interchangeable if it makes no difference which one is used in an observable computation. For example, an observable computation on natural-number stacks could be $\Lambda X. \lambda \mathbf{r} : T_{\text{STACK}_{\text{Nat}}}[X] . \mathbf{r}.\text{top}(\mathbf{r}.\text{push } n \ \mathbf{r}.\text{empty})$. Thus, for $A, B, \mathbf{a} : T[A], \mathbf{b} : T[B], \text{Obs}$,

Observational Equivalence: $\bigwedge_{D \in \text{Obs}} \forall f : \forall X. (T[X] \rightarrow D) . (fA \ \mathbf{a}) = (fB \ \mathbf{b})$

Observational equivalence can be hard to prove. A more manageable criterion for interchangeability lies in the concept of *data refinement* [15,7] and the use of relations to show *representation independence* [23,32,30], leading to *logical relations* for lambda calculus [24,25,35,26]. In the relational logic of [27] one uses the action of types on relations to express the above ideas. Two data types are related by a simulation relation if there exists a relation R on their respective data representations that is preserved by their corresponding operations:

Existence of Simulation Relation: $\exists R \subset A \times B . \mathbf{a} (T[R]) \mathbf{b}$

With relational parametricity we get a connection to observational equivalence.

Theorem 1. *The following is derivable in the logic using PARAM.*

$$\begin{aligned} & \forall A, B. \forall \mathbf{a} : T[A], \mathbf{b} : T[B] . \exists R \subset A \times B . \mathbf{a} (T[R]) \mathbf{b} \\ & \Rightarrow \bigwedge_{D \in \text{Obs}} \forall f : \forall X. (T[X] \rightarrow D) . (fA \ \mathbf{a}) = (fB \ \mathbf{b}) \end{aligned}$$

Proof: This follows from the PARAM-instance $\forall Y. \forall f : \forall X. (T[X] \rightarrow Y) . f(\forall X. T[X] \rightarrow \text{eq}_Y) f$. $\square$

Consider the assumption that $T[X]$ has only first-order function profiles:

FADT_{Obs}: Every profile $T_i[X] = T_{i1}[X] \rightarrow \dots \rightarrow T_{ni}[X] \rightarrow T_{ci}[X]$ of $T[X]$ is first order, and such that $T_{ci}[X]$ is either X or some $D \in \text{Obs}$.

Assuming **FADT_{Obs}** for $T[X]$, Theorem 1 becomes a two-way implication [11,12].

For data types with higher-order operations, we only have Theorem 1 in general. More apt relational notions for explaining interchangeability of data

types have been found; prelogical relations [18,17], lax logical relations and L -relations [28,20,19], and *abo*-simulation relations [11,12,13].

For specification refinement one is interested in establishing observational equivalence. For this it suffices to find a simulation relation and then use Theorem 1. The problem at higher order is that there might not exist a simulation relation, even in the presence of observational equivalence.

Nonetheless, it is in many cases possible to find simulation relations at higher order. It is worthwhile to utilise this, since it is harder to deal with the alternative notions in practice; prelogical relations involve an infinite family of relations, *abo*-relations involve definability. Therefore, this paper establishes a proof strategy for refinement at higher order using standard simulation relations.

The strategy for proving observational refinement formalised by Bidoit *et al* [4,3], expresses observational abstraction in terms of a congruence. Using this congruence, one quotients over the data representation. Additionally, it may be necessary to restrict the data representation before quotienting, and in that case one also needs to construct subobjects. For example, sets might be implemented using lists for data representation, but the operations may be optimised, and otherwise fail, by assuming sorted lists. Since lists represent the same set up to duplication of elements, the list algebra is quotiented by a partial congruence that equates lists modulo duplicates, and which is defined only on sorted lists. This strategy is implemented in the type-theoretical setting by extending the logic with the following axiom schemata. They are tailored specifically for refinement.

Definition 1 (Existence of Subobjects (SUB) [9]).

$$\begin{aligned} \text{SUB} : \forall X . \forall \mathfrak{x} : T[X] . \forall R \subset X \times X . (\mathfrak{x} \ T[R] \ \mathfrak{x}) \wedge (\mathfrak{x} \ T[P_R] \ \mathfrak{x}) \Rightarrow \\ \exists S . \exists \mathfrak{s} : T[S] . \exists R' \subset S \times S . \exists \text{mono} : S \rightarrow X . \\ \forall s : S . s \ R' s \quad \wedge \\ \forall s, s' : S . s \ R' s' \Leftrightarrow (\text{mono } s) \ R (\text{mono } s') \wedge \\ \mathfrak{x} \ (T[(x : X, s : S) . (x =_X (\text{mono } s))]) \ \mathfrak{s} \end{aligned}$$

where $P_R \stackrel{\text{def}}{=} (x : X, y : X) . (x =_X y \wedge x \ R \ y)$. Intuitively, this essentially states that for any data type $\langle X, \mathfrak{x} \rangle$, if R is a relation that is compatible with the signature $T[X]$, then there exists a data type $\langle S, \mathfrak{s} \rangle$, a relation R' , and a monomorphism from $\langle S, \mathfrak{s} \rangle$ to $\langle X, \mathfrak{x} \rangle$, such that R' is total on $\langle S, \mathfrak{s} \rangle$ and a restriction of R via mono , and such that $\langle S, \mathfrak{s} \rangle$ is a subalgebra of $\langle X, \mathfrak{x} \rangle$.

Definition 2 (Existence of Quotients (QUOT) [29]).

$$\begin{aligned} \text{QUOT} : \forall X . \forall \mathfrak{x} : T[X] . \forall R \subset X \times X . (\mathfrak{x} \ T[R] \ \mathfrak{x} \wedge \text{equiv}(R)) \Rightarrow \\ \exists Q . \exists \mathfrak{q} : T[Q] . \exists \text{epi} : X \rightarrow Q . \\ \forall x, y : X . x \ R \ y \Leftrightarrow (\text{epi } x) =_Q (\text{epi } y) \wedge \\ \forall q : Q . \exists x : X . q =_Q (\text{epi } x) \quad \wedge \\ \mathfrak{x} \ (T[(x : X, q : Q) . ((\text{epi } x) =_Q q)]) \ \mathfrak{q} \end{aligned}$$

where $\text{equiv}(R)$ specifies R to be an equivalence relation.

Intuitively, this states that for any data type $\langle X, \mathfrak{r} \rangle$, if R is an equivalence relation on $\langle X, \mathfrak{r} \rangle$, then there exists a data type $\langle Q, \mathfrak{q} \rangle$ and an epimorphism from $\langle X, \mathfrak{r} \rangle$ to $\langle Q, \mathfrak{q} \rangle$, such that $\langle Q, \mathfrak{q} \rangle$ is a quotient algebra of $\langle X, \mathfrak{r} \rangle$.

Theorem 2. *SUB, QUOT hold in the parametric per-model of [1], under $FADT_{Obs}$.*

The proof of this theorem [12] relies on the model's ability to provide subobjects and quotients, and maps over these for any given morphism.

4 Higher-Order Quotient and Subobject Maps

In the *per*-model, first-order maps over subobjects and quotients are constructed from a given map by reusing the realiser. This does not work at higher order, since for functional arguments we have to contravariantly do this in reverse.

Consider *e.g.*, sequences over $\mathbb{N}$, whose encodings in $\mathbb{N}$ we write as the sequences themselves. Consider a function rfl on $\mathbb{N}$ that given a sequence, returns the sequence with the first item repeated. Define the *pers* *List*, *Bag*, and *Set* by

$$\begin{aligned} n \text{ List } m &\Leftrightarrow n \text{ and } m \text{ encode the same list} \\ n \text{ Bag } m &\Leftrightarrow n \text{ and } m \text{ encode the same list, modulo permutation} \\ n \text{ Set } m &\Leftrightarrow n \text{ and } m \text{ encode the same list, modulo permutation and repetition} \end{aligned}$$

Here, rfl is a realiser for a map $f_{rfl}: \text{Set} \rightarrow \text{Set}$, but is not a realiser for any map in $\text{Bag} \rightarrow \text{Bag}$, *i.e.*, we have $rfl (\text{Set} \rightarrow \text{Set}) \text{ } rfl$ but not $rfl (\text{Bag} \rightarrow \text{Bag}) \text{ } rfl$.

In fact, the general problem is that there may not be a suitable function at all, let alone one sharing the same realiser.

In the following we sketch a setoid model based on ideas in [16]. This allows the construction of subobject and quotient maps by reusing realisers, also at higher order. Then we give axiomatic criteria for the construction of subobject and quotient maps at higher order. The setoid model fulfils these criteria.

We will work under the following reasonable assumption.

$HADT_{Obs}$: Every profile $T_i[X] = T_{i1}[X] \rightarrow \cdots \rightarrow T_{ni}[X] \rightarrow T_{ci}[X]$ of signature $T[X]$ is such that $T_{ij}[X]$ has no occurrences of universal types other than those in *Obs*, and $T_{ci}[X]$ is either X or some $D \in Obs$.

4.1 A Setoid Model

Types are now interpreted as setoids, *i.e.*, pairs $\langle \mathcal{A}, \sim_{\mathcal{A}} \rangle$, consisting of a *per* $\mathcal{A}$ and a *per* $\sim_{\mathcal{A}}$ on $\mathcal{A}$, *i.e.*, a saturated *per* on $Dom(\mathcal{A}) \times Dom(\mathcal{A})$, giving the desired equality on the interpreted type. Given setoids $\langle \mathcal{A}, \sim_{\mathcal{A}} \rangle$ and $\langle \mathcal{B}, \sim_{\mathcal{B}} \rangle$, we form a setoid $\langle \mathcal{A}, \sim_{\mathcal{A}} \rangle \rightarrow \langle \mathcal{B}, \sim_{\mathcal{B}} \rangle \stackrel{\text{def}}{=} \langle \mathcal{A} \rightarrow \mathcal{B}, \sim_{\mathcal{A} \rightarrow \mathcal{B}} \rangle$, where $\sim_{\mathcal{A} \rightarrow \mathcal{B}}$ is the saturated relation $\sim_{\mathcal{A}} \rightarrow \sim_{\mathcal{B}} \subseteq Dom(\mathcal{A} \rightarrow \mathcal{B}) \times Dom(\mathcal{A} \rightarrow \mathcal{B})$. Saturation of $\sim$ is the condition $(m \mathcal{A} n \wedge n \sim n' \wedge n' \mathcal{B} m') \Rightarrow m \sim m'$.

A relation $\mathcal{R}$ between setoids $\langle \mathcal{A}, \sim_{\mathcal{A}} \rangle$ and $\langle \mathcal{B}, \sim_{\mathcal{B}} \rangle$ is now given by a saturated relation on $Dom(\sim_{\mathcal{A}}) \times Dom(\sim_{\mathcal{B}})$. Complex relations are defined as one would expect. The setoid definition of subobjects and quotients go as follows.

Definition 3 (Subobject Setoid). Let $\mathcal{P}$ be a predicate on setoid $\langle \mathcal{X}, \sim_{\mathcal{X}} \rangle$, meaning that $\mathcal{P}$ fulfils the unary saturation condition $\mathcal{P}(x) \wedge x \sim_{\mathcal{X}} y \Rightarrow \mathcal{P}(y)$. Define the relation, also denoted $\mathcal{P}$, on $\langle \mathcal{X}, \sim_{\mathcal{X}} \rangle$ by $x \mathcal{P} y \stackrel{\text{def}}{\Leftrightarrow} x \sim_X y \wedge \mathcal{P}(x)$. Then the subobject $\mathcal{R}_{\mathcal{P}}(\langle \mathcal{X}, \sim_{\mathcal{X}} \rangle)$ of $\langle \mathcal{X}, \sim_{\mathcal{X}} \rangle$ restricted on $\mathcal{P}$, is defined by $\langle \mathcal{X}, \mathcal{P} \rangle$.

Definition 4 (Quotient Setoid). Let $\mathcal{R}$ be a equivalence relation on setoid $\langle \mathcal{X}, \sim_{\mathcal{X}} \rangle$. Define the quotient $\langle \mathcal{X}, \sim_{\mathcal{X}} \rangle / \mathcal{R}$ of $\langle \mathcal{X}, \sim_{\mathcal{X}} \rangle$ w.r.t. $\mathcal{R}$ by $\langle \mathcal{X}, \mathcal{R} \rangle$.

Theorem 3. Suppose $T[X]$ adheres to HADT_{Obs} . Then SUB and QUOT hold in the setoid model indicated above.

With setoids we may construct quotient maps from a given map and *vice versa* by reusing realisers, since the original domain inhabitation is preserved by subobjects and quotients. However, Theorem 3 is given as a corollary to a general result of the axiomatic criteria in the next section.

4.2 Axiomatic Criteria for Subobject and Quotient Maps

We now develop a general axiomatic scheme for obtaining subobject and quotient maps. The setoid approach in the previous section is an instance of this scheme.

For quotients, the general problem is that for a given map $f : X/R \rightarrow X/R$, there need not exist a map $g : X \rightarrow X$ such that for all $x : X$, $[g(x)] = f([x])$, i.e., $\text{epi}(g(x)) = f(\text{epi}(x))$, where $\text{epi} : X \rightarrow X/R$ maps an element to its equivalence class. This is the case for the *per*-model. The axiom of choice (AC) gives such a map g , because then epi has an inverse, and the desired g is given by $\lambda x : X. \text{epi}^{-1}(f \text{epi}(x))$. AC does not hold in the *per*-model, nor does it hold in the setoid model of the previous section. In this section, we develop both a weaker condition sufficient to give higher-order quotient maps, and a condition for obtaining higher-order subobject maps.

According to HADT_{Obs} , we consider arrow types over types $U_0, U_1, \dots$, where any U_i is either X or some $D \in Obs$. For this, define families U^i by

$$\begin{aligned} U^0 &= U_0 \\ U^{i+1} &= (U^i) \rightarrow U_{i+1} \end{aligned}$$

For example, $U^2 = ((U_0 \rightarrow U_1) \rightarrow U_2)$.

Quotient Maps. For $U = U^n$, define $Q(U)^i$ for any equivalence relation R ,

$$\begin{aligned} Q(U)^0 &= U_0 \\ Q(U)^1 &= U_0/R \rightarrow U_1 \\ Q(U)^{i+1} &= (Q(U)^{i-1} \rightarrow U_i/R) \rightarrow U_{i+1}, 1 \leq i \leq n-1 \end{aligned}$$

where, $U_i/R = X/R$, if U_i is X , and $U_i/R = D$, if U_i is $D \in Obs$, e.g., $Q(U^2)^2 = ((U_0 \rightarrow U_1/R) \rightarrow U_2)$. In any $Q(U)^i$, quotients U_j/R occur only negatively.

Given $Q(U)^n$, we get derived relations, functions and types by the substitution operators $Q(U)^n[\xi]^+$ and $Q(U)^n[\xi]^-$, according to ξ being a relation, function or type; $Q(U)^n[\xi]^+$ substitutes ξ for positive occurrences of X in $Q(U)^n$,

and $Q(U)^n[\xi]^-$ substitutes ξ for every (negative) occurrence of X/R in $Q(U)^n$. Relational and functional identities are then denoted by their domains.

Thus for $U = U^n$ and the equivalence relation R , we can define the relation

$$R(U)^n \stackrel{\text{def}}{=} Q(U)^n[R]^+$$

In any $R(U)^i$, R occurs positively, and identities U_j/R occur only negatively. The point of all this is that, if R is an equivalence relation on X , then $R(U)^i$ is an equivalence relation on $Q(U)^i$. This means that we may form the quotient $Q(U)^i/R(U)^i$. For example, consider $U = U^1 = X \rightarrow X$. Then $Q(U)^1 = X/R \rightarrow X$ and $R(U)^1 = X/R \rightarrow R$, and $X/R \rightarrow R$ is an equivalence relation on $X/R \rightarrow X$. In contrast, $R \rightarrow X/R$ is not necessarily an equivalence relation on $X \rightarrow X/R$. However, $(R \rightarrow X/R) \rightarrow R$ is an equivalence relation on $(X \rightarrow X/R) \rightarrow X$, that is, $R(U^2)^2$ is an equivalence relation on $Q(U^2)^2$, for $U^2 = (X \rightarrow X) \rightarrow X$.

Now consider the relation $\text{graph}(\text{epi}) \stackrel{\text{def}}{=} (x : X, q : X/R) . ((\text{epi } x) =_{X/R} q)$ where the map $\text{epi} : X \rightarrow X/R$ maps elements to their R -equivalence class. A sufficient condition for obtaining higher-order functions over quotients is now

Quot-Arr: For R an equivalence relation on X , and any given $U = U^n$,

$$Q(U)^n/R(U)^n \cong Q(U)^n[X/R]^+$$

where the isomorphism $\text{iso} : Q(U)^n[X/R]^+ \rightarrow Q(U)^n/R(U)^n$ is such that any f in the equivalence class $\text{iso}(\beta)$ is such that $f(Q(U)^n[\text{graph}(\text{epi})]^+) \beta$.

Note that **Quot-Arr** is not an extension to our logic; we do not have quotient types. Rather, **Quot-Arr** is a condition to check in any relevant model, in which the terminology concerning quotients is well defined. In [16], **Quot-Arr** is expressible in the logic, and **Quot-Arr** is shown strictly weaker than the axiom of choice.

Let us exemplify why **Quot-Arr** suffices. The challenge of this paper is higher-order operations in data types, and then the soundness of QUOT and SUB where $T[X]$ has higher-order operation profiles. To illustrate the use of **Quot-Arr** in semantically validating QUOT, suppose $T[X]$ has a profile $g : (X \rightarrow X) \rightarrow X$ and that $R \subset X \times X$ is an equivalence relation. Consider now any $\mathfrak{x} : T[X]$. Assuming $\mathfrak{x} (T[R]) \mathfrak{x}$, we must produce a $\mathfrak{q} : T[X/R]$, such that $\mathfrak{x} (T[\text{graph}(\text{epi})]) \mathfrak{q}$. For $\mathfrak{x}.g : (X \rightarrow X) \rightarrow X$, this involves finding a $\mathfrak{q}.g : (X/R \rightarrow X/R) \rightarrow X/R$, such that

$$\mathfrak{x}.g ((\text{graph}(\text{epi}) \rightarrow \text{graph}(\text{epi})) \rightarrow \text{graph}(\text{epi})) \mathfrak{q}.g \quad (1)$$

Consider now the following instance of **Quot-Arr**.

$$\text{Quot-Arr}_1 : (X/R \rightarrow X)/(X/R \rightarrow R) \cong (X/R \rightarrow X/R)$$

With $Quot\text{-}Arr_1$ we can construct the following commuting diagram.

$$\begin{array}{ccccccc}
 (X/R \rightarrow X) & \xrightarrow{\text{epi} \rightarrow X} & (X \rightarrow X) & \xrightarrow{\mathfrak{r}.g} & X & \xrightarrow{\text{epi}} & X/R \\
 \downarrow \text{epi}_{X/R \rightarrow X} & & & & & & \\
 (X/R \rightarrow X) / (X/R \rightarrow R) & & & & & & \\
 \uparrow \text{iso} & \searrow \text{lift}(\text{epi} \circ \mathfrak{r}.g \circ (\text{epi} \rightarrow X)) & & & & & \\
 X/R \rightarrow X/R & & & & & &
 \end{array}$$

where $\text{epi} \rightarrow X$ maps any $f: X/R \rightarrow X$ to $\lambda x: X.f(\text{epi } x)$, and iso is so that any f in the equivalence class $\text{iso}(\beta)$ satisfies $f(\text{eq}_{X/R} \rightarrow \text{graph}(\text{epi})) \beta$. The desired $\mathfrak{q}.g: (X/R \rightarrow X/R) \rightarrow X/R$ is given by

$$\text{lift}(\text{epi} \circ \mathfrak{r}.g \circ (\text{epi} \rightarrow X)) \circ \text{iso}$$

Here lift is the operation that lifts any $\gamma: Z \rightarrow Y$ to $\text{lift}(\gamma): Z/\sim \rightarrow Y$, given an equivalence relation $\sim$ on Z , provided that γ satisfies $x \sim y \Rightarrow \gamma x = \gamma y$ for all $x, y: Z$. Then, $\text{lift}(\gamma)$ is the map satisfying $\text{lift}(\gamma) \circ \text{epi} = \gamma$. To be able to lift $\text{epi} \circ \mathfrak{r}.g \circ (\text{epi} \rightarrow X)$ in this way, we must check that $\text{epi} \circ \mathfrak{r}.g \circ (\text{epi} \rightarrow X)$ satisfies $f(\text{eq}_{X/R} \rightarrow R) f' \Rightarrow (\text{epi} \circ \mathfrak{r}.g \circ (\text{epi} \rightarrow X))(f) =_{X/R} (\text{epi} \circ \mathfrak{r}.g \circ (\text{epi} \rightarrow X))(f')$, for all $f, f': (X/R \rightarrow X)$. Assuming $f(\text{eq}_{X/R} \rightarrow R) f'$, we get $(\text{epi} \rightarrow X)(f)(R \rightarrow R)(\text{epi} \rightarrow X)(f')$. Then by $\mathfrak{r} T[R] \mathfrak{r}$, the result follows. This warrants the construction of $\mathfrak{q}.g$.

To show that $\mathfrak{q}.g$ is the desired function, we must check that it satisfies (1). This cannot be read directly from the above diagram; for instance, although $\mathfrak{q}.g$ is constructed essentially in terms of $\mathfrak{r}.g$, it is clear that $\text{epi} \rightarrow X$ maps only to those α in $X \rightarrow X$ that do not discern between input of the same R -equivalence class, and these α might not cover the domain of inputs giving all possible outputs. Intuitively though, this suffices since R -equivalence is really all that matters.

More formally, suppose $\alpha: X \rightarrow X$ and $\beta: X/R \rightarrow X/R$ are such that

$$\alpha(\text{graph}(\text{epi}) \rightarrow \text{graph}(\text{epi})) \beta \quad (2)$$

We want $(\mathfrak{r}.g \alpha) \text{graph}(\text{epi}) (\mathfrak{q}.g \beta)$. First show for any $\alpha: X \rightarrow X$, there exists $f_\alpha: X/R \rightarrow X$, s.t. $(\text{epi} \rightarrow X) f_\alpha (R \rightarrow R) \alpha$ and $\text{iso}(\beta) = \text{epi}_{X/R \rightarrow X}(f_\alpha)$, i.e.,

$$\lambda x: X.f_\alpha(\text{epi } x) (R \rightarrow R) \alpha \quad (3)$$

$$\text{iso}(\beta) = [f_\alpha]_{X/R \rightarrow R} \quad (4)$$

The assumption on iso in $Quot\text{-}Arr_1$ is that any f in the equivalence class $\text{iso}(\beta)$ is such that

$$f(\text{eq}_{X/R} \rightarrow \text{graph}(\text{epi})) \beta \quad (5)$$

so any of these f are candidates for f_α . For such an f we show $a R a' \Rightarrow (\lambda x: X.f(\text{epi } x)a) R \alpha a'$, i.e., $[a] = [a'] \Rightarrow [f[a]] = [\alpha a']$. We have from (5), $[a] = [a'] \Rightarrow [f[a]] = \beta[a']$, and by (2), we have $[a] = [a'] \Rightarrow [\alpha a] = \beta[a']$. Together, this gives the desired property for f , so we have the existence of f_α satisfying (3) and (4). From (2) and (5) we also get

$$\lambda x: X.f_\alpha(\text{epi } x) \text{ (graph(epi)} \rightarrow \text{graph(epi)}) } \beta$$

From the above diagram, and (3) and (4), this gives $(\mathfrak{r}.g(\lambda x: X.f_\alpha(\text{epi } x))) \text{ graph(epi)} (\mathfrak{q}.g \beta)$. By $\mathfrak{r} T[R] \mathfrak{r}$, and since we have $\alpha (R \rightarrow R) (\lambda x: X.f_\alpha(\text{epi } x))$, we thus get $(\mathfrak{r}.g \alpha) \text{ graph(epi)} (\mathfrak{q}.g \beta)$.

The general form of this diagram for any given $U = U^n$ and U_c , is

$$\begin{array}{ccccc} Q(U)^n & \xrightarrow{\text{epi}(U)^n} & U_n & \xrightarrow{\mathfrak{r}.g} & U_c & \xrightarrow{\text{epi}} & U_c/R \\ \downarrow \text{epi}_{Q(U)^n} & & & & & & \\ Q(U)^n/R(U)^n & & & & & & \\ \uparrow \text{iso} & \searrow \text{lift}(\text{epi} \circ \mathfrak{r}.g \circ (\text{epi}(U)^n)) & & & & & \\ U[(U_i/R)/U_i] & & & & & & \end{array}$$

where for a given $U = U^n$, we define the function $\text{epi}(U)^n \stackrel{\text{def}}{=} Q(U)^n[\text{epi}]^-$.

Subobject Maps. A similar story applies to subobjects. For any predicate P on X , we write $R_P(X)$ for the subobject of X classified by those $x: X$ such that $P(x)$ holds. Let the monomorphism $\text{mono}: R_P(X) \rightarrow X$ map elements to their correspondents in X . For use in arrow-type relations, we construct a binary relation from P , also denoted P , by

$$P \stackrel{\text{def}}{=} (x: X, y: X) . (x =_X y \wedge \exists y': R_P(X) . y = (\text{mono } y'))$$

Now for a given $U = U^n$, define $S(U)^i$ for some P as follows

$$\begin{aligned} S(U)^0 &= R_P(U_0) \\ S(U)^1 &= U_0 \rightarrow R_P(U_1) \\ S(U)^{i+1} &= (S(U)^{i-1} \rightarrow U_i) \rightarrow R_P(U_{i+1}), 1 \leq i \leq n-1 \end{aligned}$$

where, if U_i is X then $R_P(U_i) = R_P(X)$, and if U_i is $D \in \text{Obs}$ then $R_P(U_i) = D$. For example $S(U^2)^2 = ((R_P(U_0) \rightarrow U_1) \rightarrow R_P(U_2))$. In any $S(U)^i$, subobjects $R_P(U_j)$ occur only positively. For any $U = U^n$ and some P , define the relation

$$P(U)^n \stackrel{\text{def}}{=} S(U)^n[P]^-$$

The substitution operators $S(U)^n[\xi]^-$ and $S(U)^n[\xi]^+$ are analogues to $Q(U)^n[\xi]^-$ and $Q(U)^n[\xi]^+$. Identities are denoted by their domains.

Intuitively, one would think that for any given $U = U^n$, we should now postulate an isomorphism between $R_{P(U)^n}(S(U)^n)$ and $S(U)^n[(R_P(X))^-]$. This would be in dual analogy to *Quot-Arr*. However, this isomorphism does not exist even in the setoid model. For example, we will not be able to find an isomorphism between $R_{P \rightarrow R_P(X)}(X \rightarrow R_P(X))$ and $R_P(X) \rightarrow R_P(X)$. However, it turns out that we can in fact use an outermost quotient instead of subobjects for the isomorphism, in the same way as we did for *Quot-Arr*.

Thus, if P is a predicate on X , then $P(U)^i$ is an equivalence relation on $S(U)^i$. This means that we may form the quotient $S(U)^i/P(U)^i$, e.g., if $U = U^1 = X \rightarrow X$, then $S(U)^1 = X \rightarrow R_P(X)$ and $P(U)^1 = P \rightarrow R_P(X)$, and $P \rightarrow R_P(X)$ is an equivalence relation on $X \rightarrow R_P(X)$. Again, in contrast, $R_P(X) \rightarrow P$ is not necessarily an equivalence relation on $R_P(X) \rightarrow X$. However, $(R_P(X) \rightarrow P) \rightarrow R_P(X)$ is an equivalence relation on $(R_P(X) \rightarrow X) \rightarrow R_P(X)$, that is, $P(U^2)^2$ is an equivalence relation on $S(U^2)^2$, for $U^2 = (X \rightarrow X) \rightarrow X$.

For the relation $graph(mono) \stackrel{def}{=} (x : X, s : R_P(X)) \cdot (x =_X (mono\ s))$, a sufficient condition for obtaining higher-order functions over subobjects is,

Sub-Arr: For P a predicate on X , and any given $U = U^n$,

$$S(U)^n/P(U)^n \cong S(U)^n[(R_P(X))^-]$$

where the isomorphism $iso : S(U)^n[(R_P(X))^-] \rightarrow S(U)^n/P(U)^n$ is such that any f in the equivalence class $iso(\beta)$ is such that $f(S(U)^n[graph(mono)]^-) \beta$.

Again, **Sub-Arr** is not an axiom in our logic, but is a condition that we can check for models in which the terminology in **Sub-Arr** has a well-defined meaning.

To illustrate **Sub-Arr**, suppose $T[X]$ has a profile $g : (X \rightarrow X) \rightarrow X$. For any $\mathfrak{x} : T[X]$, assume $\mathfrak{x} T[P] \mathfrak{x}$. We must exhibit a $\mathfrak{s} : T[R_P(X)]$, such that $\mathfrak{x} T[graph(mono)] \mathfrak{s}$. For $\mathfrak{x}.g : (X \rightarrow X) \rightarrow X$, this means finding a $\mathfrak{s}.g : (R_P(X) \rightarrow R_P(X)) \rightarrow R_P(X)$, s.t.

$$\mathfrak{x}.g ((graph(mono) \rightarrow graph(mono)) \rightarrow graph(mono)) \mathfrak{s}.g \quad (6)$$

Consider now the following instance of **Sub-Arr**.

Sub-Arr₁: For a predicate P on X ,

$$(X \rightarrow R_P(X)) / (P \rightarrow R_P(X)) \cong R_P(X) \rightarrow R_P(X)$$

Using **Sub-Arr₁**, we can construct the following commuting diagram.

$$\begin{array}{ccccc}
 (X \rightarrow R_P(X)) & \xrightarrow{X \rightarrow mono} & (X \rightarrow X) & \xrightarrow{\mathfrak{x}.g} & X & \xleftarrow{mono} & R_P(X) \\
 \text{epi}_{X \rightarrow R_P(X)} \downarrow & & & & & & \\
 (X \rightarrow R_P(X)) / (P \rightarrow R_P(X)) & & & & & & \\
 \uparrow iso & \searrow lift(\mathfrak{x}.g \circ (X \rightarrow mono)) & & & & & \\
 R_P(X) \rightarrow R_P(X) & & & & & &
 \end{array}$$

Then, $\mathfrak{s}.g : (\mathbf{R}_P(X) \rightarrow \mathbf{R}_P(X)) \rightarrow \mathbf{R}_P(X)$ is given by $\text{lift}(\mathfrak{x}.g \circ (X \rightarrow \text{mono})) \circ \text{iso}$. To justify the lifting of $\mathfrak{x}.g \circ (X \rightarrow \text{mono})$, we must show for all $f, f' : X \rightarrow \mathbf{R}_P(X)$ satisfying $f (P \rightarrow \mathbf{R}_P(X)) f'$, that $\mathfrak{x}.g \circ (X \rightarrow \text{mono})(f) =_X \mathfrak{x}.g \circ (X \rightarrow \text{mono})(f')$. Note that $\text{lift}(\mathfrak{x}.g \circ (X \rightarrow \text{mono}))$ then maps to X , so in addition we must show that $\text{lift}(\mathfrak{x}.g \circ (X \rightarrow \text{mono}))$ in fact maps to $\mathbf{R}_P(X)$. Now, if $f (P \rightarrow \mathbf{R}_P(X)) f'$, we get $(X \rightarrow \text{mono})(f) (P \rightarrow P) (X \rightarrow \text{mono})(f')$. By assumption, we have $\mathfrak{x} T[P] \mathfrak{x}$, in particular $\mathfrak{x}.g ((P \rightarrow P) \rightarrow P) \mathfrak{x}.g$, and the result follows. If for some $y, \exists y' : \mathbf{R}_P(X) . \text{mono } y' = y$, we assume that it is elementary to find such a y' . Thus, since mono is a monomorphism, we may map $\text{lift}(\mathfrak{x}.g \circ (X \rightarrow \text{mono}))$ to $\mathbf{R}_P(X)$, and so we have a function $\mathfrak{s}.g : ((\mathbf{R}_P(X) \rightarrow \mathbf{R}_P(X)) \rightarrow \mathbf{R}_P(X))$.

To show that $\mathfrak{s}.g$ is the desired function, we must check that it satisfies (6). Suppose $\alpha : X \rightarrow X$ and $\beta : \mathbf{R}_P(X) \rightarrow \mathbf{R}_P(X)$ are such that

$$\alpha \text{ (graph(mono)} \rightarrow \text{graph(mono)) } \beta \quad (7)$$

We want $(\mathfrak{x}.g \alpha) \text{ graph(mono)} (\mathfrak{s}.g \beta)$. First show for any $\alpha : X \rightarrow X$, there exists $f_\alpha : X \rightarrow \mathbf{R}_P(X)$, such that $(X \rightarrow \text{mono})f_\alpha (P \rightarrow P) \alpha$ and $\text{iso}(\beta) = \text{epi}_{X \rightarrow \mathbf{R}_P(X)}(f_\alpha)$, i.e.,

$$\lambda x : X. \text{mono}(f_\alpha x) (P \rightarrow P) \alpha \quad (8)$$

$$\text{iso}(\beta) = [f_\alpha]_{P \rightarrow \mathbf{R}_P(X)} \quad (9)$$

The assumption on iso in **Sub-Arr**₁ is that any f in the equivalence class $\text{iso}(\beta)$ is such that

$$f \text{ (graph(mono)} \rightarrow \text{eq}_{\mathbf{R}_P(X)}) \beta \quad (10)$$

so any of these f are candidates for f_α . For such an f , show (8), i.e., $a = a' \wedge \exists a''. \text{mono } a'' = a' \Rightarrow \text{mono}(fa) = \alpha a' \wedge \exists b . \text{mono } b = \alpha a'$. We have from (10), $a = \text{mono } a'' \Rightarrow fa = \beta a''$, and by assumption on α and β , we have $a' = \text{mono } a'' \Rightarrow \alpha a' = \text{mono}(\beta a'')$. This gives the desired property for f , so we have the existence of f_α satisfying (8) and (9). From (10) we also get

$$\lambda x : X. \text{mono}(f_\alpha x) \text{ (graph(mono)} \rightarrow \text{graph(mono)) } \beta$$

From the above diagram, and (8) and (9), this gives $(\mathfrak{x}.g(\lambda x : X. \text{mono}(f_\alpha x))) \text{ graph(mono)} (\mathfrak{s}.g \beta)$. By $\mathfrak{x} T[P] \mathfrak{x}$, and since $\alpha (P \rightarrow P) \lambda x : X. \text{mono}(f_\alpha x)$, we thus get $(\mathfrak{x}.g \alpha) \text{ graph(mono)} (\mathfrak{s}.g \beta)$.

Here is the general form of this diagram for any given $U = U^n$ and U_c , is

$$\begin{array}{ccccc}
 S(U)^n & \xrightarrow{\text{mono}(U)^n} & U_n & \xrightarrow{\mathfrak{x}.g} & U_c \xleftarrow{\text{mono}} \mathbf{R}_P(U_c) \\
 \text{epi}_{S(U)^n} \downarrow & & & & \\
 S(U)^n / P(U)^n & & & & \\
 \uparrow \text{iso} & \searrow \text{lift}(\mathfrak{x}.g \circ (\text{mono}(U)^n)) & & & \\
 U[(\mathbf{R}_P(U_i)) / U_i] & & & &
 \end{array}$$

where for a given $U = U^n$, we define the function $\text{mono}(U)^n \stackrel{\text{def}}{=} S(U)^n[\text{mono}]^+$.

This schema is more general than what is called for in the refinement-specific SUB. In SUB, the starting point is a relation R , and the predicate with which one restricts the domain X , is $P_R(x) \stackrel{\text{def}}{=} x R x$. The corresponding binary relation is then $P_R \stackrel{\text{def}}{=} (x:X, y:X) . (x =_X y \wedge x R x)$.

In closing, we mention that the *per* model, parametric or not, does not satisfy *Quot-Arr* nor *Sub-Arr*. We summarise:

Theorem 4. *Suppose $T[X]$ adheres to HADT_{Obs} . Then SUB and QUOT hold in any model that satisfies *Sub-Arr* and *Quot-Arr*.*

Theorem 5. *The setoid model satisfies *Quot-Arr* and *Sub-Arr*, by the isomorphism being denotational equality.*

Proof: See [12]. □

Corollary 6. *SUB and QUOT hold in the setoid model indicated above.*

5 Final Remarks

We have devised and validated a method in logic for proving specification refinement for data types with higher-order operations. The method is based on standard simulation relations, accommodating the fact that these are easier to deal with than alternative notions when performing refinement. In general however, there may not exist standard simulation relations at higher order, even in the presence of observational equivalence. It is possible to devise specialised solutions to this using abstraction barrier-observing simulation relations [10,12], or pre-logical relations expressed in System F. Beyond that, it is desirable to find general axiomatic criteria analogous to *Sub-Arr* and *Quot-Arr*, using alternative notions of simulation relations. This is currently under investigation.

Acknowledgments. Martin Hofmann has contributed with essential input.

References

1. E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70:35–64, 1990.
2. H.P. Barendregt. Lambda calculi with types. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 118–309. Oxford University Press, 1992.
3. M. Bidoit and R. Hennicker. Behavioural theories and the proof of behavioural properties. *Theoretical Computer Science*, 165:3–55, 1996.
4. M. Bidoit, R. Hennicker, M. Wirsing. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173:393–443, 1997.
5. C. Böhm and A. Berarducci. Automatic synthesis of typed λ -programs on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.

6. M. Cerioli, M. Gogolla, H. Kirchner, B. Krieg-Brückner, Z. Qian, and M. Wolf, eds.. *Algebraic System Specification and Development. Survey and Annotated Bibliography, 2nd Ed., BISS Monographs*, vol. 3. Shaker Verlag, 1997.
7. O.-J. Dahl. *Verifiable Programming, Revised version 1993*. Prentice Hall Int. Series in Computer Science; C.A.R. Hoare, Series Editor. Prentice-Hall, UK, 1992.
8. J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.
9. J. Hannay. Specification refinement with System F. In *Computer Science Logic. Proc. of CSL'99*, vol. 1683 of *Lecture Notes in Comp. Sci.*, pages 530–545. Springer Verlag, 1999.
10. J. Hannay. Specification refinement with System F, the higher-order case. In *Recent Trends in Algebraic Development Techniques. Selected Papers from WADT'99*, volume 1827 of *Lecture Notes in Comp. Sci.*, pages 162–181. Springer Verlag, 1999.
11. J. Hannay. A higher-order simulation relation for System F. In *Foundations of Software Science and Computation Structures. Proc. of FOSSACS 2000*, vol. 1784 of *Lecture Notes in Comp. Sci.*, pages 130–145. Springer Verlag, 2000.
12. J. Hannay. *Abstraction Barriers and Refinement in the Polymorphic Lambda Calculus*. PhD thesis, Laboratory for Foundations of Computer Science (LFCS), University of Edinburgh, 2001.
13. J. Hannay. Abstraction barrier-observing relational parametricity. In *Typed Lambda Calculi and Applications. Proc. of TLCA 2002, Lecture Notes in Comp. Sci.*, Springer Verlag, 2002. To appear.
14. R. Hasegawa. Parametricity of extensionally collapsed term models of polymorphism and their categorical properties. In *Theoretical Aspects of Computer Software. Proc. of TACS'91*, vol. 526 of *Lecture Notes in Comp. Sci.*, pages 495–512. Springer Verlag, 1991.
15. C.A.R. Hoare. Proofs of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.
16. M. Hofmann. *Extensional Concepts in Intensional Type Theory, Report CST-117-95 and Technical Report ECS-LFCS-95-327*. PhD thesis, Laboratory for Foundations of Computer Science (LFCS), University of Edinburgh, 1995.
17. F. Honsell, J. Longley, D. Sannella, and A. Tarlecki. Constructive data refinement in typed lambda calculus. In *Foundations of Software Science and Computation Structures. Proc. of FOSSACS 2000*, vol. 1784 of *Lecture Notes in Comp. Sci.*, pages 161–176. Springer Verlag, 2000.
18. F. Honsell and D. Sannella. Prelogical relations. *Information and Computation*, 178:23–43, 2002.
19. Y. Kinoshita, P.W. O'Hearn, J. Power, M. Takeyama, and R.D. Tennent. An axiomatic approach to binary logical relations with applications to data refinement. In *Theoretical Aspects of Computer Software. Proc. of TACS'97*, vol. 1281 of *Lecture Notes in Comp. Sci.*, pages 191–212. Springer Verlag, 1997.
20. Y. Kinoshita and J. Power. Data refinement for call-by-value programming languages. In *Computer Science Logic. Proc. of CSL'99*, vol. 1683 of *Lecture Notes in Comp. Sci.*, pages 562–576. Springer Verlag, 1999.
21. Q. Ma and J.C. Reynolds. Types, abstraction and parametric polymorphism, part 2. In *Mathematical Foundations of Programming Semantics, Proc. of MFPS*, vol. 598 of *Lecture Notes in Comp. Sci.*, pages 1–40. Springer Verlag, 1991.
22. H. Mairson. Outline of a proof theory of parametricity. In *Functional Programming and Computer Architecture. Proc. of the 5th acm Conf.*, vol. 523 of *Lecture Notes in Comp. Sci.*, pages 313–327. Springer Verlag, 1991.

23. R. Milner. An algebraic definition of simulation between programs. In *Joint Conferences on Artificial Intelligence, Proceedings of the 2nd International Conference, JCAI, London (UK)*, pages 481–489. Morgan Kaufman Publishers, 1971.
24. J.C. Mitchell. On the equivalence of data representations. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 305–330. Academic Press, 1991.
25. J.C. Mitchell. *Foundations for Programming Languages*. Foundations of Computing. MIT Press, 1996.
26. P.W. O’Hearn and R.D. Tennent. Relational parametricity and local variables. In *20th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Proceedings*, pages 171–184. ACM Press, 1993.
27. G.D. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Typed Lambda Calculi and Applications. Proc. of TLCA ’93*, vol. 664 of *Lecture Notes in Comp. Sci.*, pages 361–375. Springer Verlag, 1993.
28. G.D. Plotkin, J. Power, D. Sannella, and R.D. Tennent. Lax logical relations. In *Automata, Languages and Programming. Proc. of ICALP 2000*, vol. 1853 of *Lecture Notes in Comp. Sci.*, pages 85–102. Springer Verlag, 2000.
29. E. Poll and J. Zwanenburg. A logic for abstract data types as existential types. In *Typed Lambda Calculus and Applications. Proc. of TLCA ’99*, vol. 1581 of *Lecture Notes in Comp. Sci.*, pages 310–324. Springer Verlag, 1999.
30. J.C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 83, Proc. of the IFIP 9th World Computer Congress*, pages 513–523. Elsevier Science Publishers B.V. (North-Holland), 1983.
31. D. Sannella and A. Tarlecki. Essential concepts of algebraic specification and program development. *Formal Aspects of Computing*, 9:229–269, 1997.
32. O. Schoett. Behavioural correctness of data representations. *Science of Computer Programming*, 14:43–57, 1990.
33. C. Strachey. Fundamental concepts in programming languages. Lecture notes from the International Summer School in Programming Languages, Copenhagen, 1967.
34. I. Takeuti. An axiomatic system of parametricity. *Fundamenta Informaticae*, 20:1–29, 1998.
35. R.D. Tennent. Correctness of data representations in Algol-like languages. In A.W. Roscoe, editor, *A Classical Mind: Essays in Honour of C.A.R. Hoare*. Prentice Hall International, 1997.
36. J. Zwanenburg. *Object-Oriented Concepts and Proof Rules: Formalization in Type Theory and Implementation in Yarrow*. PhD thesis, Tech. Univ. Eindhoven, 1999.

Efficient Pebbling for List Traversal Synopses

Yossi Matias^{1*} and Ely Porat²

¹ School of Computer Science, Tel Aviv University, matias@cs.tau.ac.il

² Department of Mathematics and Computer Science, Bar-Ilan University, 52900
Ramat-Gan, Israel, (972-3)531-8407, porately@cs.biu.ac.il

Abstract. We show how to support efficient back traversal in a unidirectional list, using small memory and with essentially no slowdown in forward steps. Using $O(\lg n)$ memory for a list of size n , the i 'th back-step from the farthest point reached so far takes $O(\lg i)$ time worst case, while the overhead per forward step is at most epsilon for arbitrary small constant $\epsilon > 0$. An arbitrary sequence of forward and back steps is allowed. A full trade-off between memory usage and time per back-step is presented: k vs. $kn^{1/k}$ and vice versa. Our algorithm is based on a novel pebbling technique which moves pebbles on a “virtual binary tree” that can only be traversed in a pre-order fashion.

The list traversal synopsis extends to general directed graphs, and has other interesting applications, including memory efficient hash-chain implementation. Perhaps the most surprising application is in showing that for any program, arbitrary rollback steps can be efficiently supported with small overhead in memory, and marginal overhead in its ordinary execution. More concretely: Let P be a program that runs for at most T steps, using memory of size M . Then, at the cost of recording the input used by the program, and increasing the memory by a factor of $O(\lg T)$ to $O(M \lg T)$, the program P can be extended to support an arbitrary sequence of forward execution and rollback steps, as follows. The i 'th rollback step takes $O(\lg i)$ time in the worst case, while forward steps take $O(1)$ time in the worst case, and $1 + \epsilon$ amortized time per step.

1 Introduction

A unidirectional list enables easy forward traversal in constant time per step. However, getting from a certain object to its preceding object cannot be done effectively. It requires forward traversal from the beginning of the list and takes time proportional to the distance to the current object, using $O(1)$ additional memory. In order to support more effective back-steps on a unidirectional list, it is required to add auxiliary data structures.

* Research supported in part by the Israel Science Foundation.

The goal of this work is to support memory- and time-efficient back traversal in unidirectional lists, without essentially increasing the time per forward traversal. In particular, under the constraint that forward steps should remain constant, we would like to minimize the number of pointers kept for the lists, the memory used by the algorithm, and the time per back-step, supporting an arbitrary sequence of forward and back steps.

Of particular interest are situations in which the unidirectional list is already given, and we have access to the list but no control over its implementation. The list may represent a data structure implemented in computer memory or in a database, it may reside on a separate computer system. The list may also represent a computational process, where the objects in the list are configurations in the computation and the next pointer represents a computational step.

1.1 Main Results

The main result of this paper is an algorithm that supports efficient back traversal in a unidirectional list, using small memory and with essentially no slowdown in forward steps: $1 + \epsilon$ amortized time per forward step for arbitrary small constant $\epsilon > 0$, and $O(1)$ time in the worst case. Using $O(\lg n)$ memory, back traversals can be supported in $O(\lg n)$ time per back-step, where n is the distance from the beginning of the list to farthest point reached so far. In fact, we show that a back traversal of limited scope can be executed more effectively: $O(\lg i)$ time for the i 'th back-step, for any $i \leq n$, using $O(\lg n)$ memory.

The following trade-offs are obtained: $O(kn^{1/k})$ time per back-step, using k additional pointers, or $O(k)$ time per back-step, using $O(kn^{1/k})$ additional pointers; in both cases supporting $O(1)$ time per forward step (independent of k). Our results extend to general directed graphs, with additional memory of $\lg d_v$ bits for each node v along the backtrack path, where d_v is the outdegree of node v .

The crux of the list traversal algorithm is an efficient pebbling technique which moves pebbles on a virtual binary or k -ary trees that can only be traversed in a pre-order fashion. We introduce the *virtual pre-order tree data structure* which enables managing the pebbles positions in a concise and simple manner.

1.2 Applications

Consider a program P running in time T . Then, using our list pebbling algorithm, the program can be extended to a program P' that supports rollback steps, where a rollback after step i means that the program returns to the configuration it had after step $i - 1$. Arbitrary rollback steps can be added to the execution of the program P' at a cost of increasing the memory requirement by a factor of $O(\lg T)$, and having the i 'th rollback step supported in $O(\lg i)$ time.

The overhead for the forward execution of the program can be kept an arbitrary small constant.

Allowing effective rollback steps may have interesting applications. For instance, a desired functionality for debuggers is to allow pause and rollback during execution. Another implication is the ability to take programs that simulate processes and allow running them backward in arbitrary positions. Thus a program can be run with ϵ overhead and allow pausing at arbitrary points, and running backward an arbitrary number of steps with logarithmic time overhead. The memory required is keeping state configuration of $\lg T$ points, and additional $O(\lg T)$ memory. Often, debuggers and related applications avoid keeping full program states by keeping only differences between the program states. If this is allowed, then a more appropriate representation of the program would be a linked list in which every node represents a sequence of program states, such that the accumulated size of the differences is in the order of a single program state.

Our pebbling technique can be used to support backward computation of a hash-chain in time $O(kn^{1/k})$ using k hash values, or in time $O(k)$ using $O(kn^{1/k})$ hash values. A *hash-chain* is obtained by repeatedly applying a one way-hash function, starting with a secret seed. There are many cryptographic applications, including micro-payment, authentication, and session-key maintenance, which are based on rolling back a hash-chain. Our results enable effective implementation with arbitrary memory size.

The list pebbling algorithm extends to directed trees and general directed graphs. Applications include the effective implementation of the parent function (“..”) for XML trees, and effective graph traversals with applications to “light-weight” Web crawling and garbage collection.

1.3 Related Work

The Schorr-Waite algorithm [9] has numerous applications; see e.g., [10,11,3]. It would be interesting to explore to what extent these applications could benefit from the non-intrusive nature of our algorithm. There is an extensive literature on graph traversal with bounded memory but for other problems than the one addressed in this paper; see, e.g., [5,2]. Pebbling models were extensively used for bounded space upper and lower bounds. See e.g., the seminal paper by Pippenger [8] and more recent papers such as [2].

The closest work to ours is the recent paper by Ben-Amram and Petersen [1]. They present a clever algorithm that, using memory of size $k \leq \lg n$, supports back-step in $O(kn^{1/k})$ time. However, in their algorithm forward steps take $O(k)$ time. Thus, their algorithm supports $O(\lg n)$ time per back-step, using $O(\lg n)$ memory but with $O(\lg n)$ time per forward step, which is unsatisfactory in our

context. Ben-Amram and Petersen also prove a near-matching lower bound, implying that to support back traversal in $O(n^{1/k})$ time per back-step it is required to have $\Omega(k)$ pebbles. Our algorithm supports similar trade-off for back-steps as the Ben-Amram Petersen algorithm, while supporting simultaneously constant time per forward step. In addition, our algorithm extends to support $O(k)$ time per back-step, using memory of size $O(kn^{1/k})$.

Recently, and independently to our work, Jakobsson and Coppersmith [6,4] propose a so-called fractal-hashing technique that enables backtracking hash-chains in $O(\lg n)$ amortized time using $O(\lg n)$ memory. Thus, by keeping $O(\lg n)$ hash values along the hash-chain, their algorithms enables, starting at the end of the chain, to get repeatedly the preceding hash value in $O(\lg n)$ amortized time. Note that our pebbling algorithm enables a full memory-time trade-off for hash-chain execution, and can guarantee that the time per execution is bounded in the worst case.

The most challenging aspect of our algorithm is the proper management of the pointers positions under the restriction that forward steps have very little effect on their movement, to achieve ϵ -overhead per forward step. This is obtained by using the virtual pre-order tree data structure in conjunction with a so-called recycling-bin data structure and other techniques, to manage the positions of the back-pointers in a concise and simple manner.

Due to space limitations, many details are omitted from this extended abstract, and are given in the full paper [7].

2 The Virtual Pre-order Tree Data Structure

In this section we illustrate the basic idea of the list pebbling algorithm, and demonstrate it through a limited functionality of having a sequence of back-steps only. A full algorithm must support an arbitrary sequence of forward and backward steps, and we will also be interested in refinements, such as reducing to minimum the number of pebbles. Adapting the skeleton data structures to support the full algorithm and its refinements may be quite complicated, since controlling and handling the positions of the various pointers becomes a challenge. For the further restriction that forward steps do not incur more than constant overhead (independent of k), the problem becomes even more difficult and we are not aware of any previously known technique to handle this.

To have control over the pointers positioning, we present in Section 2.1 the *virtual pre-order tree* data structure, and show how it supports the sequence of back-steps similarly to the skeleton data structure. In the next sections, we will see how the virtual pre-order tree data structure is used to support the full algorithm as well as more advanced algorithms.

2.1 The Virtual Pre-order Tree Data Structure

The reader is reminded that in a pre-order traversal, the successor of an internal node in the tree is always its left child; the successor of a leaf that is a left child is its right sibling; and the successor of a leaf that is a right child is defined as the right sibling of the nearest ancestor that is a left child. An alternative description is as follows: consider the largest sub-tree of which this leaf is the right-most leaf, and let u be the root of that sub-tree. Then the successor is the right-sibling of u . Consequently, the backward traversal on the tree will be defined as follows. The successor of a node that is a left child is its parent. The successor of a node v that is a right child is the rightmost leaf of the left sub-tree of v 's parent.

The *virtual pre-order tree* data structure consists of (1) an implicit binary tree, whose nodes correspond to the nodes of the linked list, in a pre-order fashion, and (2) an explicit sub-tree of the implicit tree, whose nodes are pebbled. For the basic algorithm, the pebbled sub-tree consists of the path from the root to the current position.

Each pebble represents a pointer; i.e., pebbled nodes can be accessed in constant time. We defer to later sections the issues of how to maintain the pebbles, and how to navigate within the implicit tree, without actually keeping it.

3 The List Pebbling Algorithm

In this section we describe the list pebbling algorithm, which supports an arbitrary sequence of forward and back steps. Each forward step takes $O(1)$ time, where each back-step takes $O(\lg n)$ amortized time, using $O(\lg n)$ pebbles. We will first present the basic algorithm which uses $O(\lg^2 n)$ pebbles, then describe the pebbling algorithm which uses $O(\lg n)$ pebbles without considerations such as pebble maintenance, and finally describe a full implementation using a so-called recycling bin data structure.

The list pebbling algorithm uses a new set of pebbles, denoted as *green pebbles*. The pebbles used as described in Section 2 are now called the *blue pebbles*. The purpose of the green pebbles is to be kept as placeholders behind the blue pebbles, as those are moved to new nodes in forward traversal. Thus, getting back into a position for which a green pebble is still in place takes $O(1)$ time.

3.1 The Basic List Pebbling Algorithm

Define a *left-subpath* (*right-subpath*) as a path consisting of nodes that are all left children (right children). Consider the (blue-pebbled) path p from the root to node i . We say that v is a left-child of p if it has a right sibling that is in p

(that is, v is not in p , it is a left child, and its parent is in p but not the node i). As we move forward, green pebbles are placed on right-subpaths that begin at left children of p . Since p consists of at most $\lg n$ nodes, the number of green pebbles is at most $\lg^2 n$.

When moving backward, green pebbles will become blue, and as a result, their left subpaths will not be pebbled. Re-pebbling these sub-paths will be done when needed. When moving forward, if current position is an internal node, then p is extended with a new node, and a new blue pebble is created. No change occurs with the green pebbles. If the current position is a leaf, then the pebbles at the entire right-subpath ending with that leaf are converted from blue to green. Consequently, all the green sub-paths that are connected to this right-subpath are un-pebbled. That is, their pebbles are released and can be used for new blue pebbles.

We consider three types of back-steps:

(i) *Current position is a left child:* The predecessor is the parent, which is on p , and hence pebbled. Moving takes $O(1)$ time; current position is to be un-pebbled.

(ii) *Current position is a right child, and a green sub-path is connected to its parent:* Move to the leaf of the green sub-path in $O(1)$ time, convert the pebbles on this sub-path to blue, and un-pebble the current position.

(iii) *Current position is a right child, and its parent's sub-path is not pebbled:* Reconstruct the green pebbles on the right sub-path connected to its parent v , and act as in the second case. This reconstruction is obtained by executing forward traversal of the left sub-tree of v . We amortize this cost against the sequence of back-steps starting at the right sibling of v and ending at the current position. This sequence includes all nodes in the right sub-tree of v . Hence, each back-step is charged with one reconstruction step in this sub-tree.

Consider a back step from a node u . Since such back step can only be charged once for each complete sub-tree that u belongs to, we have:

Claim. Each back step can be charged at most $\lg n$ times.

We can conclude that the basic list pebbling algorithm supports $O(\lg n)$ amortized list-steps per back-step, one list-step per forward step, using $O(\lg^2 n)$ pebbles.

3.2 The List Pebbling Algorithm with $O(\lg n)$ Pebbles

The basic list pebbling algorithm is improved by the following modification. Let v be a left child of p and let v' be the right sibling of v . Denote v to be the *last*

left child of p if the left subpath starting at v' ends at the current position; let the right subpath starting at the last left child be the *last right subpath*. Then, if v is not the last left child of p , the number of pebbled nodes in the right subpath starting at v is at all time at most the length of the left subpath in p , starting at v' . If v is the last left child of p , the entire right subpath starting at v can be pebbled. We denote the (green) right subpath starting at v as the *mirror subpath* of the (blue) left subpath starting at v' . Nodes in the mirror subpath and the corresponding left subpath are said to be *mirrored* according to their order in the subpaths. The following clearly holds:

Claim. The number of green pebbles is at most $\lg n$.

When moving forward, there are two cases:

- (1) *Current position is an internal node:* as before, p is extended with a new node, and a new blue pebble is created. No change occurs with the green pebbles (note the mirror subpath begins at the last left child of p).
- (2) *Current position i is a leaf that is on a right subpath starting at v (which could be i , if i is a left child):* we pebble (blue) the new position, which is the right sibling of v , and the pebbles at the entire right subpath ending at i are converted from blue to green. Consequently, (1) all the green sub-paths that are connected to the right subpath starting at v are un-pebbled; and (2) the left subpath in p which ended at v now ends at the parent of v , so the mirror (green) node to v should now be un-pebbled. The released pebbles can be reused for new blue pebbles.

Moving backward is similar to the basic algorithm. There are three types of back-steps.

- (1) *Current position is a left child:* predecessor is the parent, which is on p , and hence pebbled. Moving takes $O(1)$ time; current position is to be un-pebbled. No change occurs with green pebbles, since the last right subpath is unchanged.
- (2) *Current position is a right child, and the (green) subpath connected to its parent is entirely pebbled:* Move to the leaf of the green subpath in $O(1)$ time, convert the pebbles on this subpath to blue, and un-pebble the current position. Since the new blue subpath is a left subpath, it does not have a mirror green subpath. However, if the subpath begins at v , then the left subpath in p ending at v is not extended, and its mirror green right subpath should be extended as well. This extension is deferred to the time the current position will become the end of this right subpath, addressed next.
- (3) *Current position is a right child, and the (green) subpath connected to its parent is only partially pebbled:* Reconstruct the green pebbles on the right subpath connected to its parent v , and act as in the second case. This reconstruction is obtained by executing forward traversal of the sub-tree T_1 starting at v , where v is the last pebbled node on the last right subpath. We amortize this cost against

the back traversal starting at the right child of the mirror node of v and ending at the current position. This sequence includes back-steps to all nodes in the left sub-tree T_2 of the mirror of v . This amortization is valid since the right child of v was un-pebbled in a forward step in which the new position was the right child of the mirror of v . Since the size of T_1 is twice the size of T_2 , each back-step is charged with at most two reconstruction steps.

As in Claim 3.1, we have that each back step can be charged at most $\lg n$ times, resulting with:

Theorem 1. *The list pebbling algorithm supports full traversal in at most $\lg n$ amortized list-steps per back-step, one list-step per forward step, using $2 \lg n$ pebbles.*

3.3 Full Algorithm Implementation Using the Recycling Bin Data Structure

The allocation of pebbles is handled by an auxiliary data structure, denoted as the *recycling bin data structure*, or *RB*. The RB data structure supports the following operations:

Put pebble: put a released pebble in the RB for future use; this occurs in the simple back-step, in which the current position is a left child, and therefore its predecessor is its parent. (Back-step Case 1.)

Get pebble: get a pebble from the RB; this occurs in a simple forward step, in which the successor of the node of the current position is its left child. (Forward-step Case 1.)

Put list: put a released list of pebbles - given by a pair of pointers to its head and to its tail - in the RB for future use; this occurs in the non-simple forward step, in which the pebbles placed on a full right path should be released. (Forward-step Case 2.)

Get list: get the most recent list of pebbles that was put in the RB and is still there (i.e., it was not yet requested by a get list operation); this occurs in a non-simple back-step, in which the current position is a right child, and therefore its predecessor is a rightmost leaf, and it is necessary to reconstruct the right path from the left sibling of the current position to its rightmost leaf. It is easy to verify that the list that is to be reconstructed is indeed the last list to be released and put in the RB. (Back-step Cases 2 or 3.)

The RB data structure consists of a bag of pebbles, and a set of lists consisting of pebbles and organized in a double-ended queue of lists. The bag can be implemented as, e.g., a stack. For each list, we keep a pointer to its header and a pointer to its tail, and the pairs of pointers are kept in doubly linked list, sorted by the order in which they were inserted to RB. Initially, the bag includes $2 \lg n$

pebbles and the lists queue is empty. Based on the claim, the $2 \lg n$ pebbles will suffice for all operations.

In situations in which we have a get pebble operation and an empty bag of pebbles, we take pebbles from one of the lists. For each list ℓ we keep a counter M_ℓ for the number of pebbles removed from the list.

The operations are implemented as follows:

Put pebble: Adding a pebble to the bag of pebbles (e.g., stack) is trivial; it takes $O(1)$ time.

Put list: a new list is added to the tail of the queue of lists in RB, to become the last list in the queue, and M_ℓ is set to 0. This takes $O(1)$ time.

Get pebble: If the bag of pebbles includes at least one pebble, return a pebble from the bag and remove it from there. If the bag is empty, then return and remove the last pebble from the list ℓ , which is the oldest among those having the minimum M , and increment its counter M_ℓ . This requires a priority queue according to the pairs $\langle M_\ell, R_\ell \rangle$ in lexicographic order, where R_ℓ is the rank of list ℓ in RB according to when it was put in it. We show below that such PQ can be supported in $O(1)$ time.

Get list: return the last list in the queue and remove it from RB. If pebbles were removed from this list (i.e., $M_\ell > 0$), then it should be reconstructed in $O(2^{M_\ell})$ time prior to returning it, as follows. Starting with the node v of the last pebble currently in the list, take 2^{M_ℓ} forward steps, and whenever reaching a node on the right path starting at node v place there a pebble obtained from RB using the get pebble operation. Note that this is Back-step Case 3, and according to the analysis and claim the amortized cost per back-step is $O(\lg n)$ time.

Claim. The priority queue can be implemented to support delmin operation in $O(1)$ time per retrieval.

We can conclude:

Theorem 2. *The list pebbling algorithm using the recycling bin data structure supports $O(\lg n)$ amortize time per back-step, $O(1)$ time per forward step, using $O(\lg n)$ pebbles.*

4 The Advanced List Pebbling Algorithm

The advanced list pebbling algorithm presented in this section supports back-steps in $O(\lg n)$ time per step in the worst case. Ensuring $O(\lg n)$ list steps per back-step in the worst case is obtained by processing the rebuilt of the missing green paths along the sequence of back traversal, using a new set of red pebbles. For each green path, there is one red pebble whose function is to progressively

move forward from the deepest pebbled node in the path, to reach the next node to be pebbled. By synchronizing the progression of the red pebbles with the back-steps, we can guarantee that green paths will be appropriately pebbled whenever needed.

The number of pebbles used by the algorithm is bounded by $\lg n$ (rather than $O(\lg n)$). This is obtained by a careful implementation and a sequence of refinements, described in the appendix.

Theorem 3. *The list pebbling algorithm can be implemented on a RAM to support $O(\lg i)$ time in the worst-case per back-step, where i is the distance from the current position to the farthest point traversed so far. Forward steps are supported in $O(1)$ time in the worst case, $1 + \epsilon$ amortized time per forward step, and no additional list-steps, using $\lg n$ pebbles. The memory used is at most $1.5(\lg n)$ words of $\lg n + O(\lg \lg n)$ bits each.*

5 Reversal of Program Execution

Uni-directional linked list can represent the execution of programs. A program state can be thought of as nodes of a list, and a program step is represented by a directed link between the nodes representing the appropriate program states. Since typically program states cannot be easily reversed, the list is in general uni-directional list.

Moving from a node in a linked list that represents a particular program back to its preceding node is equivalent to reversing the step represented by the link. Executing a back traversal on the linked list is hence equivalent to rolling back the program. Let the sequence of program states in a forward execution be $s_0, s_1, \dots, s_T$. A *rollback* of a program of some state s_j is changing its state to the preceding state s_{j-1} . A rollback step from state s_j is said to be the i 'th *rollback step* if state s_{j+i-1} is the farthest state that the program has reached so far.

We show how to efficiently support back traversal with negligible overhead to forward steps.

Theorem 4. *Let P be a program, using memory of size M and time T . Then, at the cost of recording the input used by the program, and increasing the memory by a factor of $O(\lg T)$ to $O(M \lg T)$, the program can be extended to support arbitrary rollback steps as follows. The i 'th rollback step takes $O(\lg i)$ time in the worst case, while forward steps take $O(1)$ time in the worst case, and $1 + \epsilon$ amortized time per step.*

The rolling method of Theorem 4 can be effectively combined with delta-encoding, which enables quick access to the last sequence of program states by encoding only the differences between them.

References

1. A. M. Ben-Amram and H. Petersen. Backing up in singly linked lists. In *Proceedings of the ACM STOC*, pages 780–786, 1999.
2. M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. P. Vadhan. The power of a pebble: Exploring and mapping directed graphs. In *ACM Symposium on Theory of Computing*, pages 269–278, 1998.
3. Y. C. Chung, S.-M. Moon, K. Ebcioglu, and D. Sahlin. Reducing sweep time for a nearly empty heap. In *27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '00)*, Boston, MA, 2000. ACM Press.
4. D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In *Proceedings of the Fifth Conference on Financial Cryptography (FC '02)*, 2002.
5. D. S. Hirschberg and S. S. Seiden. A bounded-space tree traversal algorithm. *Information Processing Letters*, 47(4):215–219, 1993.
6. M. Jakobsson. Fractal hash sequence representation and traversal. In *ISIT*, 2002.
7. Y. Matias and E. Porat. Efficient pebbling for list traversal synopses. Technical report, Tel Aviv University, 2002.
8. N. Pippenger. Advances in pebbling. In *In Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 407–417, 1982.
9. H. Schorr and W. M. Waite. An efficient machineindependent procedure for garbage collection in various list structures. *Communications of the ACM*, 10(8):501–506, Aug. 1967.
10. J. Sobel and D. P. Friedman. Recycling continuations. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, volume 34(1), pages 251–260, 1999.
11. D. Walker and J. G. Morrisett. Alias types for recursive data structures. In *Types in Compilation*, pages 177–206, 2000.

Function Matching: Algorithms, Applications, and a Lower Bound

Amihood Amir^{1*}, Yonatan Aumann¹, Richard Cole^{2**}, Moshe Lewenstein¹,
and Ely Porat¹

¹ Bar-Ilan University
amir@cs.biu.ac.il

{aumann,moshe,porately}@cs.biu.ac.il
² New York University cole@cs.nyu.edu

Abstract. We introduce a new matching criterion – *function matching* – that captures several different applications. The *function matching problem* has as its input a text T of length n over alphabet Σ_T and a pattern $P = P[1]P[2] \cdots P[m]$ of length m over alphabet Σ_P . We seek all text locations i for which, for some function $f : \Sigma_P \rightarrow \Sigma_T$ (f may also depend on i), the m -length substring that starts at i is equal to $f(P[1])f(P[2]) \cdots f(P[m])$.

We give a randomized algorithm which, for any given constant k , solves the function matching problem in time $O(n \log n)$ with probability $\frac{1}{n^k}$ of declaring a false positive. We give a deterministic algorithm whose time is $O(n|\Sigma_P| \log m)$ and show that it is almost optimal in the newly formalized convolutions model. Finally, a variant of the third problem is solved by means of two-dimensional parameterized matching, for which we also give an efficient algorithm.

Keywords: Pattern matching, function matching, parameterized matching, color indexing, register allocation, protein folding.

1 Introduction

In the traditional pattern matching model, one seeks exact occurrences of a given pattern in a text, i.e. text locations where every text symbol is *equal* to its corresponding pattern symbol. In the *parameterized matching* problem, introduced by Baker [7], one seeks text locations where there exists a bijection f on the alphabet for which every text symbol is equal to *the image under f* of the corresponding pattern symbol. In the applications we will describe below, f cannot be a bijection. Rather, it should be simply a function. More precisely, P matches T at location i if for every element $a \in \Sigma$, *all* occurrences of a have the same corresponding symbol in T . In other words, unlike in parameterized

* Partially supported by a FIRST grant of the Israel Academy of Sciences and Humanities, and NSF grant CCR-01-04494.

** Partially supported by NSF grant CCR-01-05678.

matching, there may be a *several* different symbols in the pattern which are mapped to the same text symbol.

Consider the following problems where parameterized matching is insufficient and function matching is required.

Programming Languages: There is a growing class of real-time systems applications where software codes are embedded on small chips with limited memory, e.g. chips in appliances. In these applications it is important to have as small a number of memory variables as possible. A similar problem exists in compiler design, where it is desirable to minimize the register-memory traffic, and re-use global registers as much as possible. This need to compact code by global register allocation and spill code minimization is an active research topic in the programming languages community (see e.g. [13,12]).

Automatically identifying functionally equivalent pieces of such compact code would make it easier to reuse these pieces (and, for example, to replace multiple such pieces by one piece in embedded code). Baker's parameterized matching was a first step in this direction. It identified codes that are identical, up to a one-to-one mapping of the variable names. This paper considers a generalization that identifies codes in which the mapping of the variable names (or registers) is possibly a many-to-one mapping. This identifies a larger set of candidate code portions which might be functionally equivalent (the equivalence would depend on the interleaving of and updates to the variables and so would require further postprocessing for confirmation).

Computational Biology: The Grand Challenge *protein folding* problem is one of the most important problems in computational biology (see e.g. [14]). The goal is to determine a protein's tertiary structure (how it folds) from the linear arrangement of its peptide sequence.

This is an area of extremely active research and a myriad of methods have and are being considered in attempts to solve this problem. One possible technique that is being investigated is *threading* (e.g. [8,17]). The idea is to try to "thread" a given protein sequence into a known structure. A starting point is to consider peptide subsequences that are known to fold in a particular way. These subsequences can be used as patterns. Given a new sequence, with unknown tertiary structure, one can seek known patterns in its peptide sequence, and use the folding of the known subsequences as a starting point in determining the full structure. However, a subsequence of different peptides that bond in the same way as the pattern peptides may still fold in a similar way. Such functionally equivalent subsequences will not be detected by exact matching. Function matching can serve as a filter, that identifies a superset of possible choices whose bondings can then be more carefully examined.

Image Processing: One of the interesting problems in web searching is searching for color images (e.g. [16,6,3]). The simplest possible cases is searching for an icon in a screen, a task that the Human-Computer Interaction Lab at the University of Maryland was confronted with. If the colors are fixed, this is exact two-dimensional pattern matching [2]. However, if the color maps in pattern and text differ, the exact matching algorithm would not find the pattern. Parame-

terized two dimensional search is precisely what is needed. If, in addition, we are faced with a loss of resolution in the text, e.g. due to truncation, then we would need to use a two dimensional function matching search.

The above examples are a sample of diverse application areas encountering search problems that are not solved by state of the art methods in pattern matching. This need led us to introduce, in this paper, the *function matching* criterion, and to explore the *two dimensional parameterized matching* problem.

Function matching is a natural generalization of parameterized matching. However, relaxing the bijection restriction introduces non-trivial technical difficulties. Many powerful pattern matching techniques such as automata methods, subword trees, dueling and deterministic sampling assume transitivity of the matching relation (see [10] for techniques). For any pattern matching criteria where transitivity does not exist, the above methods do not help.

Examples of pattern matching with non-transitive matching relation are string matching with “don’t cares”, less-than matching, pattern matching with mismatches and swapped matching. It is interesting to note that the efficient algorithms for solving the above problems all used convolutions as their main tool. Convolutions were introduced by Fischer and Paterson [11] as a technique for solving pattern matching problems with wildcards, where indeed the match relation is not transitive.

It turns out that many such problems can be solved by a “standard” application of convolutions (e.g. matching with “don’t cares”, matching with mismatches in bounded finite alphabets, and swapped matching). Muthukrishnan and Palem were the first to explicitly identify this application method and introduced a *boolean convolutions model* [15] with locality restrictions and obtained several lower bounds in this model. Since the introduction of the boolean convolutions model, several papers appeared using general, rather than boolean convolutions. In this paper we provide a formal definition for a more general convolutions model that broadens the class of problems being considered.

The new convolutions model encapsulates the solution to many non-standard matching problems. Even more importantly, a rigorous formal definition of such a model is useful in proving lower bounds. While such bounds do not lower bound the solution complexity in a general RAM, they do help in understanding the limits of the convolution method, hitherto the only powerful tool for non-standard pattern matching.

There are three main contributions in this paper.

1. A solution to a number of search problems in diverse fields, achieved by the introduction of a new type of generalized pattern matching, that of function matching.
2. A formalization of a new general convolutions model. This leads to a deterministic solution. We prove that this solution is almost *tight* in the convolutions model. We also present an efficient randomized solution of the function matching problem.
3. Solutions to the problem of exact search in color images with different color maps. This is done via efficient randomized and deterministic algorithms for two-dimensional parameterized and function matching.

In section 2 we give the basic definitions and present progressively more efficient deterministic solutions, culminating in a $O(n|\Sigma_P| \log m)$ algorithm, where $|\Sigma_P|$ is the pattern alphabet size. We also present a Monte Carlo algorithm that solves the function matching problem in time $O(n \log m)$ time with failure probability no larger than $\frac{1}{n^k}$, where k is a given constant. In section 3 we formalize the new convolution model. We then show a lower bound proving that our deterministic algorithm is tight in the convolutions model and discuss the limitations of that model. Finally, in section 4 we present a randomized algorithm that solves the two-dimensional parameterized matching problem in time $O(n^2 \log n)$ with probability of false positives no larger than $\frac{1}{n^k}$, for given constant k . We also present a deterministic algorithm that solves the two-dimensional parameterized matching problem in time $O(n^2 \log^2 m)$.

2 Algorithms

The key notion is that of a *cover*.

Definition: Let U and V be equal length strings. Symbol τ in U is said to *cover* symbol σ in V if every occurrence of σ in V is aligned with an occurrence of τ in U (i.e. they occur in equal index locations). U is said to cover σ in V if there is some symbol τ in U covering σ . Finally, the cover is said to be an *exact* cover if every occurrence of τ in U is aligned with an occurrence of σ in V .

Definition: There is a *function match* of V with U if every symbol occurring in V is covered by U (but this relation need not be symmetric). If each of the covers is an exact cover the match is a *parameterized match* (and this relation is symmetric).

The term function match arises by considering the mapping from V 's symbols to U 's symbols specified by the match; it is a plain function in a function match and it is one-to-one in a parameterized match. In both cases the function is onto.

Definition: Given a text T (of length n) and a pattern P (of length m) the function matching problem is to find the alignments (positionings) of P such that P function matches the aligned portion of T . Note that every match may use a different function to associate the symbols of P with those in the aligned portion of T .

As is standard, we can limit T to have length at most $2m$, by breaking T into pieces of length $2m$, successive pieces overlapping by $m - 1$ symbols.

It is straightforward to give an $O(nm)$ time algorithm for function matching; it simply checks each possible alignment of the pattern in turn, each in time $O(m)$. This is left to the reader.

We start by outlining a simple $O(n|\Sigma_P||\Sigma_T| \log m)$ time algorithm, where Σ_P and Σ_T are the pattern and text alphabets, respectively. This algorithm finds, for each pair $\sigma \in \Sigma_P$ and $\tau \in \Sigma_T$, those alignments of the pattern with the text for which τ covers σ . This will take time $O(n \log m)$ for one pair. A function matching exists for an alignment exactly if every symbol occurring in P is covered.

Definition: The σ -*indicator* of string U , $\chi_\sigma(U)$ is a binary string of length U in which each occurrence of σ is replaced by a 1, and every other symbol occurrence is replaced by 0.

The procedure used the strings $\chi_\sigma(P)$ and $\chi_\sigma(T)$. For each alignment of $\chi_\sigma(P)$ with $\chi_\sigma(T)$ it computes the dot product of $\chi_\sigma(P)$ and the aligned portion of $\chi_\sigma(T)$. But the product is exactly the number of occurrences of σ in P aligned with occurrences of τ in T . This is a cover of σ by τ exactly if the dot product equals the number of occurrences of σ in P . The dot products, for each alignment of $\chi_\sigma(P)$ with $\chi_\sigma(T)$, are all readily computed in $O(n \log m)$ time by means of a convolution [11]. We have shown:

Theorem 1. *Function matching can be solved deterministically in time $O(n|\Sigma_P||\Sigma_T| \log m)$.*

We obtain a faster algorithm by determining simultaneously for all τ occurring in T and for one σ occurring in P , those alignments of P for which some τ covers σ . This is done in time $O(n \log m)$ and is repeated for each σ yielding an algorithm with running time $O(n|\Sigma_P| \log m)$. Our algorithm exploits the following observation.

Lemma 1. *Let $a_1, \dots, a_k$ be natural numbers. Then $k \sum_{h=1}^k (a_h)^2 = (\sum_{h=1}^k a_h)^2$ iff $a_i = a_j$, for $1 \leq i < j \leq k$.*

The algorithm uses the strings T and T_2 , where T_2 is defined by $T_2[i] = (T[i])^2$, $i = 0, \dots, n-1$. For each σ , for each alignment of P with each of T and T_2 , the dot product of $\chi_\sigma(P)$ with the aligned portion of each of T and T_2 is computed. By Lemma 1 T covers σ in a given alignment exactly if the dot product of P with the aligned portion of T_2 is k times larger than the dot product of P with T , where k is the number of occurrences of σ in P . This yields:

Theorem 2. *The function matching problem can be solved deterministically in time $O(n|\Sigma_P| \log m)$.*

We seek further speedups via randomization. We give a Monte Carlo algorithm that, given a constant k , reports all function matches and with probability at most $\frac{1}{n^k}$ reports a non-match as a match. Our first step is to reduce function matching to *paired function matching*. In paired function matching the pattern is a *paired string*, a string in which each symbol appears at most twice. We then give a randomized algorithm for paired function matching.

For the reduction we create a new text T' , whose length is $2n$, and a new pattern P' , whose length is $2m$. There will be a match of P with T starting at location i in T exactly if there is a match of P' starting at location $2i-1$ in T' . T' is obtained by replacing each symbol in T by two consecutive instances of the same symbol; e.g. if $T = abca$ then $T' = aabbccaa$. To define P' , a little notation is helpful. Suppose symbol σ appears k times in P . Then new symbols $\sigma_1, \sigma_2, \dots, \sigma_{k+1}$ are used in P' . The i th occurrence of σ is replaced by the pair of symbols σ_i, σ_{i+1} . e.g. if $P = aababca$ then $P' = a_1a_2a_2a_3b_1b_2a_3a_4b_2b_3c_1c_2a_4a_5$. It is easy to see that function matches of P' in T' and of P in T correspond as described above. Thus it remains to give the algorithm for paired function matching.

This algorithm replaces the symbols of P' and T' by integers, chosen uniformly at random from the range $[1, 2n^{k+1}]$ as follows. For the text T' , for each symbol σ , a random value v_σ is chosen, and each occurrence of σ is replaced by v_σ ,

forming a string T'' . For the pattern P' , for each symbol σ , occurring twice, a random value u_σ is chosen. The first occurrence of σ is replaced by u_σ and the second occurrence by $-u_\sigma$; if a symbol occurs once it is replaced by value 0. This forms string P'' . Now, for each possible alignment of P'' with T'' , the dot product of P'' with the aligned portion of T'' is computed. Clearly, if there is a function match of P with T , the corresponding dot product evaluates to 0. We show that when there is a function mismatch, the corresponding dot product is non-zero with high probability.

If there is a function mismatch then there is a symbol σ in P'' aligned with distinct symbols τ and ρ in T'' . Imagine that the assignment of random values assigns values v_τ, v_ρ, u_σ last. Consider the dot product expressed as a function of v_τ, v_ρ, u_σ ; it has the form $A + Bv_\tau + Cv_\rho + u_\sigma(v_\tau - v_\rho)$ (assuming the τ and ρ aligned with σ appear in left to right order), where A, B , and C are the values obtained after making all the other random choices. It is easy to see that there is at most a $\frac{2}{2n^{k+1}} = \frac{1}{n^{k+1}}$ probability of this polynomial evaluating to 0. As there are $n - m + 1$ possible alignments of P with T , the overall failure probability is at most $\frac{1}{n^k}$. We have shown:

Theorem 3. *There is a randomized algorithm for function matching that, given a constant k , runs in time $O(kn \log m)$; it reports all function matches and, with probability at least $1 - \frac{1}{n^k}$ reports no mismatches as matches.*

3 Lower Bounds

The unfettered nature of the function matching problem is what makes it difficult. Traditional pattern matching methods such as automata, duels or witnesses, apparently are of no help since there is no transitivity in the function matching relation. Moreover, it is far from evident whether one can set rules during a pattern preprocessing stage that will allow text scanning, since the relationship between the text and pattern is quite loose. This is what pushed us to consider convolutions as the method for the upper bound. Unfortunately, our deterministic algorithm's complexity is no better than that of the naive for alphabets of unbounded size.

Whenever resorting to a randomized algorithm, it behooves the algorithm's developer to explain why they randomized. In this section we give evidence for the belief that an efficient deterministic solution to the problem, if such exists, may be very difficult. We do it by showing a lower bound of $\Omega(\frac{m}{b})$ convolutions with b -bit inputs and outputs for the function matching problem in the convolutions model. Convolutions, as a tool for string matching, were introduced by Fischer and Paterson [11]. Muthukrishnan and Palem [15] considered a Boolean convolutions model with locality restrictions for which they obtained a number of lower bounds. We did not find a formal definition of general convolutions as a resource in the literature. Recent uses of convolutions with non-Boolean inputs led us to broaden the class of convolutions being considered for lower bound proofs. In fact, Muthukrishnan and Palem proved a lower bound of $\Omega(\log \Sigma)$ boolean convolutions for string matching with wildcards with alphabet Σ ; but their lower bound does not hold for more general convolutions as indicated by

Cole and Hariharan's recent two convolution algorithm [9]. Our model does not cover all conceivable convolutions-based methods. However, it broadens the class for which lower bounds can be proven. The next subsection formally defines the general convolutions model that we propose.

3.1 The Convolutions Model

We begin by defining the class of problems that are solved by the convolutions model.

Definition: A *pattern matching problem* is defined as follows:

MATCH RELATION: A binary relation $M(a, b)$, where $a = a_0 \dots a_k$, $b = b_0 \dots b_\ell$ and $a, b \in \Sigma^*$.

INPUT: A *text array*, $T = T[0], \dots, T[n-1]$, and a *pattern array* $P = P[0], \dots, P[m-1]$, $P[i], T[j] \in \Sigma$, $i = 0, \dots, m-1$, $j = 0, \dots, n-1$.

OUTPUT: The set of indices $S \subseteq \{0, \dots, n-1\}$ where the pattern P *matches*, i.e. all indices i where $M(P, T_i)$, and T_i is the suffix of T starting at location i . We also call the output set of indices the *target elements*.

Example: *String Matching with Don't Cares*

The match relation is defined as follows. Let $\Sigma = \{0, 1\}$. Let $\phi \notin \Sigma$ (ϕ is the *don't care symbol*). Let $|a| = k$ and $|b| = \ell$. If $k > \ell$ then there is no match. Otherwise, a matches b iff $a_i = b_i$ or $a_i = \phi$ or $b_i = \phi$, $i = 0, \dots, k-1$. The text and pattern arrays are $T = T[0], \dots, T[n-1]$ and $P[0], \dots, P[m-1]$, respectively. The target elements are all locations i in the text array T where there is an exact occurrence of P (where ϕ matches both 0 and 1).

As its name suggests, the convolutions model uses convolutions as basic operations on arrays. Another basic operation it uses is *preprocessing*. There is a difference, however, between pattern and text preprocessing. We place no restriction on the pattern preprocessing. The text preprocessing, however, must be *local*. When proving lower bounds in the convolutions model, we are mainly interested in the *number* of convolutions necessary to achieve the solution, rather than the time complexity of the solution (this is akin to counting the number of comparisons in the comparison model for sorting).

Definition: Let g be a pattern preprocessing function. A *g -local text preprocessing function* $f_g : N^n \rightarrow N^n$ is a function for which there exists n functions $f_g^j : N \rightarrow N$, such that $(f_g(T))[j] = f_g^j(T[j])$, $j = 0, \dots, n-1$.

In words, the “locality” of function f_g is manifested by the fact that the value in index j of $f_g(T)$ is computed based solely on the pattern preprocessing (output $g(P)$), the index j , and the value of $T[j]$.

Examples:

1. Let T be an array. Then $\chi_a(T)$ is clearly a local array function, since the only index of T that participates in computing $\chi_a(T)[j]$ is j .
2. Let T be an array of numbers. The function f such that $f(T)[j] = T[j] - (\sum_{i=0}^{n-1} T[i])/n$ is not a local array function.

We now have all the building blocks of the convolutions model.

Definition: The *convolutions computation model* is a specialized model of computation that solves a subset of the pattern matching problems.

Given a pattern matching problem whose input is text T and pattern P , a solution in the convolutions model has the following form. Let g_i , $i = 1, \dots, h(n)$ be pattern preprocessing functions, and let f_{g_i} , $i = 1, \dots, h(n)$ be the corresponding local text preprocessing functions. The model also uses a parameter b .

1. Compute $h(n)$ convolutions $C_i \leftarrow f_{g_i}(T) \otimes g_i(P)$, $i = 1, \dots, h(n)$, with b -bit inputs and outputs.
2. Compute the matches as follows. The decision of whether location j of the text is a match is decided by a computation whose inputs are a subset of $\{C_i[j] \mid i = 1, \dots, h(n)\}$.

Examples:

1. *Exact String Matching with Don't Cares*

This problem's solution was provided by Fischer and Paterson [11] in the convolutions model. The two convolutions are:

$$C_1 \leftarrow \chi_0(T) \otimes \chi_1(P)$$

$$C_2 \leftarrow \chi_1(T) \otimes \chi_0(P)$$

The text locations i for which $C_1[i] = C_2[i] = 0$ are precisely the match locations.

2. *Approximate Hamming Distance over a fixed bounded Alphabet*

This problem was considered for unbounded alphabets in [1]. For bounded alphabets, the problem is defined in the convolutions model as follows. The matching relation $M_e(a, b)$ is all pairs of substrings over alphabet $\Sigma = \{1, \dots, k\}$ for which $|a| \leq |b|$ and the number of *mismatches* between a and b (i.e. the indices j for which $a_j \neq b_j$) is no greater than e . the solution in the convolutions model is as follows. Compute the convolutions:

$$C_i \leftarrow \chi_i(T) \otimes \chi_{\bar{i}}(P), \quad i = 1, \dots, k, \quad \text{where}$$

$$\chi_{\bar{a}}(x) = \begin{cases} 1 & \text{if } x \neq a \\ 0 & \text{if } x = a \end{cases}$$

The match locations are all indices j where $\sum_{i=1}^k C_i[j] \leq e$.

3. *Less-than Matching over a fixed bounded Alphabet*

This problem was considered for unbounded alphabets in [4]. For bounded alphabets, the problem is defined in the convolutions model as follows. The matching relation $M(a, b)$ is all pairs of substrings over alphabet $\Sigma = \{1, \dots, k\}$ for which $|a| \leq |b|$ and $a_j \leq b_j \quad \forall j = 0, \dots, |a| - 1$. the solution in the convolutions model is as follows. Compute the convolutions:

$$C_i \leftarrow \chi_i(T) \otimes \chi_{<i}(P), \quad i=1, \dots, k, \quad \text{where}$$

$$\chi_{<i}(x) = \begin{cases} 1 & \text{if } x < i \\ 0 & \text{if } x \geq i \end{cases}$$

The match locations are all indices j where $\sum_{i=1}^k C_i[j] \neq 0$.

4. Our solutions in Section 2 are also algorithms in the convolutions model.

3.2 Lower Bounds

The solutions we presented in section 2 for the function matching problem were also in the convolutions model. The following theorem shows that our algorithm's complexity is almost tight in the convolutions model.

Theorem 4. *The function matching problem requires $\Omega(\frac{m}{b})$ convolutions in the convolutions model.*

Proof: We will show that the *word equality problem* can be linearly reduced to the function matching problem. The word equality problem is:

INPUT: Two m bit words, $W_1 = W_1[0], \dots, W_1[m-1]$ and $W_2 = W_2[0], \dots, W_2[m-1]$.

DECIDE: Whether $W_1 = W_2$ (i.e. $W_1[i] = W_2[i]$, $i = 0, \dots, m-1$) or not.

The following communication complexity lower bound for the word equality problem is known. Suppose processor PA starts with word W_1 and processor PB with word W_2 . Then to decide word equality they need to exchange $\Omega(m)$ bits [18].

We show that any algorithm for function matching in the convolutions model using $h(m)$ b -bit convolutions can be used to solve the word equality problem with a transmission of $b \cdot h(m)$ bits, implying $h(m) = \Omega(\frac{m}{b})$.

We consider the operation of the function matching algorithm on the following pattern and text. $T = W_1W_2$, the concatenation of W_1 and W_2 , and $P = 1, 2, \dots, m, 1, 2, \dots, m$. Note that T function matches P if and only if $W_1 = W_2$.

Now suppose that function matching is solved by some algorithm F in the convolutions model. F computes $h(m)$ convolutions $C_1, \dots, C_{h(m)}$ and then uses the results of $C_i[1]$, $i = 1, \dots, h(m)$ to decide whether there is a function-match.

Note that for every convolution $C = A \otimes B$, $C[1] = \sum_{h=0}^{2m-1} A[h]B[h]$. However, this is equal to $\sum_{h=0}^{m-1} A[h]B[h] + \sum_{h=0}^{m-1} A[h+m]B[h+m]$. For each convolution, PA will compute $\sum_{h=0}^{m-1} A[h]B[h]$, which is based solely on $T[1], \dots, T[m]$ and P , and PB will compute $\sum_{h=0}^{m-1} A[h+m]B[h+m]$, which is based solely on $T[m+1], \dots, T[2m]$ and P . PA will then transmit its b -bit result to PB for each of the $h(m)$ convolutions used by F , and PB can at this point determine the result of the word equality problem. $\square$

It is important to be careful in interpreting the results of the convolutions model in a RAM complexity model since complexity in the convolutions model is measured by number of convolutions, rather than RAM operations. When evaluating the number of operations it takes to compute a convolution one must consider the number of bits in a RAM word. The standard in the pattern matching literature is an $O(\log m)$ bit word and the currently fastest known algorithm for computing convolutions is by using the FFT. Its time complexity is $O(n \log m)$ word operations. Thus, the number of RAM operations required to compute function matching in the convolutions model would appear to be $\Omega(nm)$. Of course, conceivably, by ingenious use of special case convolutions one might be able to evaluate the convolutions more quickly, though no such approach has occurred to us.

4 Two Dimensional Algorithms

The one dimensional parameterized matching problem was efficiently solved in [5]. However, as discussed in [3], the move to two dimensions implies a possible computational difficulty if no separable attributes exist. Parameterized matching is not separable – if all columns (or rows) have parameterized matches, it does not necessarily imply that the entire matrix has a parameterized match. Thus we are forced to seek other approaches.

Our Problem:

INPUT: Two dimensional text T of size $n \times n$, and two dimensional pattern P of size $m \times m$.

OUTPUT: All locations $[i, j]$ in T where there is a parameterized (function) occurrence of the pattern.

First we show how to reduce two-dimensional function matching to the one-dimensional case, yielding an $O(n^2 \log n)$ work randomized algorithm. We then show how, with an additional $O(n^2 \log n)$ work, to solve two-dimensional parameterized matching, again with a randomized algorithm. Finally, we give a deterministic algorithm for parameterized matching, which takes $O(n^2 \log^2 m)$ time.

The two-dimensional text T is written in row major order to give a one-dimensional text T' . The pattern is padded with wildcards (or equivalently, new characters, each appearing once) to produce m rows of length n ; the padded pattern is written in row major order to give a one-dimensional pattern P' . Clearly, there is a match of P at location (i, j) in T exactly if there is a match of P' at location $n(i - 1) + j$ in T' . We have shown:

Corollary 1. *There is a randomized algorithm for two-dimensional function matching, which, when given a constant k , runs in time $O(kn^2 \log n)$, reports all function matches, and with probability at most $\frac{1}{n^k}$ falsely reports a mismatch as a match.*

In a parameterized match, the number of distinct symbols in the aligned portion of the text must equal the number of distinct symbols in the pattern. Amir, Church and Dar [3] gave an $O(n^2 \log n)$ time algorithm for this problem, the character count problem: determine, for each $m \times m$ subarray of an $n \times n$ array, the number of distinct characters appearing in the subarray. So we have shown:

Corollary 2. *There is a randomized algorithm for two-dimensional parameterized matching, which, when given a constant k , runs in time $O(kn^2 \log n)$, reports all parameterized matches, and with probability at most $\frac{1}{n^k}$ falsely reports a mismatch as a match.*

Next, we give an efficient deterministic algorithm for two-dimensional parameterized matching (time $O(n^2 \log^2 m)$). It uses one convolution on a size $O(n^2 \log m)$ vector, which can also be viewed as $O(\log m)$ convolutions on size n^2 vectors. As in one-dimension, this is considerably more efficient than what is known for function matching. Incidentally, we note that this convolution is outside the convolutions model.

It is helpful to recall the one-dimensional parameterized matching algorithm, due to Amir, Farach, and Muthukrishnan [7]. It is similar to the Knuth-Morris-Pratt string matching algorithm. The key idea is to recode the occurrences of each symbol in terms of their separation; namely, if symbol a occurs in the pattern (or text) at locations with indices $i_1 \leq i_2 \leq \dots \leq i_k$, these occurrences of a are replaced by the numbers $0, i_2 - i_1, i_3 - i_2, \dots, i_k - i_{k-1}$, respectively. For each symbol occurrence, this is simply the distance to the nearest occurrence of the same symbol to the left, if any. Except for the first occurrence of each symbol, a parameterized match in the original text and pattern corresponds to a standard match in the recoded text and pattern.

One perspective on this is that all occurrences of the same symbol in the pattern (and the text) have been connected into a structure; identifying a match becomes a question of finding the alignments for which the structures in the pattern and text match. We will seek a similar construction for the two-dimensional problem. However, this will now require creating connections to $O(\log n)$ neighbors of each symbol occurrence.

Our solution has the following form. For each occurrence I of a symbol in the pattern (resp. in the text) the relative locations of some $8 \log n$ instances $J_1, J_2, \dots$ of the same symbol (resp. in the pattern and text) are recorded. We say that J_ℓ is a neighbor of I , for $\ell = 1, 2, \dots$, and also that I and J_ℓ are *linked*. If I is in position (w, y) and J_ℓ is in position (x, z) , their relative position is recorded as $(x - w, z - y)$. Each potential neighbor is selected according to a specific rule, which may or may not identify a neighbor (e.g., a rule could be: the next occurrence of the symbol to the right in the same row). The rules for pattern and text will be slightly different.

The collections of selected symbols have the following properties:

- (i) For each alignment of the pattern with the text, for each symbol a , if the occurrences of a in the pattern and text match, then for each occurrence I_p of a in the pattern and the aligned occurrence I_t in the text, the neighbor of I_p selected by the i th rule for the pattern is aligned with the neighbor of I_t selected by the i th rule for the text (the converse need not be true, however).
- (ii) All occurrences of a in the pattern are linked, for each symbol a .

To see why the converse need not hold in Property (i) consider the rule “the next instance of this symbol to the right in the same row”; since the text may extend further to the right than the pattern, for a given alignment this rule could yield a symbol occurrence in the text and not in the pattern (the text symbol in question would be to the right of the pattern) and of course this does not preclude a match.

The text and pattern are recoded using the following encoding for each symbol occurrence. Each symbol occurrence is encoded by an equal length sequence of $O(\log n)$ relative positions, ordered as follows: the relative position of the symbol occurrence yielded by rule (1), by rule (2), by rule (3), and so on. If a rule yields no occurrence this is recoded by the “relative position” $(0, 0)$.

The matching problem is made one-dimensional by writing the recoded arrays in row major order using rows of length n (the missing entries in the pattern are replaced by sequences of $O(\log n)$ pairs $(0, 0)$). Treating $(0, 0)$ as a wildcard, it is easy to see that a parameterized match of the original pattern with the

text corresponds exactly to a standard match with wildcards of the recoded pattern with the recoded text. The recoded text has length $O(n^2 \log m)$, thus this wildcard matching can be solved in time $O(n^2 \log m \log n)$ [9] (and by standard techniques this can be reduced to $O(n^2 \log^2 m)$ time).

We turn to the selection of neighbors. For the moment we suppose the pattern dimension $m = 2^i$ for some integer i . For each location (x, y) in the pattern, we divide most of the remainder of the pattern (excluding location (x, y)) into $8 \log n - 4$ disjoint rectangles. Each rectangle provides one neighbor.

The first four rectangles comprise row x and column y partitioned at location (x, y) , i.e., (i) the points (x, z) with $z > y$, (ii) the points (x, w) with $w < x$, (iii) the points (z, y) with $z > x$, and (iv) the points (w, y) with $w < x$. Next, we describe how the quadrant below and to the right of (x, y) is divided into contiguous rectangles. Each rectangle comprises a distinct selection of contiguous rows, covering all columns from $y + 1$ to m , starting at row $x + 1$. From top to bottom, the sequence of rectangles have the following number of rows: $1, 2, 4, \dots, m/4 = 2^{i-2}, m/4, m/8, \dots, 2, 1, 1$, with the series stopping at the last rectangle that fits inside the pattern. This may mean that a portion of the quadrant is left uncovered.

Suppose a is the symbol in location (x, y) . Each rectangle is traversed in column major order to find the first occurrence of an a , if any. These are the neighbors of the a in location (x, y) . Analogous partitionings and traversals in directions away from location (x, y) are used for the other quadrants.

A very similar partitioning is used on the text, except that now each rectangle extends through $n - 1$ columns or to the right boundary of the text, whichever comes sooner. (This is for the SE quadrant; other quadrants are handled similarly.)

Clearly Property (i) above holds. It remains to show Property (ii).

Lemma 2. *Let (w, y) and (x, z) be two locations in the pattern both holding symbol a . Then they are linked.*

Proof: Clearly, if $w = x$ there is a series of links along row x connecting these two locations. Similarly if $y = z$.

So WLOG suppose that $w < x$ and $y < z$. We claim that either (x, z) lies in one of the rectangles defined for location (w, y) or (w, y) lies in one of the rectangles for (x, z) (or possibly both).

Suppose that $2^{k-1} < w \leq 2^k \leq m/2$. Then for (x, z) to lie outside one of (w, y) 's rectangles, $x > n - 2^{k-1}$ (for rows $w, w + 1, [w + 2, w + 3], \dots, [w + m/4, w + m/2 - 1], [w + m - m/2, w + m - m/4 - 1], \dots, [w + m - 2^{k+1}, w + m - 2^k - 1]$ are all included in (w, y) 's rectangles, and $w \geq 2^{k-1} + 1$). The symmetric argument for location (x, z) shows that (w, y) lies in one of z 's rectangles if $x > n - 2^{k-1}$. This argument does not cover the case $w = 1$, but then (w, y) 's rectangles cover every row, nor the case $w > m/2$, but then (x, z) 's rectangles cover row w .

WLOG suppose that (x, z) lies in one of (w, y) 's rectangles. It need not be that (x, z) is a neighbor of (w, y) , however. Nonetheless, by induction on $z - y$, we show they are linked. The base case, $y = z$, has already been demonstrated. Let (u, v) denote the neighbor of (w, y) in the rectangle containing (x, z) . Then

$y < v \leq z$. By induction, (u, v) and (x, z) are linked and the inductive claim follows. $\square$

It remains to show how to identify the neighbors. This is readily done in $O(m^2 \log^2 m)$ time in the pattern and $O(n^2 \log n \log m)$ time in the text (and using standard additional techniques, in $O(n^2 \log^2 m)$ time). We describe the approach for the pattern.

The idea is to maintain, for each symbol a , a window of 2^i rows, for $i = 1, 2, \dots, \log m - 2$, and in turn to slide each window down the pattern. In the window the occurrences of a are kept in a balanced tree in column major order. For each occurrence of a , its neighbors in the relevant window are found by means of $O(\log m)$ time searches. Thus, over all symbols and neighbors the searches take time $O(m^2 \log^2 m)$. To slide a window one row down requires deleting some symbol instances and adding others. This takes time $O(\log m)$ per change and as each symbol instance is added once and deleted once from a window of each size this takes time $O(m^2 \log^2 m)$ over all symbols and windows. (It is helpful to have a list of each character in row major order so as to be able to quickly decide which characters to add and to delete from the sliding window, but these lists take only $O(m^2)$ time to prepare for all the symbols.)

The preprocessing of the text is similar.

To extend this algorithm to arbitrary n , we simply expand the pattern to size $2^i \times 2^i$ by padding it with wildcards. We have shown:

Theorem 5. *There is an $O(n^2 \log^2 m)$ time algorithm for two-dimensional parameterized matching.*

References

1. K. Abrahamson. Generalized string matching. *SIAM J. Comp.*, 16(6):1039–1051, 1987.
2. A. Amir, G. Benson, and M. Farach. An alphabet independent approach to two dimensional pattern matching. *SIAM J. Comp.*, 23(2):313–323, 1994.
3. A. Amir, K. W. Church, and E. Dar. Separable attributes: a technique for solving the submatrices character count problem. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 400–401, 2002.
4. A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation*, 118(1):1–11, April 1995.
5. A. Amir, M. Farach, and S. Muthukrishnan. Alphabet dependence in parameterized matching. *Information Processing Letters*, 49:111–115, 1994.
6. G.P. Babu, B.M. Mehtre, and M.S. Kankanhalli. Color indexing for efficient image retrieval. *Multimedia Tools and Applications*, 1(4):327–348, Nov. 1995.
7. B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proc. 25th Annual ACM Symposium on the Theory of Computation*, pages 71–80, 1993.
8. J. H. Bowie, R. Luthy, and D. Eisenberg. A method to identify protein sequences that fold into a known three-dimensional structure. *Science*, (253):164–176, 1991.
9. R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proc. 34th Annual Symposium on the Theory of Computing (STOC)*, pages 592–601, 2002.
10. M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

11. M.J. Fischer and M.S. Paterson. String matching and other products. *Complexity of Computation*, R.M. Karp (editor), SIAM-AMS Proceedings, 7:113–125, 1974.
12. W.C. Krehling and C. Norris. Profile assisted register allocation. In *Proc. ACM Symp. on Applied Computing (SAC)*, pages 774–781, 2000.
13. G-Y. Lueh, T. Gross, and A-R. Adl-Tabatabai. Fusion-based register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22(3):431–470, 2000.
14. Jr. K. Merz and S. M. La Grand. *The Protein Folding Problem and Tertiary Structure Prediction*. Birkhauser, Boston, 1994.
15. S. Muthukrishnan and K. Palem. Non-standard stringology: Algorithms and complexity. In *Proc. 26th Annual Symposium on the Theory of Computing*, pages 770–779, 1994.
16. M. Swain and D. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.
17. J. Yadgari, Amihood Amir, and Ron Unger. Genetic algorithms for protein threading. In J. Glasgow, T. Littlejohn, F. Major, R. Lathrop, D. Sankoff, and C. Sensen, editors, *Proc. 6th Int'l Conference on Intelligent Systems for Molecular Biology (ISMB 98)*, pages 193–202. AAAI, AAAI Press, 1998.
18. A. C. C. Yao. Some complexity questions related to distributed computing. In *Proc. 11th Annual Symposium on the Theory of Computing (STOC)*, pages 209–213, 1979.

Simple Linear Work Suffix Array Construction^{*}

Juha Kärkkäinen and Peter Sanders

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
[juha,sanders]@mpi-sb.mpg.de.

Abstract. A suffix array represents the suffixes of a string in sorted order. Being a simpler and more compact alternative to suffix trees, it is an important tool for full text indexing and other string processing tasks. We introduce the *skew algorithm* for suffix array construction over integer alphabets that can be implemented to run in linear time using integer sorting as its only nontrivial subroutine:

1. recursively sort suffixes beginning at positions $i \bmod 3 \neq 0$.
2. sort the remaining suffixes using the information obtained in step one.
3. merge the two sorted sequences obtained in steps one and two.

The algorithm is much simpler than previous linear time algorithms that are all based on the more complicated suffix tree data structure. Since sorting is a well studied problem, we obtain optimal algorithms for several other models of computation, e.g. external memory with parallel disks, cache oblivious, and parallel. The adaptations for BSP and EREW-PRAM are asymptotically faster than the best previously known algorithms.

1 Introduction

The suffix *tree* [39] of a string is a compact trie of all the suffixes of the string. It is a powerful data structure with numerous applications in computational biology [21] and elsewhere [20]. One of the important properties of the suffix tree is that it can be constructed in linear time in the length of the string. The classical linear time algorithms [32,36,39] require a constant alphabet size, but Farach's algorithm [11,14] works also for integer alphabets, i.e., when characters are polynomially bounded integers. There are also efficient construction algorithms for many advanced models of computation (see Table 1).

The suffix *array* [18,31] is a lexicographically sorted array of the suffixes of a string. For several applications, the suffix array is a simpler and more compact alternative to the suffix tree [2,6,18,31]. The suffix array can be constructed in linear time by a lexicographic traversal of the suffix tree, but such a construction loses some of the advantage that the suffix array has over the suffix tree. The fastest *direct* suffix array construction algorithms that do not use suffix trees require $\mathcal{O}(n \log n)$ time [5,30,31]. Also under other models of computation, direct

^{*} Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

algorithms cannot match suffix tree based algorithms [9,16]. The existence of an I/O-optimal direct algorithm is mentioned as an important open problem in [9].

We introduce the *skew algorithm*, the first linear-time direct suffix array construction algorithm for integer alphabets. The skew algorithm is simpler than any suffix tree construction algorithm. (In the appendix, we give a 50 line C++ implementation.) In particular, it is much simpler than linear time suffix tree construction for integer alphabets.

Independently of and in parallel with the present work, two other direct linear time suffix array construction algorithms have been introduced by Kim et al. [28], and Ko and Aluru [29]. The two algorithms are quite different from ours (and each other).

The skew algorithm. Farach’s linear-time suffix tree construction algorithm [11] as well as some parallel and external algorithms [12,13,14] are based on the following divide-and-conquer approach:

1. Construct the suffix tree of the suffixes starting at odd positions. This is done by reduction to the suffix tree construction of a string of half the length, which is solved recursively.
2. Construct the suffix tree of the remaining suffixes using the result of the first step.
3. Merge the two suffix trees into one.

The crux of the algorithm is the last step, merging, which is a complicated procedure and relies on structural properties of suffix trees that are not available in suffix arrays. In their recent direct linear time suffix array construction algorithm, Kim et al. [28] managed to perform the merging using suffix arrays, but the procedure is still very complicated.

The skew algorithm has a similar structure:

1. Construct the suffix array of the suffixes starting at positions $i \bmod 3 \neq 0$. This is done by reduction to the suffix array construction of a string of two thirds the length, which is solved recursively.
2. Construct the suffix array of the remaining suffixes using the result of the first step.
3. Merge the two suffix arrays into one.

Surprisingly, the use of two thirds instead of half of the suffixes in the first step makes the last step almost trivial: a simple comparison-based merging is sufficient. For example, to compare suffixes starting at i and j with $i \bmod 3 = 0$ and $j \bmod 3 = 1$, we first compare the initial characters, and if they are the same, we compare the suffixes starting at $i + 1$ and $j + 1$ whose relative order is already known from the first step.

Results. The simplicity of the skew algorithm makes it easy to adapt to other models of computation. Table 1 summarizes our results together with the best previously known algorithms for a number of important models of computation. The column “alphabet” in Table 1 identifies the model for the alphabet Σ .

In a *constant* alphabet, we have $|\Sigma| = \mathcal{O}(1)$, an *integer* alphabet means that characters are integers in a range of size $n^{\mathcal{O}(1)}$, and *general* alphabet only assumes that characters can be compared in constant time.

Table 1. Suffix array construction algorithms. The algorithms in [11,12,13,14] are *indirect*, i.e., they actually construct a suffix *tree*, which can then be transformed into a suffix array

model of computation	complexity	alphabet	source
RAM	$\mathcal{O}(n \log n)$ time $\mathcal{O}(n)$ time	general integer	[31,30,5] [11,28,29],skew
External Memory [38] D disks, block size B , fast memory of size M	$\mathcal{O}\left(\frac{n}{DB} \log_{\frac{M}{B}} \frac{n}{B} \log_2 n\right)$ I/Os $\mathcal{O}\left(n \log_{\frac{M}{B}} \frac{n}{B} \log_2 n\right)$ internal work $\mathcal{O}\left(\frac{n}{DB} \log_{\frac{M}{B}} \frac{n}{B}\right)$ I/Os $\mathcal{O}\left(n \log_{\frac{M}{B}} \frac{n}{B}\right)$ internal work	integer integer	[9] [14],skew
Cache Oblivious [15] M/B cache blocks of size B	$\mathcal{O}\left(\frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B} \log_2 n\right)$ cache faults $\mathcal{O}\left(\frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B}\right)$ cache faults	general general	[9] [14],skew
BSP [37] P processors h -relation in time $L + gh$ $P = \mathcal{O}(n^{1-\epsilon})$ processors	$\mathcal{O}\left(\frac{n \log n}{P} + \left(L + \frac{gn}{P}\right) \frac{\log^3 n \log P}{\log(n/P)}\right)$ time $\mathcal{O}\left(\frac{n \log n}{P} + L \log^2 P + \frac{gn \log n}{P \log(n/P)}\right)$ time $\mathcal{O}(n/P + L \log^2 P + gn/P)$ time	general general integer	[12] skew skew
EREW-PRAM [25]	$\mathcal{O}(\log^4 n)$ time, $\mathcal{O}(n \log n)$ work $\mathcal{O}(\log^2 n)$ time, $\mathcal{O}(n \log n)$ work	general general	[12] skew
arbitrary-CRCW-PRAM [25]	$\mathcal{O}(\log n)$ time, $\mathcal{O}(n)$ work (rand.)	constant	[13]
priority-CRCW-PRAM [25]	$\mathcal{O}(\log^2 n)$ time, $\mathcal{O}(n)$ work (rand.)	constant	skew

The skew algorithm for RAM, external memory and cache oblivious models is the first optimal *direct* algorithm. For BSP and EREW-PRAM models, we obtain an improvement over *all* previous results, including the first linear work BSP algorithm. On all the models, the skew algorithm is much simpler than the best previous algorithm.

In many applications, the suffix array needs to be augmented with additional data, the most important being the longest common prefix (lcp) array [1,2,26,27,31]. In particular, the suffix tree can be constructed easily from the suffix and lcp arrays [11,13,14]. There is a linear time algorithm for computing the lcp array from the suffix array [27], but it does not appear to be suitable for parallel or external computation. We extend our algorithm to compute also the lcp array while retaining the complexities of Table 1. Hence, we also obtain improved suffix tree construction algorithms for the BSP and EREW-PRAM models.

The paper is organized as follows. In Section 2, we describe the basic skew algorithm, which is then adapted to different models of computation in Section 3. The algorithm is extended to compute the longest common prefixes in Section 4.

2 The Skew Algorithm

For compatibility with C and because we use many modulo operations we start arrays at position 0. We use the abbreviations $[a, b] = \{a, \dots, b\}$ and $s[a, b] = [s[a], \dots, s[b]]$ for a string or array s . Similarly, $[a, b) = [a, b - 1]$ and $s[a, b) = s[a, b - 1]$. The operator $\circ$ is used for the concatenation of strings. Consider a string $s = s[0, n)$ over the alphabet $\Sigma = [1, n]$. The suffix array SA contains the suffixes $S_i = s[i, n)$ in sorted order, i.e., if $\text{SA}[i] = j$ then suffix S_j has rank $i + 1$ among the set of strings $\{S_0, \dots, S_{n-1}\}$. To avoid tedious special case treatments, we describe the algorithm for the case that n is a multiple of 3 and adopt the convention that all strings α considered have $\alpha[|\alpha|] = \alpha[|\alpha| + 1] = 0$. The implementation in the Appendix fills in the remaining details. Figure 1 gives an example.

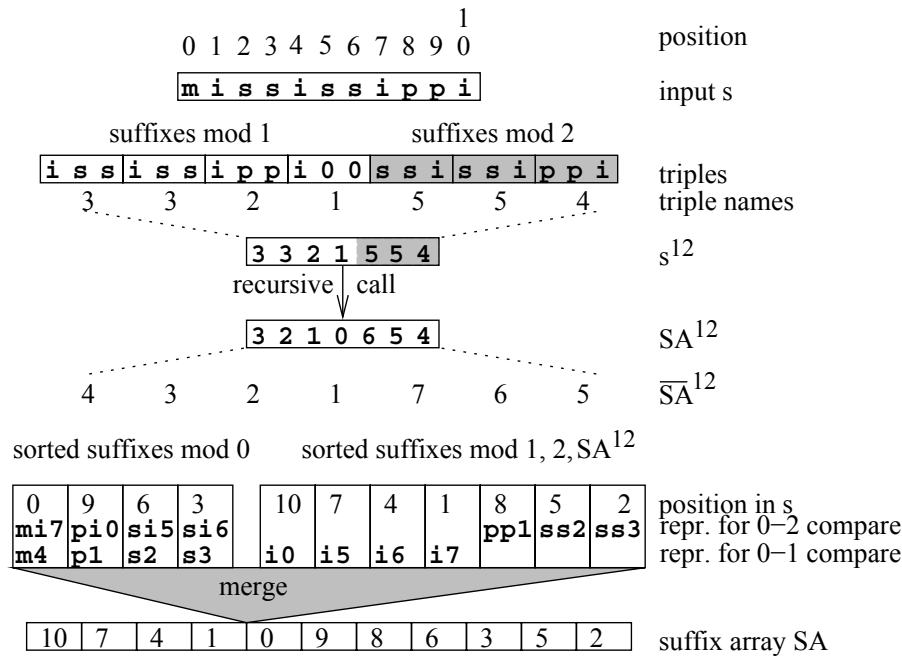


Fig. 1. The skew algorithm applied to $s = \text{mississippi}$.

The first and most time consuming step of the skew algorithm sorts the suffixes S_i with $i \bmod 3 \neq 0$ among themselves. To this end, it first finds *lexicographic names* $s'_i \in [1, 2n/3]$ for the triples $s[i, i + 2]$ with $i \bmod 3 \neq 0$, i.e., numbers with the property that $s'_i \leq s'_j$ if and only if $s[i, i + 2] \leq s[j, j + 2]$. This can be done in linear time by radix sort and scanning the sorted sequence

of triples — if triple $s[i, i + 2]$ is the k -th different triple appearing in the sorted sequence, we set $s'_i = k$.

If all triples get different lexicographic names, we are done with step one. Otherwise, the suffix array SA^{12} of the string

$$s^{12} = [s'_i : i \bmod 3 = 1] \circ [s'_i : i \bmod 3 = 2]$$

is computed recursively. Note that there can be no more lexicographic names than characters in s^{12} so that the alphabet size in a recursive call never exceeds the size of the string. The recursively computed suffix array SA^{12} represents the desired order of the suffixes S_i with $i \bmod 3 \neq 0$. To see this, note that $s^{12}[\frac{i-1}{3}, \frac{n}{3})$ for $i \bmod 3 = 1$ represents the suffix $S_i = s[i, n) \circ [0]$ via lexicographic naming. The 0 characters at the end of s make sure that $s^{12}[n/3 - 1]$ is unique in s^{12} so that it does not matter that s^{12} has additional characters. Similarly, $s^{12}[\frac{n+i-2}{3}, \frac{2n}{3})$ for $i \bmod 3 = 2$ represents the suffix $S_i = s[i, n) \circ [0, 0]$.

The second step is easy. The suffixes S_i with $i \bmod 3 = 0$ are sorted by sorting the pairs $(s[i], S_{i+1})$. Since the order of the suffixes S_{i+1} is already implicit in SA^{12} , it suffices to stably sort those entries $SA^{12}[j]$ that represent suffixes S_{i+1} , $i \bmod 3 = 0$, with respect to $s[i]$. This is possible in linear time by a single pass of radix sort.

The skew algorithm is so simple because also the third step is quite easy. We have to merge the two suffix arrays to obtain the complete suffix array SA . To compare a suffix S_j with $j \bmod 3 = 0$ with a suffix S_i with $i \bmod 3 \neq 0$, we distinguish two cases:

If $i \bmod 3 = 1$, we write S_i as $(s[i], S_{i+1})$ and S_j as $(s[j], S_{j+1})$. Since $i + 1 \bmod 3 = 2$ and $j + 1 \bmod 3 = 1$, the relative order of S_{j+1} and S_{i+1} can be determined from their position in SA^{12} . This position can be determined in constant time by precomputing an array $\overline{SA}^{12}$ with $\overline{SA}^{12}[i] = j + 1$ if $SA^{12}[j] = i$. This is nothing but a special case of lexicographic naming.¹

Similarly, if $i \bmod 3 = 2$, we compare the triples $(s[i], s[i + 1], S_{i+2})$ and $(s[j], s[j + 1], S_{j+2})$ replacing S_{i+2} and S_{j+2} by their lexicographic names in $\overline{SA}^{12}$.

The running time of the skew algorithm is easy to establish.

Theorem 1. *The skew algorithm can be implemented to run in time $\mathcal{O}(n)$.*

Proof. The execution time obeys the recurrence $T(n) = \mathcal{O}(n) + T(\lceil 2n/3 \rceil)$, $T(n) = \mathcal{O}(1)$ for $n < 3$. This recurrence has the solution $T(n) = \mathcal{O}(n)$. $\square$

3 Other Models of Computation

Theorem 2. *The skew algorithm can be implemented to achieve the following performance guarantees on advanced models of computation:*

¹ $\overline{SA}^{12} - 1$ is also known as the *inverse suffix array* of SA^{12} .

model of computation	complexity	alphabet
External Memory [38] D disks, block size B , fast memory of size M	$\mathcal{O}\left(\frac{n}{DB} \log_{\frac{M}{B}} \frac{n}{B}\right)$ I/Os $\mathcal{O}\left(n \log_{\frac{M}{B}} \frac{n}{B}\right)$ internal work	integer
Cache Oblivious [15]	$\mathcal{O}\left(\frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B}\right)$ cache faults	general
BSP [37] P processors h -relation in time $L + gh$	$\mathcal{O}\left(\frac{n \log n}{P} + L \log^2 P + \frac{gn \log n}{P \log(n/P)}\right)$ time	general
$P = \mathcal{O}(n^{1-\epsilon})$ processors	$\mathcal{O}(n/P + L \log^2 P + gn/P)$ time	integer
EREW-PRAM [25]	$\mathcal{O}(\log^2 n)$ time and $\mathcal{O}(n \log n)$ work	general
priority-CRCW-PRAM [25]	$\mathcal{O}(\log^2 n)$ time and $\mathcal{O}(n)$ work (rand.)	constant

Proof. External Memory: Sorting tuples and lexicographic naming is easily reduced to external memory integer sorting. I/O optimal deterministic² parallel disk sorting algorithms are well known [34,33]. We have to make a few remarks regarding internal work however. To achieve optimal internal work for all values of n , M , and B , we can use radix sort where the most significant digit has $\lfloor \log M \rfloor - 1$ bits and the remaining digits have $\lfloor \log M/B \rfloor$ bits. Sorting then starts with $\mathcal{O}(\log_{M/B} n/M)$ data distribution phases that need linear work each and can be implemented using $\mathcal{O}(n/DB)$ I/Os using the same I/O strategy as in [33]. It remains to stably sort the elements by their $\lfloor \log M \rfloor - 1$ most significant bits. For this we can use the distribution based algorithm from [33] directly. In the distribution phases, elements can be put into a bucket using a full lookup table mapping keys to buckets. Sorting buckets of size M can be done in linear time using a linear time internal algorithm.

Cache Oblivious: We use the comparison based model here since it is not known how to do cache oblivious integer sorting with $\mathcal{O}(\frac{n}{B} \log_{M/B} \frac{n}{B})$ cache faults and $o(n \log n)$ work. The result is an immediate corollary of the optimal comparison based sorting algorithm [15].

EREW PRAM: We can use Cole’s merge sort [8] for sorting and merging. Lexicographic naming can be implemented using linear work and $\mathcal{O}(\log P)$ time using prefix sums. After $\mathcal{O}(\log P)$ levels of recursion, the problem size has reduced so far that the remaining subproblem can be solved on a single processor. We get an overall execution time of $\mathcal{O}(n \log n/P + \log^2 P)$.

BSP: For the case of many processors, we proceed as for the EREW-PRAM algorithm using the optimal comparison based sorting algorithm [19] that takes time $\mathcal{O}(n \log n/P + (gn/P + L) \frac{\log n}{\log(n/P)})$.

For the case of few processors, we can use a linear work sorting algorithm based on radix sort [7] and a linear work merging algorithm [17]. The integer

² Simpler randomized algorithms with favorable constant factors are also available [10].

sorting algorithm remains applicable at least during the first $\Theta(\log \log n)$ levels of recursion of the skew algorithm. Then we can afford to switch to a comparison based algorithm without increasing the overall amount of internal work.

CRCW PRAM: We employ the stable integer sorting algorithm [35] that works in $\mathcal{O}(\log n)$ time using linear work for keys with $\mathcal{O}(\log \log n)$ bits. This algorithm can be used for the first $\Theta(\log \log \log n)$ iterations. Then we can afford to switch to the algorithm [22] that works for polynomial size keys at the price of being inefficient by a factor $\mathcal{O}(\log \log n)$. Lexicographic naming can be implemented by computing prefix sums using linear work and logarithmic time. Comparison based merging can be implemented with linear work and $\mathcal{O}(\log n)$ time using [23]. $\square$

The resulting algorithms are simple except that they may use complicated subroutines for sorting to obtain theoretically optimal results. There are usually much simpler implementations of sorting that work well in practice although they may sacrifice determinism or optimality for certain combinations of parameters.

4 Longest Common Prefixes

Let $\text{lcp}(i, j)$ denote the length of the longest common prefix (lcp) of the suffixes S_i and S_j . The longest common prefix array LCP contains the lengths of the longest common prefixes of suffixes that are adjacent in the suffix array, i.e., $\text{LCP}[i] = \text{lcp}(\text{SA}[i], \text{SA}[i + 1])$. A well-known property of lcp's is that for any $0 \leq i < j < n$,

$$\text{lcp}(i, j) = \min_{i \leq k < j} \text{LCP}[k] .$$

Thus, if we preprocess LCP in linear time to answer range minimum queries in constant time [3,4,24], we can find the longest common prefix of any two suffixes in constant time.

We will show how the LCP array can be computed from the LCP^{12} array corresponding to SA^{12} in linear time. Let $j = \text{SA}[i]$ and $k = \text{SA}[i + 1]$. We explain two cases; the others are similar.

First, assume that $j \bmod 3 = 1$ and $k \bmod 3 = 2$, and let $j' = (j - 1)/3$ and $k' = (n + k - 2)/3$ be the corresponding positions in s^{12} . Since j and k are adjacent in SA, so are j' and k' in SA^{12} , and thus $\ell = \text{lcp}^{12}(j', k') = \text{LCP}^{12}[\overline{\text{SA}}^{12}[j'] - 1]$. Then $\text{LCP}[i] = \text{lcp}(j, k) = 3\ell + \text{lcp}(j + 3\ell, k + 3\ell)$, where the last term is at most 2 and can be computed in constant time by character comparisons.

As the second case, assume $j \bmod 3 = 0$ and $k \bmod 3 = 1$. If $s[j] \neq s[k]$, $\text{LCP}[i] = 0$ and we are done. Otherwise, $\text{LCP}[i] = 1 + \text{lcp}(j + 1, k + 1)$, and we can compute $\text{lcp}(j + 1, k + 1)$ as above as $3\ell + \text{lcp}(j + 1 + 3\ell, k + 1 + 3\ell)$, where $\ell = \text{lcp}^{12}(j', k')$ with $j' = ((j + 1) - 1)/3$, $k' = (n + (k + 1) - 2)/3$. An additional complication is that, unlike in the first case, $j + 1$ and $k + 1$ may not be adjacent in SA, and consequently, j' and k' may not be adjacent in SA^{12} . Thus we have to compute ℓ by performing a range minimum query in LCP^{12} instead of a direct lookup. However, this is still constant time.

Theorem 3. *The extended skew algorithm computing both SA and LCP can be implemented to run in linear time.*

To obtain the same extension for other models of computation, we need to show how to answer $\mathcal{O}(n)$ range minimum queries on LCP¹². We can take advantage of the balanced distribution of the range minimum queries shown by the following property.

Lemma 1. *No suffix is involved in more than two lcp queries at the top level of the extended skew algorithm.*

Proof. Let S_i and S_j be two suffixes whose lcp $\text{lcp}(i, j)$ is computed to find the lcp of the suffixes S_{i-1} and S_{j-1} . (The other case that $\text{lcp}(i, j)$ is needed for the lcp of S_{i-2} and S_{j-2} is similar.) Then S_{i-1} and S_{j-1} are lexicographically adjacent suffixes and $s[i-1] = s[j-1]$. Thus, there cannot be another suffix S_k , $S_i < S_k < S_j$, with $s[k-1] = s[i-1]$. This shows that a suffix can be involved in lcp queries only with its two lexicographically nearest neighbors that have the same preceding character. $\square$

We describe a simple algorithm for answering the range minimum queries that can be easily adapted to the models of Theorem 2. It is based on the ideas in [3,4] (which are themselves based on earlier results).

The LCP¹² array is divided into blocks of size $\log n$. For each block $[a, b]$, precompute and store the following data:

- For all $i \in [a, b]$, a $\log n$ -bit vector Q_i that identifies all $j \in [a, i]$ such that $\text{LCP}^{12}[j] < \min_{k \in [j+1, i]} \text{LCP}^{12}[k]$.
- For all $i \in [a, b]$, the minimum values over the ranges $[a, i]$ and $[i, b]$.
- The minimum for all ranges that end just before or begin just after $[a, b]$ and contain exactly a power of two full blocks.

If a range $[i, j]$ is completely inside a block, its minimum can be found with the help of Q_j in constant time (see [3] for details). Otherwise, $[i, j]$ can be covered with at most four of the ranges whose minimum is stored, and its minimum is the smallest of those minima.

Theorem 4. *The extended skew algorithm computing both SA and LCP can be implemented to achieve the complexities of Theorem 2.*

Proof. (Outline) **External Memory and Cache Oblivious:** The range minimum algorithm can be implemented with sorting and scanning.

Parallel models: The blocks in the range minima data structure are distributed over the processors in the obvious way. Preprocessing range minima data structures reduces to local operations and a straightforward computation proceeding from shorter to longer ranges. Lemma 1 ensures that queries are evenly balanced over the data structure. $\square$

5 Discussion

The skew algorithm is a simple and asymptotically efficient direct algorithm for suffix array construction that is easy to adapt to various models of computation. We expect that it is a good starting point for actual implementations, in particular on parallel machines and for external memory.

The key to the algorithm is the use of suffixes S_i with $i \bmod 3 \in \{1, 2\}$ in the first, recursive step, which enables simple merging in the third step. There are other choices of suffixes that would work. An interesting possibility, for example, is to take suffixes S_i with $i \bmod 7 \in \{3, 5, 6\}$. Some adjustments to the algorithm are required (sorting the remaining suffixes in multiple groups and performing a multiway merge in the third step) but the main ideas still work. In general, a suitable choice is a periodic set of positions according to a *difference cover*. A difference cover D modulo v is a set of integers in the range $[0, v)$ such that, for all $i \in [0, v)$, there exist $j, k \in D$ such that $i \equiv k - j \pmod{v}$. For example $\{1, 2\}$ is a difference cover modulo 3 and $\{3, 5, 6\}$ is a difference cover modulo 7, but $\{1\}$ is not a difference cover modulo 2. Any nontrivial difference cover modulo a constant could be used to obtain a linear time algorithm. Difference covers and their properties play a more central role in the suffix array construction algorithm in [5], which runs in $\mathcal{O}(n \log n)$ time using sublinear extra space in addition to the string and the suffix array.

An interesting theoretical question is whether there are faster CRCW-PRAM algorithms for direct suffix array construction. For example, there are very fast algorithms for padded sorting, list sorting and approximate prefix sums [22] that could be used for sorting and lexicographic naming in the recursive calls. The result would be some kind of suffix list or padded suffix array that could be converted into a suffix array in logarithmic time.

References

1. M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. The enhanced suffix array and its applications to genome analysis. In *Proc. 2nd Workshop on Algorithms in Bioinformatics*, volume 2452 of *LNCS*, pages 449–463. Springer, 2002.
2. M. I. Abouelhoda, E. Ohlebusch, and S. Kurtz. Optimal exact string matching based on suffix arrays. In *Proc. 9th Symposium on String Processing and Information Retrieval*, volume 2476 of *LNCS*, pages 31–43. Springer, 2002.
3. S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors: A survey and a new distributed algorithm. In *Proc. 14th Annual Symposium on Parallel Algorithms and Architectures*, pages 258–264. ACM, 2002.
4. M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proc. 4th Latin American Symposium on Theoretical Informatics*, volume 1776 of *LNCS*, pages 88–94. Springer, 2000.
5. S. Burkhardt and J. Kärkkäinen. Fast lightweight suffix array construction and checking. In *Proc. 14th Annual Symposium on Combinatorial Pattern Matching*. Springer, June 2003. To appear.
6. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, SRC (digital, Palo Alto), May 1994.

7. A. Chan and F. Dehne. A note on coarse grained parallel integer sorting. *Parallel Processing Letters*, 9(4):533–538, 1999.
8. R. Cole. Parallel merge sort. *SIAM J. Comput.*, 17(4):770–785, 1988.
9. A. Crauser and P. Ferragina. Theoretical and experimental study on the construction of suffix arrays in external memory. *Algorithmica*, 32(1):1–35, 2002.
10. R. Dementiev and P. Sanders. Asynchronous parallel disk sorting. In *Proc. 15th Annual Symposium on Parallelism in Algorithms and Architectures*. ACM, 2003. To appear.
11. M. Farach. Optimal suffix tree construction with large alphabets. In *Proc. 38th Annual Symposium on Foundations of Computer Science*, pages 137–143. IEEE, 1997.
12. M. Farach, P. Ferragina, and S. Muthukrishnan. Overcoming the memory bottleneck in suffix tree construction. In *Proc. 39th Annual Symposium on Foundations of Computer Science*, pages 174–183. IEEE, 1998.
13. M. Farach and S. Muthukrishnan. Optimal logarithmic time randomized suffix tree construction. In *Proc. 23th International Conference on Automata, Languages and Programming*, pages 550–561. IEEE, 1996.
14. M. Farach-Colton, P. Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000.
15. M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Annual Symposium on Foundations of Computer Science*, pages 285–298. IEEE, 1999.
16. N. Futamura, S. Aluru, and S. Kurtz. Parallel suffix sorting. In *Proc. 9th International Conference on Advanced Computing and Communications*, pages 76–81. Tata McGraw-Hill, 2001.
17. A. V. Gerbessiotis and C. J. Siniolakis. Merging on the BSP model. *Parallel Computing*, 27:809–822, 2001.
18. G. Gonnet, R. Baeza-Yates, and T. Snider. New indices for text: PAT trees and PAT arrays. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992.
19. M. T. Goodrich. Communication-efficient parallel sorting. *SIAM J. Comput.*, 29(2):416–432, 1999.
20. R. Grossi and G. F. Italiano. Suffix trees and their applications in string algorithms. Rapporto di Ricerca CS-96-14, Università “Ca’ Foscari” di Venezia, Italy, 1996.
21. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
22. T. Hagerup and R. Raman. Waste makes haste: Tight bounds for loose parallel sorting. In *Proc. 33rd Annual Symposium on Foundations of Computer Science*, pages 628–637. IEEE, 1992.
23. T. Hagerup and C. Rüb. Optimal merging and sorting on the EREW-PRAM. *Information Processing Letters*, 33:181–185, 1989.
24. D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13:338–355, 1984.
25. J. Jája. *An Introduction to Parallel Algorithms*. Addison Wesley, 1992.
26. J. Kärkkäinen. Suffix cactus: A cross between suffix tree and suffix array. In Z. Galil and E. Ukkonen, editors, *Proc. 6th Annual Symposium on Combinatorial Pattern Matching*, volume 937 of *LNCS*, pages 191–204. Springer, 1995.
27. T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proc. 12th Annual Symposium on Combinatorial Pattern Matching*, volume 2089 of *LNCS*, pages 181–192. Springer, 2001.

28. D. K. Kim, J. S. Sim, H. Park, and K. Park. Linear-time construction of suffix arrays. In *Proc. 14th Annual Symposium on Combinatorial Pattern Matching*. Springer, June 2003. To appear.
29. P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. In *Proc. 14th Annual Symposium on Combinatorial Pattern Matching*. Springer, June 2003. To appear.
30. N. J. Larsson and K. Sadakane. Faster suffix sorting. Technical report LU-CS-TR:99-214, Dept. of Computer Science, Lund University, Sweden, 1999.
31. U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, Oct. 1993.
32. E. M. McCreight. A space-economic suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.
33. M. H. Nodine and J. S. Vitter. Deterministic distribution sort in shared and distributed memory multiprocessors. In *Proc. 5th Annual Symposium on Parallel Algorithms and Architectures*, pages 120–129. ACM, 1993.
34. M. H. Nodine and J. S. Vitter. Greed sort: An optimal sorting algorithm for multiple disks. *J. ACM*, 42(4):919–933, 1995.
35. S. Rajasekaran and J. H. Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM J. Comput.*, 18(3):594–607, 1989.
36. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
37. L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 22(8):103–111, Aug. 1990.
38. J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory, I: Two level memories. *Algorithmica*, 12(2/3):110–147, 1994.
39. P. Weiner. Linear pattern matching algorithm. In *Proc. 14th Symposium on Switching and Automata Theory*, pages 1–11. IEEE, 1973.

A Source Code

The following C++ file contains a complete linear time implementation of suffix array construction. This code strives for conciseness rather than for speed — it has only 50 lines not counting comments, empty lines, and lines with a bracket only. A driver program can be found at

<http://www.mpi-sb.mpg.de/~sanders/programs/suffix/>.

```
inline bool leq(int a1, int a2, int b1, int b2) // lexicographic order
{ return(a1 < b1 || a1 == b1 && a2 <= b2); } // for pairs
inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3)
{ return(a1 < b1 || a1 == b1 && leq(a2,a3, b2,b3)); } // and triples

// stably sort a[0..n-1] to b[0..n-1] with keys in 0..K from r
static void radixPass(int* a, int* b, int* r, int n, int K)
{ // count occurrences
  int* c = new int[K + 1]; // counter array
  for (int i = 0; i <= K; i++) c[i] = 0; // reset counters
  for (int i = 0; i < n; i++) c[r[a[i]]]++; // count occurrences
  for (int i = 0, sum = 0; i <= K; i++) // exclusive prefix sums
  { int t = c[i]; c[i] = sum; sum += t; }
```

```

    for (int i = 0; i < n; i++) b[c[r[a[i]]]++] = a[i];        // sort
    delete [] c;
}

// find the suffix array SA of s[0..n-1] in {1..K}^n
// require s[n]=s[n+1]=s[n+2]=0, n>=2
void suffixArray(int* s, int* SA, int n, int K) {
    int n0=(n+2)/3, n1=(n+1)/3, n2=n/3, n02=n0+n2;
    int* s12 = new int[n02 + 3]; s12[n02]= s12[n02+1]= s12[n02+2]=0;
    int* SA12 = new int[n02 + 3]; SA12[n02]=SA12[n02+1]=SA12[n02+2]=0;
    int* s0 = new int[n0];
    int* SA0 = new int[n0];

    // generate positions of mod 1 and mod 2 suffixes
    // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1
    for (int i=0, j=0; i < n+(n0-n1); i++) if (i%3 != 0) s12[j++] = i;

    // lsb radix sort the mod 1 and mod 2 triples
    radixPass(s12, SA12, s+2, n02, K);
    radixPass(SA12, s12, s+1, n02, K);
    radixPass(s12, SA12, s, n02, K);

    // find lexicographic names of triples
    int name = 0, c0 = -1, c1 = -1, c2 = -1;
    for (int i = 0; i < n02; i++) {
        if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2)
            { name++; c0 = s[SA12[i]]; c1 = s[SA12[i]+1]; c2 = s[SA12[i]+2]; }
        if (SA12[i] % 3 == 1) { s12[SA12[i]/3] = name; } // left half
        else { s12[SA12[i]/3 + n0] = name; } // right half
    }

    // recurse if names are not yet unique
    if (name < n02) {
        suffixArray(s12, SA12, n02, name);
        // store unique names in s12 using the suffix array
        for (int i = 0; i < n02; i++) s12[SA12[i]] = i + 1;
    } else // generate the suffix array of s12 directly
        for (int i = 0; i < n02; i++) SA12[s12[i] - 1] = i;

    // stably sort the mod 0 suffixes from SA12 by their first character
    for (int i=0, j=0; i < n02; i++) if (SA12[i] < n0) s0[j++] = 3*SA12[i];
    radixPass(s0, SA0, s, n0, K);

    // merge sorted SA0 suffixes and sorted SA12 suffixes
    for (int p=0, t=n0-n1, k=0; k < n; k++) {
#define GetI() (SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2)
        int i = GetI(); // pos of current offset 12 suffix
        int j = SA0[p]; // pos of current offset 0 suffix
        if (SA12[t] < n0 ? // different compares for mod 1 and mod 2 suffixes
            leq(s[i], s12[SA12[t] + n0], s[j], s12[j/3]) :

```



```

        leq(s[i],s[i+1],s12[SA12[t]-n0+1], s[j],s[j+1],s12[j/3+n0]))
    {
        // suffix from SA12 is smaller
        SA[k] = i;  t++;
        if (t == n02) // done --- only SA0 suffixes left
            for (k++; p < n0; p++, k++) SA[k] = SA0[p];
    } else { // suffix from SA0 is smaller
        SA[k] = j;  p++;
        if (p == n0) // done --- only SA12 suffixes left
            for (k++; t < n02; t++, k++) SA[k] = GetI();
    }
}
delete [] s12; delete [] SA12; delete [] SA0; delete [] s0;
}

```

Expansion Postponement via Cut Elimination in Sequent Calculi for Pure Type Systems*

Francisco Gutiérrez and Blas Ruiz

Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga. Campus Teatinos 29071, Málaga. Spain
{pacog, blas}@lcc.uma.es

Abstract. The sequent calculus used in this paper is interesting because (1) it is equivalent to the standard formulation (*natural*) for Pure Type System (PTS), and (2) the corresponding cut-free subsystem makes it possible to introduce a notion of Cut Elimination (*CE*). This property has a deep impact on PTS and in logical frameworks based in PTS. *CE* is an open problem for normalizing generic PTS. Likewise, other proposed versions of cut elimination have not been solved in dependent type systems.

Another interesting problem is *Expansion Postponement (EP)*, posed by Henk Barendregt in August 1990. Except for PTS with important restrictions, *EP* is thus far an open problem, even for normalizing PTS. Surprisingly, in this paper we prove that *EP* is a consequence of *CE*.

Keywords: pure type systems, sequent calculi, cut elimination, expansion postponement.

Track: B.

1 Introduction

Pure Type Systems (PTSs) [1,2] provide a flexible and general framework to study dependent type system properties. These systems are the basis for logical frameworks and proof-assistants that heavily use dependent types [3,4].

In this paper we use the sequent calculi for PTS introduced in [5]. These sequent calculi are influenced by the correspondence between Gentzen's natural deduction *NJ* and the sequent calculus *LJ* for intuitionistic logics [6]. Recall that the natural system *NJ* uses rules to eliminate the connectives ($\rightarrow$, $\vee$, $\wedge$) on the right. An example of such rules is the rule ($\rightarrow E$) (or modus ponens). In the sequent calculus *LJ*, there is no rule that eliminates the connectives on the right; the rule ($\rightarrow E$) is replaced by the rules (*cut*) and ($\rightarrow L$):

$$(cut) \frac{\Gamma \vdash D \quad \Gamma, D \vdash C}{\Gamma \vdash C}, \quad (\rightarrow L) \frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \rightarrow B \vdash C}.$$

The standard (or natural) notion of derivation $\Gamma \vdash a : A$ for PTS is defined by the inductive system shown in Fig.1. By modifying the rules in Fig.1 we can

* This research was partially supported by the project TIC2001-2705-C03-02.

obtain different systems. The standard PTS will be denoted by $\mathcal{N}$. In order to obtain a sequent calculus from the natural type inference relation, the (apl) rule or Π elimination rule

$$(apl) \frac{\Gamma \vdash f : \Pi x : A. F \quad \Gamma \vdash a : A}{\Gamma \vdash f a : F[x := a]}$$

has to be dispensed with, since it eliminates the Π constructor. Recall that Π is a generalization of the connective $\rightarrow$ for dependent types, and the (apl) rule corresponds to modus ponens. Influenced by the Howard-Curry-de Bruijn correspondence [7], an adaptation of Gentzen's $(\rightarrow L)$ rule will be used instead. In particular, we consider an adaptation of the left rule used by Kleene [8](page 481) in the G_3 system:

$$\frac{A \rightarrow B, \Gamma \vdash A \quad A \rightarrow B, B, \Gamma \vdash C}{A \rightarrow B, \Gamma \vdash C}.$$

Hence, we consider the rules:

$$\begin{aligned} (\mathcal{K}) \quad & \frac{\Gamma \vdash a : A \quad \Gamma, x : S, \Delta \vdash c : C}{\Gamma, \Delta[x := ya] \vdash c[x := ya] : C[x := ya]} \quad \left\{ \begin{array}{l} y : \Pi z : A. B \in \Gamma, \\ S =_{\beta} B[z := a], \end{array} \right. \\ (cut) \quad & \frac{\Gamma \vdash d : D \quad \Gamma, y : D \vdash c : C}{\Gamma \vdash c[y := d] : C[y := d]}. \end{aligned}$$

$\mathcal{K}$ (for Kleene) denotes the system obtained by replacing the (apl) rule in the original system $\mathcal{N}$ (see Fig.1) by the $(\mathcal{K})$ and (cut) rules. Similarly, $\mathcal{K}^{cf}$ ($\mathcal{K}$ -cutfree) denotes the system obtained by eliminating the (cut) rule.

The $\mathcal{K}$ system is equivalent to the natural system $\mathcal{N}$, and obviously, $\mathcal{K}^{cf}$ is also correct. A notion of completeness of $\mathcal{K}^{cf}$ generates the cut elimination property.

Recall Gentzen's *hauptsatz*: every *LJ* derivation can be obtained without using the cut rule, which is known as cut elimination. This result is an essential technique in proof theory [9]. Likewise, a similar notion of Cut Elimination (CE) can be formulated: every $\mathcal{K}$ *normalized* derivation can be obtained without using the cut rule. CE will have a deep impact on PTS. Thus, CE can be applied to develop proof-search strategies with dependent types, similar to those proposed in [10,11,3].

In [12] we prove that *CE* is equivalent to the admissibility of a rule to type applications in the system $\mathcal{K}^{cf}$. As a result, *CE* is obtained in two important families of systems.

CE is an open problem for generic normalized systems. This is not surprising, and in the present paper we prove that *CE* is actually sufficient to prove the *Expansion Postponement (EP)* problem [13] posed by Henk Barendregt in August 1990. If we consider the $\vdash_r$ system obtained when the (β) rule is substituted by the predicate β -reduction rule

$$\frac{\Gamma \vdash a : A \quad A \twoheadrightarrow_{\beta} A'}{\Gamma \vdash a : A'},$$

then EP turns into the following conjecture: any judgement $\Gamma \vdash a : A$ can be obtained by first deriving $\Gamma \vdash_r a : A'$, for some redex $A' \xleftarrow{\beta} A$, and then possibly by applying β -expansion.

The relevance of EP stems from its application to the correctness proof of certain type checking systems ([14,13,15]). Bear in mind that except for PTS with important restrictions, EP is thus far an open problem, even for normalizing PTS [16].

Robert Pollack exposes in [17] the relation between EP and the problem of finding reasonable algorithms for type checking in PTS. In this sense, he proposes different ways to represent PTS in order to derive directly type checking algorithms: the syntax-directed systems.

[13] emphasizes that EP is a necessary condition for the completeness of the syntax-directed type checking algorithms proposed by Pollack, and therefore it is possible to find complete algorithms only for systems enjoying EP . Similarly, [14] also conjectures that the completeness of the algorithm is essentially equivalent to EP .

It is well-known that EP can be solved by the completeness of a certain system $\mathcal{N}_n$ that infers normal types only [13,18]. We will see that CE ensures that $\mathcal{K}$ is correct with respect to $\mathcal{N}_n$, and therefore EP is easily obtained.

The paper is organized as follows: in Section 2 we briefly describe PTS; in Section 3 we define the sequent calculus; Section 4 and Section 5 introduce cut elimination and expansion postponement properties; in Section 6 we prove the relation between cut elimination and expansion postponement, and finally, we present some conclusions and related works.

2 Pure Type Systems

In this section we review PTS and their main properties. For further details the reader is referred to [1,2,19,18].

Considering an infinite set of variables $\mathcal{V}$ ($x, y, \dots \in \mathcal{V}$) and a set of constants or *sorts* $\mathcal{S}$ ($s, s_1, \dots \in \mathcal{S}$), the set $\mathcal{T}$ of terms for a PTS is inductively defined as:

$$\begin{aligned} a \in \mathcal{V} \cup \mathcal{S} &\Rightarrow a \in \mathcal{T}, \\ A, C, a, b \in \mathcal{T} &\Rightarrow a\ b, \lambda x : A. b, \Pi x : A. C \in \mathcal{T}. \end{aligned}$$

A PTS is defined by its specification, that is, a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, where $\mathcal{A} \subseteq \mathcal{S}^2$ is a set of *axioms*, and $\mathcal{R} \subseteq \mathcal{S}^3$ a set of *rules*. Instances of this tuple embed important theories, such as $\lambda 2$, F^ω , and the Calculus of Constructions. The standard (or natural) notion of derivation $\Gamma \vdash a : A$ is defined by the inductive system shown in Fig.1, and the standard corresponding PTS will be denoted by $\mathcal{N}$.

We denote the β -reduction as $\rightarrow_\beta$ and the equality generated by $\rightarrow_\beta$ as $=_\beta$. The set of β -normal forms is denoted $\beta\text{-nf}$ and a_β denotes the β -normal form of a ; $\text{FV}(a)$ denotes the set of free variables. $A[x := B]$ denotes, as usual, substitution. A *context* Γ is a sequence (possibly empty) $\langle x_1 : A_1, \dots, x_n : A_n \rangle$ of declarations $x_i : A_i$ where $x_i \in \mathcal{V}$ and $A_i \in \mathcal{T}$. We drop the $\langle \rangle$ symbols when there is no

ambiguity. We write $x_i : A_i \in \Gamma$ when the declaration $x_i : A_i$ is in Γ , and by using the $(\doteq)$ symbol to establish definitions, we have

$$\begin{aligned} \Gamma \subseteq \Gamma' &\doteq & \forall x \in \mathcal{V} [x : A \in \Gamma \Rightarrow x : A \in \Gamma'], \\ \text{Var}(\Gamma) &\doteq & \{x_1, \dots, x_n\}, \\ \text{FV}(\Gamma) &\doteq & \text{FV}(A_1) \cup \dots \cup \text{FV}(A_n). \end{aligned}$$

We can extend the β -reduction to contexts and therefore define the β -normal form for contexts:

$$\langle \rangle_\beta \doteq \langle \rangle, \quad \langle x : A, \Gamma \rangle_\beta \doteq \langle x : A_\beta, \Gamma_\beta \rangle.$$

We say that Γ is a legal context (denoted with $\Gamma \vdash$) if $\exists c, C [\Gamma \vdash c : C]$. We recall elementary properties of PTS:

Lemma 1 (Elementary Properties) *If $\Gamma \vdash c : C$, then:*

- (i) $\text{FV}(c : C) \subseteq \text{Var}(\Gamma)$, and if $x_i, x_j \in \text{Var}(\Gamma)$, then $i \neq j \Rightarrow x_i \neq x_j$. (*FrVrs*)
- (ii) $s_1 : s_2 \in \mathcal{A} \Rightarrow \Gamma \vdash s_1 : s_2$. (*TypAx*)
- (iii) $y : D \in \Gamma \Rightarrow \Gamma \vdash y : D$. (*TypVr*)
- (iv) *Type correctness*: $\Gamma \vdash c : C \wedge C \notin \mathcal{S} \Rightarrow \exists s \in \mathcal{S} [\Gamma \vdash C : s]$. (*CrTyps*)
- (v) *Context correctness*: $\Gamma, x : A, \Delta \vdash d : D \Rightarrow \exists s \in \mathcal{S} [\Gamma \vdash A : s]$. (*CrCtx*)

We also need typical properties of PTS: subject β -reduction ($S\beta$), predicate β -reduction ($P\beta$), substitution lemma (Sbs), and thinning lemma ($Thnng$):

$$\begin{aligned} &\frac{\Gamma \vdash a : A \quad a \rightarrow_\beta a'}{\Gamma \vdash a' : A} (S\beta), & \frac{\Gamma \vdash a : A \quad A \rightarrow_\beta A'}{\Gamma \vdash a : A'} (P\beta), \\ &\frac{\Gamma \vdash d : D \quad \Gamma, y : D, \Delta \vdash c : C}{\Gamma, \Delta[y := d] \vdash c[y := d] : C[y := d]} (Sbs), & \frac{\Gamma \vdash b : B \quad \Gamma \subseteq \Psi \vdash}{\Psi \vdash b : B} (Thnng). \end{aligned}$$

Let us recall that the natural system $\mathcal{N}$ satisfies a generation lemma (see Lemma 19 in [1]).

In this paper, every free *object* in the right hand side of an implication or in the conclusion of a rule is existentially quantified. For example, the *CrCtx* property of Lemma 1 can be enunciated as: $\Gamma, x : A, \Delta \vdash d : D \Rightarrow \Gamma \vdash A : s$.

The lemma below is rarely referred to in the literature; however, it will be used here to simplify some proofs. This lemma characterize the set of types for every term.

Lemma 2 (The Shape of Types) (van Benthem Jutting [19]) *The set of terms of a PTS can be divided in two disjoint classes T_v and T_s , inductively defined as:*

$$\begin{aligned} x \in T_v & & b \in T_v &\Rightarrow b c, \lambda x : A. b \in T_v, \\ s, \Pi x : A. B \in T_s & & b \in T_s &\Rightarrow b c, \lambda x : A. b \in T_s, \end{aligned}$$

so that

$$\Gamma \vdash a : A, a : A' \Rightarrow \begin{cases} a \in T_v \Rightarrow A =_\beta A', \\ a \in T_s \Rightarrow A \rightarrow_\beta \Pi \Delta. s \wedge A' \rightarrow_\beta \Pi \Delta. s', \end{cases}$$

where $\Pi \langle \rangle. M \doteq M$ and $\Pi \langle x : X, \Delta \rangle. M \doteq \Pi x : X. (\Pi \Delta. M)$.

(ax)	$\frac{}{\vdash s_1 : s_2}$	$s_1 : s_2 \in \mathcal{A}$
(var)	$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	$x \notin \text{Var}(\Gamma)$
$(weak)$	$\frac{\Gamma \vdash b : B \quad \Gamma \vdash A : s}{\Gamma, x : A \vdash b : B}$	$b \in \mathcal{S} \cup \mathcal{V}, x \notin \text{Var}(\Gamma)$
(Π)	$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3}$	$(s_1, s_2, s_3) \in \mathcal{R}$
(apl)	$\frac{\Gamma \vdash f : \Pi x : A. F \quad \Gamma \vdash a : A}{\Gamma \vdash f a : F[x := a]}$	
(λ)	$\frac{\Gamma \vdash \Pi x : A. B : s \quad \Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x : A. b : \Pi x : A. B}$	
(β)	$\frac{\Gamma \vdash a : A \quad \Gamma \vdash A' : s}{\Gamma \vdash a : A'}$	$A =_\beta A'$

Fig. 1. Inference rules for PTS. For the sake of readability, $s_1 : s_2 \in \mathcal{A}$ stands for $(s_1, s_2) \in \mathcal{A}$.

3 Sequent Calculi for PTS

In order to obtain a sequent calculus from the natural type inference relation, the (apl) rule or Π elimination rule has to be dispensed with, since it eliminates the Π constructor.

Definition 3 1. We consider the rules:

$$\begin{aligned}
 (\mathcal{K}) \quad & \frac{\Gamma \vdash a : A \quad \Gamma, x : S, \Delta \vdash c : C}{\Gamma, \Delta[x := y a] \vdash c[x := y a] : C[x := y a]} \quad \left\{ \begin{array}{l} y : \Pi z : A. B \in \Gamma, \\ S =_\beta B[z := a], \end{array} \right. \\
 (cut) \quad & \frac{\Gamma \vdash d : D \quad \Gamma, y : D \vdash c : C}{\Gamma \vdash c[y := d] : C[y := d]}.
 \end{aligned}$$

2. $\mathcal{K}$ denotes the systems obtained by replacing the (apl) rule in the original system $\mathcal{N}$ (see Fig.1) by the $(\mathcal{K})$ and (cut) rules. The type inference relation of $\mathcal{K}$ will be denoted as $\vdash_{\mathcal{K}}$.
3. Similarly, $\mathcal{K}^{\text{cf}}$ denotes the systems obtained by eliminating the (cut) rule. Its type inference relation will be denoted as $\vdash_{\mathcal{K}^{\text{cf}}}$.

Like PTS, $\mathcal{K}$ and $\mathcal{K}^{\text{cf}}$ denote many systems depending on the $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ specification. Elementary properties of PTS hold for sequent calculi as well.

Lemma 4 *Lemma 1 holds for $\mathcal{K}$ and $\mathcal{K}^{\text{cf}}$ systems.*

Proof See Lemma 5 in [5]. ■

Theorem 5 (Correctness and Completeness of Sequent Calculus)

$\mathcal{N} \equiv \mathcal{K}$.

Proof See [5]. ■

Because of the form of the $(\mathcal{K})$ rule, every object (subject, context, and type) in each derivation in $\vdash_{\mathcal{K}^{\text{cf}}}$ is in β -normal form. In fact,

Lemma 6 (The Shape of Types in Cut Free Sequent Calculi) *In every $\mathcal{K}^{\text{cf}}$ system, we have that:*

- (i) $\Gamma \vdash_{\mathcal{K}^{\text{cf}}} m : M \Rightarrow \Gamma, m, M \in \beta\text{-nf}$.
- (ii) $a \in T_v \wedge \Gamma \vdash_{\mathcal{K}^{\text{cf}}} a : A, a : A' \Rightarrow A \equiv A'$.
- (iii) $a \in T_s \wedge \Gamma \vdash_{\mathcal{K}^{\text{cf}}} a : A, a : A' \Rightarrow A \equiv \Pi \Delta.s \wedge A' \equiv \Pi \Delta.s'$.

Proof (i) it follows by IDs using the fact that the $[x := y a]$ operator preserves normal forms when $a \in \beta\text{-nf}$. In order to prove (ii) – (iii), it suffices to apply Theorem 5 and then Lemma 2 and (i). ■

Corollary 7 $\Gamma \vdash_{\mathcal{K}^{\text{cf}}} B : s \wedge \Gamma \vdash B : s' \Rightarrow \Gamma \vdash_{\mathcal{K}^{\text{cf}}} B : s' \quad s, s' \in \mathcal{S}$.

Proof. By induction on the structure of B . By the generation lemma, B cannot be an abstraction. If it is a constant, then again by the generation lemma in $\vdash$ we have $B : s' \in \mathcal{A}$ and apply *TypAx* in $\mathcal{K}^{\text{cf}}$. If $B \in T_v$, we apply Lemma 6(ii) to get $s \equiv s'$. Should B be an application, it must have the form $y f_1 \dots f_n$ that is in T_v , and the previous reasoning is applied again. If $B \equiv \Pi t : M.N$, we apply the generation lemma in both systems, followed by IH twice and (Π) . ■

4 Cut Elimination

The notion of cut elimination for PTS is strongly influenced by the presence of the rule of β -conversion of types; therefore the system $\mathcal{K}$ can type objects (types, contexts, and terms) in not β -normal form. But from Lemma 6 we obtain that $\mathcal{K}^{\text{cf}}$ yields objects in β -normal form. Therefore, in $\mathcal{K}^{\text{cf}}$ system, we can dispense with the (β) rule since it does not yield different types. In a condensed form, *Cut Elimination* is enunciated as:

$$\Gamma, m, M \in \beta\text{-nf} \wedge \Gamma \vdash_{\mathcal{K}} m : M \Rightarrow \Gamma \vdash_{\mathcal{K}^{\text{cf}}} m : M. \quad (CE)$$

This is the central property of the $\mathcal{K}$ sequent calculus.

By Theorem 5, $\vdash_{\mathcal{K}}$ can be taken as the standard relation (Fig.1). To prove *CE*, if we proceed by ID of $\Gamma \vdash m : M$ and the last rule applied is β -conversion, then IH cannot be applied on the premises since their types are not necessarily in β -normal form. We can then reformulate *CE* in the following equivalent way:

$$\Gamma, m \in \beta\text{-nf} \wedge \Gamma \vdash m : M \Rightarrow \Gamma \vdash_{\mathcal{K}^{\text{cf}}} m : M_{\beta}. \quad (CE)$$

(where M_β denotes the β -normal form of M). However, the property of normalization must be imposed on the system¹. Under these considerations, the problem above is avoided but a new problem arises when the last rule applied is the *(apl)* rule. Therefore, a new rule for typing applications in β -normal form is needed in the $\mathcal{K}^{\text{cf}}$ system.

Lemma 8 *If $\mathcal{K}$ is normalizing then CE is equivalent to the admissibility of the rule:*

$$\frac{\Gamma \vdash_{\mathcal{K}^{\text{cf}}} a : A \quad \Gamma \vdash_{\mathcal{K}^{\text{cf}}} f : \Pi z : A. B}{\Gamma \vdash_{\mathcal{K}^{\text{cf}}} f a : B[z := a]_\beta} f a \in \beta\text{-nf}. \quad (\text{AplN})$$

Proof. ($\Rightarrow$) It follows from $\vdash_{\mathcal{K}^{\text{cf}}} \subseteq \vdash$, *(apl)*, and *CE*.

($\Leftarrow$) Assume *AplN*. Then we prove that *CE* by ID $\Gamma \vdash m : M$. Only two cases are shown. When the last rule applied is *(apl)*, we apply IH twice and then the *AplN* property. When the last rule applied is

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad \Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x : A. b : \Pi x : A. B} \quad (s_1, s_2, s_3) \in \mathcal{R},$$

by IH we have that $\Gamma \vdash_{\mathcal{K}^{\text{cf}}} A : s_1$ and $\Gamma, x : A \vdash_{\mathcal{K}^{\text{cf}}} b : B_\beta$, and then we have to prove that $\Gamma, x : A \vdash_{\mathcal{K}^{\text{cf}}} B_\beta : s_2$. We apply correctness of types in $\vdash_{\mathcal{K}^{\text{cf}}}$ to the last derivation:

- $B_\beta \equiv s \in \mathcal{S}$. Then, since $\Gamma, x : A \vdash s : s_2$, we have that $B_\beta : s_2 \in \mathcal{A}$, and then by Lemma 4(ii) (*TypAx*) we get $\Gamma, x : A \vdash_{\text{cf}} B_\beta : s_2$.
- $\Gamma, x : A \vdash_{\text{cf}} B_\beta : s$. By the second premise and *S β* we obtain $\Gamma, x : A \vdash B_\beta : s_2$; then we apply Corollary 7. ■

5 Expansion Postponement

If we consider the $\vdash_r$ system obtained when the rule *(β)* is substituted by the predicate β -reduction rule (*P β* , see Section 2), then *Expansion Postponement* (*EP*) turns into the following conjecture: any judgement $\Gamma \vdash a : A$ can be obtained by first deriving $\Gamma \vdash_r a : A'$, for some redex A' , and then possibly by applying β -expansion. Therefore *EP* is characterized by the following property:

$$\Gamma \vdash a : A \quad \Rightarrow \quad \Gamma \vdash_r a : A' \wedge A \rightarrow_\beta A'. \quad (\text{EP})$$

The *EP* formulation motivates the definition of the following reflexive and transitive relation $\sqsubseteq$:

$$\vdash_1 \sqsubseteq \vdash_2 \doteq \forall \Gamma, a, A [\Gamma \vdash_1 a : A \Rightarrow \Gamma \vdash_2 a : A' \leftarrow_\beta A].$$

Therefore, the property $\vdash \sqsubseteq \vdash_r$ captures *EP*.

An alternative to analyzing *EP* is to study the normalizing systems with types in β -normal form. In the sequel, we consider normalizing systems and let us then consider the system $\mathcal{N}_n$ obtained by the $\vdash_n$ relation, defined by the *(ax)*,

(var_n)	$\frac{\Gamma \vdash_n A : s}{\Gamma, x : A \vdash_n x : A_\beta} \quad x \notin \Gamma$
(apl_n)	$\frac{\Gamma \vdash_n f : \Pi x : A. F \quad \Gamma \vdash_n a : A}{\Gamma \vdash_n f a : F[x := a]_\beta}$
(λ_n)	$\frac{\Gamma, x : A \vdash_n b : B \quad \Gamma \vdash_n \Pi x : A. B : s}{\Gamma \vdash_n \lambda x : A. b : \Pi x : A. B_\beta}$

Fig. 2. Additional rules for the $\vdash_n$ relation

(*weak*), (*II*) rules (see Fig.1), and the rules of Fig.2. This system is considered in [13]. It is easy to prove by ID that $\Gamma \vdash_n a : A \Rightarrow A \in \beta\text{-nf}$ and that the system is correct: $\vdash_n \subseteq \vdash$. On the other hand, the implication $\Gamma \vdash_n c : C \Rightarrow \Gamma \vdash_r c : C$ is easy by ID. Hence, *EP* is a consequence of $\vdash \sqsubseteq \vdash_n$.

Except for PTS with important restrictions, *EP* and the $\sqsubseteq$ -completeness of $\vdash_n$ are still open problems, but they admit solutions for particular PTS [18,16].

In order to study the implication *CE* $\Rightarrow$ *EP* we shall use two technical lemmas.

Lemma 9 (Semi-Commutation of Substitution and β -Reduction) *Let us assume that the β -normal form always exist. Then*

- (i) $(A_\beta)^\circ_\beta \equiv A^\circ_\beta$.
- (ii) $x \neq y \wedge x \notin \text{FV}(d) \Rightarrow (B^\circ_\beta[x := f^\circ])_\beta \equiv (B[x := f]_\beta)^\circ_\beta$.

where the priority of the operator $^\circ \equiv [y := d]$ is higher than that of $(-)_\beta$.

Proof. (i). It suffices to apply substitutivity of $\rightarrow_\beta$ and Church-Rosser. (ii):

$$\begin{aligned}
& (B^\circ_\beta[x := f^\circ])_\beta \\
& \equiv \because (i) \text{ with } A := B^\circ, \circ := [x := f^\circ] \\
& (B^\circ[x := f^\circ])_\beta \\
& \equiv \because \text{untyped } \lambda\text{-calculus substitution lemma [2] (Lemma 2.1.6):} \\
& \quad x \neq y \quad x \notin \text{FV}(d) \Rightarrow B^\circ[x := f^\circ] \equiv B[x := f]^\circ \\
& (B[x := f]^\circ)_\beta \\
& \equiv \because (i) \text{ with } A := B[x := f] \\
& (B[x := f]_\beta)^\circ_\beta \blacksquare
\end{aligned}$$

Lemma 10 (Context Substitution) *For any PTS*

$$\Gamma, y : Y, \Delta \vdash_n c : C \wedge \Gamma \vdash_n Y' : s \wedge Y' =_\beta Y \Rightarrow \Gamma, y : Y', \Delta \vdash_n c : C,$$

and we have the strong context substitution property:

$$\Gamma \vdash_n a : A \wedge \Gamma' \vdash_n \wedge \Gamma =_\beta \Gamma' \Rightarrow \Gamma' \vdash_n a : A.$$

¹ Recall that a PTS is normalizing if it verifies $\Gamma \vdash a : A \Rightarrow a$ is weak normalizing (also, by type correctness, A is weak normalizing).

Proof By ID of $\psi \equiv \Gamma, y : Y, \Delta \vdash_n c : C$. If ψ has been inferred using a rule whose premise includes the context $\Gamma' \supseteq \Gamma$, it suffices to apply IH and the same rule. Thus, we consider the (var_n) and $(weak)$ rules only. If $\Delta \equiv \langle \rangle$, we apply (var_n) or $(weak)$. The other cases follow by IH and the rule. The *strong context substitution* property follows by induction on the context using the first one. ■

6 Expansion Postponement from Cut Elimination

In this section we prove that CE solves EP .

Lemma 11 (Correctness $\mathcal{K}^{cf}$ w.r.t. $\vdash_n$) *For every normalizing PTS, we have that $\vdash_{\mathcal{K}^{cf}} \subseteq \vdash_n$.*

Proof. In the first place, we prove that the $\vdash_n$ system satisfies the following restriction to the $(\mathcal{K})$ rule:

$$(n\mathcal{K}) \quad \frac{\Gamma \vdash_n a : A \quad \Gamma, x : S, \Delta \vdash_n c : C}{\Gamma, \Delta^\circ \vdash_n c^\circ : C^\circ} \quad \left\{ \begin{array}{l} \Gamma, a, A, S, \Delta, c, C \in \beta\text{-nf}, \\ y : \Pi z : A. B \in \Gamma, \\ S =_\beta B[z := a], \end{array} \right.$$

where $^\circ \equiv [x := ya]$ and obviously Δ°, c° , and C° are in β -normal form too. We reason by ID of $\psi \equiv \Gamma, x : S, \Delta \vdash_n c : C$.

1. $\psi : -(var_n)^2$ with $\Delta \equiv \langle \rangle$, and $x : S \equiv c : C$, with $\Gamma \vdash_n S : s$ and $x \notin S$.
Since Γ is a legal context, we have that $\Gamma \vdash_n y : \Pi z : A. B \in \Gamma$, and by applying the (apl_n) rule we get $\Gamma \vdash_n ya : B[z := a]_\beta (\equiv S \equiv S^\circ)$.
2. $\psi : -(var_n)$, with $\Delta \equiv \Delta_1, u : U$, and (because of $U \in \beta\text{-nf}$), $\Gamma, x : S, \Delta_1 \vdash_n U : s$, with $u : U \equiv c : C$; by IH we have $\Gamma, \Delta_1^\circ \vdash_n U^\circ : s$, and we finally apply the (var_n) rule.
3. $\psi : -(\lambda_n)$; because of $c \in \beta\text{-nf}$, we can assume:

$$\frac{\Gamma, x : S, \Delta \vdash_n \Pi t : P.Q : s \quad \Gamma, x : S, \Delta, t : P \vdash_n q : Q}{\Gamma, x : S, \Delta \vdash_n \lambda t : P.q : \Pi t : P.Q},$$

and then we apply IH on the premises, $P^\circ \in \beta\text{-nf}$, and the (λ_n) rule.

4. $\psi : -(apl_n)$

$$\frac{\Gamma, x : S, \Delta \vdash_n f : \Pi t : G.F, \quad g : G}{\Gamma, x : S, \Delta \vdash_n f g : F[t := g]_\beta (\equiv c : C)}.$$

By applying IH and (apl_n) : $\Gamma, \Delta^\circ \vdash_n f^\circ g^\circ : (F^\circ[t := g^\circ])_\beta$. However, by Lemma 9 and $t \neq x \wedge t \notin \text{FV}(ya)$ we get $(F^\circ_\beta[t := g^\circ])_\beta \equiv (F[t := g]_\beta)^\circ_\beta$, and now it suffices to observe that the $^\circ$ operator preserves normal forms, and hence $(F^\circ[t := g^\circ])_\beta \equiv (F[t := g]_\beta)^\circ$.

5. The remaining cases $(weak, \Pi)$ follow by IH and the corresponding rule.

² $\psi : -(r)$ denotes that the last rule applied is (r) .

Finally, we show that $\Gamma \vdash_{\mathcal{K}^{\text{cf}}} a : A \Rightarrow \Gamma \vdash_n a : A$ by ID. Since the system $\vdash_{\mathcal{K}^{\text{cf}}}$ can only infer objects in β -normal form, we only need to consider the case when the last rule applied is $(\mathcal{K})$. But in this case, we apply IH and the $(n\mathcal{K})$ rule. ■

To end this section, we state and prove the main result of this paper.

Theorem 12 *For every normalizing PTS, we have that $CE \Rightarrow \vdash \sqsubseteq \vdash_n$. Thus EP is a consequence of CE.*

Proof If CE holds, we have $\Gamma \vdash c : C \Rightarrow \Gamma_\beta \vdash_{\mathcal{K}^{\text{cf}}} c_\beta : C_\beta$ and apply Lemma 11 to obtain:

$$\Gamma \vdash a : A \Rightarrow \Gamma_\beta \vdash_n a_\beta : A_\beta.$$

Then, by applying the above property, $\vdash_n \sqsubseteq \vdash$ and context substitution, we have:

$$\Gamma \vdash_n A : s \Rightarrow \Gamma \vdash_n A_\beta : s. \quad (KEY)$$

Now, we shall use the KEY property in order to prove $\Gamma \vdash c : C \Rightarrow \Gamma \vdash_n c : C_\beta$. We proceed by ID. The only interesting case is when the last rule applied is (λ) :

$$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \lambda x : A. b : \Pi x : A. B} \quad (s_1, s_2, s_3) \in \mathcal{R}.$$

Then we proceed in the following way:

$$\frac{\frac{\Gamma, x : A \vdash_n b : B_\beta}{\Gamma, x : A \vdash_n b : B_\beta} \text{IH} \quad \frac{\frac{\Gamma \vdash_n A : s_1}{\Gamma \vdash_n A : s_1} \text{IH} \quad \frac{\frac{\Gamma, x : A \vdash_n B : s_2}{\Gamma, x : A \vdash_n B_\beta : s_2} \text{IH} \quad \text{KEY}}{\Gamma, x : A \vdash_n B_\beta : s_2} \text{KEY}}{\Gamma \vdash_n \Pi x : A. B_\beta : s_3} \text{KEY} \quad (\lambda_n). \blacksquare$$

KEY property can also be proved by subject reduction. In order to prove subject reduction we try

$$\Gamma \vdash_n c : C \Rightarrow \begin{cases} c \rightarrow_\beta c' \Rightarrow \Gamma \vdash_n c' : C, \\ \Gamma \rightarrow_\beta \Gamma' \Rightarrow \Gamma' \vdash_n c : C, \end{cases}$$

by simultaneous ID. For the (apl_n) case, the following substitution lemma ($^\circ \equiv [y := d]$) is required:

$$\Gamma \vdash_n d : D_\beta \wedge \Gamma, y : D, \Delta \vdash_n c : C \Rightarrow \Gamma, \Delta^\circ \vdash_n c^\circ : C^\circ. \quad (1)$$

[13] tries to obtain the substitution lemma (1) directly. Unfortunately their proof is not complete as indicated below. If we reason by induction on the derivation $\Gamma, y : D, \Delta \vdash_n c : C$, except for the (λ_n) case, all of them follow by IH and Lemma 9. The problem appears when the last rule applied is the (λ_n) rule

$$\frac{\Gamma, y : D, \Delta, x : A \vdash_n b : B \quad \Gamma, y : D, \Delta \vdash_n \Pi x : A. B : s}{\Gamma, y : D, \Delta \vdash_n \lambda x : A. b : \Pi x : A_\beta. B}.$$

Then, by IH we get to $\Gamma, \Delta^\circ, x : A^\circ \vdash_n b^\circ : B_\beta^\circ$ and $\Gamma, \Delta^\circ \vdash_n \Pi x : A^\circ.B^\circ : s$. But we can not apply the (λ_n) rule again because we would need $\Gamma, \Delta^\circ \vdash_n \Pi x : A^\circ.B_\beta^\circ : s$. As consequence, *EP* is still an open problem.

Thus $\vdash \sqsubseteq \vdash_n$ is equivalent to the *KEY* property.

7 Conclusions and Related Works

In this paper we have proved that *EP* is a consequence of *CE*. The relevance of *EP* stems from its application to the correctness proof of numerous type checking systems.

Theoretical properties of sequent calculi presented in this paper have been studied in [5], but a general study of *CE* is very difficult due to the proviso $S =_\beta B[z := a]$ in the $(\mathcal{K})$ rule. This situation disappears by replacing $S \equiv B[z := a]$. This change will have a deep impact in the proof of cut elimination. In [12], we study the systems described by this new rule: $\mathcal{K}'$ and $\mathcal{K}'^{\text{cf}}$ (surprisingly, $\mathcal{K}' \equiv \mathcal{K}$, and trivially, $\mathcal{K}'^{\text{cf}} \subseteq \mathcal{K}^{\text{cf}}$). The cut elimination property obtained in these systems is the Strong Cut Elimination (*SCE*), stronger than the one presented in this paper.

While (weak) *CE* is an open problem for generic normalized systems, in [12] we have proven *SCE* (and also, *CE*) in two important families of systems characterized as follows. On the one hand, those PTSs where in every rule $(s_1, s_2, s_3) \in \mathcal{R}$, the constant s_2 does not occur in the right hand side of an axiom. Thus, we obtain proofs of *SCE* in the corners $\lambda \rightarrow$ and $\lambda 2$ of the λ -cube [2]. In addition, we have proven *SCE* for another class of systems, the Π -independent: the well-typed dependent products $\Pi z : A.B$ satisfy $z \notin \text{FV}(B)$. This result yields *SCE* as a simple corollary, and corners $\lambda \rightarrow$ and $\lambda \omega$ of Barendregt's λ -cube are particular cases. A generation lemma for the $\mathcal{K}'^{\text{cf}}$ system makes it possible to refute *SCE* for the remaining systems in the λ -cube, as well as in other interesting systems: $\lambda U, \lambda HOL, \lambda AUT\_QE, \lambda AUT - 68$, and λPAL , all of them described in [2]:216.

In summary, for a wide class of systems, the proof of *CE* is directly deduced from the axioms and rules of PTS, thus providing the proof of *EP* from the specification.

Recently, other authors [10,20] have introduced notions of *CE* for particular systems. Thus, our $(\mathcal{K})$ rule generalizes the rule used by Pym [10] in his proof of *CE* for the $\lambda \Pi$ system, a system with dependent types very similar to the λP PTS. Using Pym's rule, the generation lemma for applications cannot be proved. However, this lemma is essential in both our analysis of *CE* and in the proof of [10]; therefore, *CE* is an open problem in λP , but *SCE* does not hold in λP .

Acknowledgments. The authors are very grateful to Pablo López and the anonymous referees for comments on earlier versions of this paper.

References

1. H. Geuvers, M.-J. Nederhof, Modular proof of Strong Normalization for the Calculus of Constructions, *Journal of Functional Programming* 1 (1991) 15–189.
2. H. P. Barendregt, Lambda Calculi with Types, in: S. Abramsky, D. Gabbay, T. S. Maibaum (Eds.), *Handbook of Logic in Computer Science*, Oxford University Press, 1992, Ch. 2.2, pp. 117–309.
3. F. Pfenning, Logical frameworks, in: A. Robinson, A. Voronkov (Eds.), *Handbook of Automated Reasoning*, Vol. II, Elsevier Science, 2001, Ch. 17, pp. 1063–1147.
4. H. P. Barendregt, H. Geuvers, Proof-assistants using dependent type systems, in: A. Robinson, A. Voronkov (Eds.), *Handbook of Automated Reasoning*, Vol. II, Elsevier Science, 2001, Ch. 18, pp. 1149–1238.
5. F. Gutiérrez, B. C. Ruiz, A Cut Free Sequent Calculus for Pure Type Systems Verifying the Structural Rules of Gentzen/Kleene, in: *International Workshop on Logic Based Program Development and Transformation (LOPSTR'02)*, September 17–20, Madrid, Spain, Vol. (to appear) of LNCS, Springer-Verlag, 2003, <http://polaris.lcc.uma.es/~blas/publicaciones/>.
6. G. Gentzen, Untersuchungen über das Logische Schliessen, *Math. Zeitschrift* 39 (1935) 176–210, 405–431, translation in [21].
7. H. Geuvers, Logics and type systems, Ph.D. thesis, Computer Science Institute, Katholieke Universiteit Nijmegen (1993).
8. S. C. Kleene, *Introduction to Metamathematics*, D. van Nostrand, Princeton, New Jersey, 1952.
9. M. Baaz, A. Leitsch, Cut-elimination and redundancy-elimination by resolution, *Journal of Symbolic Computation* 29 (2) (2000) 149–177.
10. D. Pym, A note on the proof theory of the $\lambda\Pi$ -calculus, *Studia Logica* 54 (1995) 199–230.
11. D. Galmiche, D. J. Pym, Proof-search in type-theoretic languages: an introduction, *Theoretical Computer Science* 232 (1–2) (2000) 5–53.
12. F. Gutiérrez, B. C. Ruiz, Sequent Calculi for Pure Type Systems, Tech. Report 06/02, Dept. de Lenguajes y Ciencias de la Computación, Universidad de Málaga (Spain), <http://polaris.lcc.uma.es/~blas/publicaciones/> (may 2002).
13. E. Poll, Expansion Postponement for Normalising Pure Type Systems, *Journal of Functional Programming* 8 (1) (1998) 89–96.
14. L. van Benthem Jutting, J. McKinna, R. Pollack, Checking Algorithms for Pure Type Systems, in: H. Barendregt, T. Nipkow (Eds.), *Types for Proofs and Programs: International Workshop TYPES'93*, no. 806 in *Lecture Notes in Computer Science*, Springer-Verlag, 1994, pp. 19–61.
15. G. Barthe, Type-checking injective pure type systems, *Journal Functional Programming* 9 (6) (1999) 675–698.
16. B. C. Ruiz, The Expansion Postponement Problem for Pure Type Systems with Universes, in: *9th International Workshop on Functional and Logic Programming (WFLP'2000)*, Dpto. de Sistemas Informáticos y Computación, Technical University of Valencia (Tech. Rep.), 2000, pp. 210–224, september 28–30, Benicassim, Spain.
17. R. Pollack, Typechecking in pure type systems, in: B. Nordström, K. Petersson, G. Plotkin (Eds.), *Informal Proceedings of 1992 Workshop on Types for Proofs and Programs*, Båstad, 1992, pp. 271–288, <http://www.dcs.ed.ac.uk/lfcinfo/research/types-bra>.

18. B. C. Ruiz, *Sistemas de Tipos Puros con Universos*, Ph.D. thesis, Universidad de Málaga (1999).
19. L. van Benthem Jutting, *Typing in Pure Type Systems*, *Information and Computation* 105 (1) (1993) 30–41.
20. M. Strecker, *Construction and Deduction in Type Theories*, Ph.D. thesis, Universität Ulm (1999).
21. G. Gentzen, *Investigations into logical deductions*, in: M. Szabo (Ed.), *The Collected Papers of Gerhard Gentzen*, North-Holland, 1969, pp. 68–131.

Secrecy in Untrusted Networks^{*}

Michele Bugliesi¹, Silvia Crafa¹, Amela Prelic², and Vladimiro Sassone³

¹ Università “Ca’ Foscari”, Venezia;

² Max-Planck-Institut für Informatik;

³ University of Sussex

Abstract. We investigate the protection of migrating agents against the untrusted sites they traverse. The resulting calculus provides a formal framework to reason about protection policies and security protocols over distributed, mobile infrastructures, and aims to stand to ambients as the spi calculus stands to π . We present a type system that separates trusted and untrusted data and code, while allowing safe interactions with untrusted sites. We prove that the type system enforces a privacy property, and show the expressiveness of the calculus via examples and an encoding of the spi calculus.

Introduction

Secure communication in the π -calculus relies on *private channels*. Process $(\nu n)(\bar{n}\langle m \rangle \mid n(x).P)$ uses a private channel n to transmit message m . Intuitively, this guarantees the secrecy of m since no third process may interfere with n . In a distributed network, however, the subprocesses $\bar{n}\langle m \rangle$ and $n(x).P$ may be located at remote sites, and the link between them be physically insecure regardless of the privacy of n . It may therefore be desirable to implement a channel meant to deliver private information with lower level mechanisms, as for instance the *encrypted* connection over a *public* channel of the spi calculus [3]:

$$(\nu n)(\bar{p}\langle \{m\}_n \rangle \mid p(y).case\ y\ of\ \{x\}_n\ in\ P) \quad (1)$$

The knowledge of n is still confined here, but its role is different: n is an encryption key, rather than a channel. The message is encrypted and communicated along a public channel p ; even though the encrypted packet is intercepted, only the intended receivers, which possess the key n , may decrypt it to obtain m (cf. [1] for a thorough discussion of the shortcomings of the scheme.)

Similar mechanisms for secrecy are available for Mobile Ambients, MA, [10]. The following process, for instance, provides for the exchange of messages between locations a and b .

$$(\nu n)(a[\ n[\ out\ a.in\ b.\langle m \rangle \] \] \mid b[\ open\ n.(x).P \]) \quad (2)$$

^{*} Research supported by EU FET-GC ‘MyThS: Models and Types for Security in Mobile Distributed Systems’ IST-2001-32617 and ‘Mikado: Mobile Calculi based on Domains’ IST-2001-32222, and by MIUR Project ‘Modelli Formali per la Sicurezza’.

Ideally, no adversary can discover or seize m , or cause a different message to be delivered at b , as m is encapsulated into the secret ambient n .

The question we address in this paper is whether the abstract enveloping mechanism above can be turned into a realistic model of security for calculi of mobile agents that need to enforce protection policies and secrecy guarantees in untrusted environments. The answer we provide is articulated, and leads us to introduce new flexible, lower-level mechanisms. Our work is inspired by the development of spi from π in the ambition of identifying suitable such primitives.

Structure of the paper. §1 discusses how to achieve secrecy in (variants of) MA, and presents the motivations for introducing specific primitives; §2 provides a formal definition of the outcome, the SBA calculus, and illustrates it with a few examples. A key point of our work is the development in §3 of a type system which governs the interactions between trusted and untrusted (*opponents*) components travelling over open networks. Types split the world in two: the trusted *system* and the untrusted *context*. Relying on such intuition, data coming from the external environment is assigned an “unknown” type Public; Public values are handled with suspicion, since there is no saying what they are, or whom they are from. The type system guarantees a *secrecy* property proved in §4: a well-typed process does not disclose its secrets to any adversary, even though these may know its public names and traverse its sub-ambients. §5 presents an encoding of the spi calculus in SBA as a starting point for future comparisons between the two calculi.

1 A Sealing Mechanism for Ambients

The literature on mobile agent security focuses mainly on the dual problems of protecting a host from incoming agents and protecting a mobile agent from malicious hosts. Cryptography is used effectively in the latter case, by setting up a network of trusted sites and mechanisms of authentication between such sites and encrypted agents on the move (cf. [20,18]). The sealing mechanism we envision aims at protecting the secrecy of data inside ambient-like mobile agents which move freely in a network of possibly unreliable sites. The first question is whether these mechanisms are needed at all.

The security model of the Ambient Calculus [8] is centred around the idea that names provide the key to access the contents (data and code) encapsulated by ambients. Accordingly, as long an ambient name is secret, its content is protected from undesired access. The protocol for message exchange in (2), which here we question, is based on such secrecy assumption.

We start from the observation that ambient movement cannot occur without ambient revealing their names to some (not necessarily trusted) component of the underlying infrastructure. This happens in current implementations (cf. the ambient managers of [13] and the pointers-to-parent of [19]), and it is hard to conceive how it could possibly be otherwise. In the internetworking of the near future, crossing boundaries (routers, gateways, firewalls, . . .) will involve

running complex protocols. Travelling active packets will have to negotiate several conditions, such as QoS guarantees and bandwidth occupation, as well as paying for the service received. The principles of interoperability across different networks and of data encapsulation will require such protocols to work as direct dialogues between the interested parties. This can only rely on direct communication and, therefore, force agents to reveal their interfaces to the network. Thus, quite as secure remote communication cannot rely exclusively on private names, the security of a mobile ambient cannot be relegated to the confidentiality of its name. Back to our example, the encapsulation mechanism of (2) turns out not to be secure, as in a realistic scenario name n will have to be disclosed.

We may think of two ways to provide for stronger security guarantees. One possibility is to commit to agents their own security, by resorting to co-capabilities. For instance, process (1) can be recast in Safe Ambients, SA, [16] as shown below.

$$(\nu n)(a[n[\text{out } a.\text{in } b.\overline{\text{open}}\ n.\langle m \rangle] \mid \overline{\text{out}}\ a] \mid b[\overline{\text{in}}\ b.\text{open}\ n.(x)P]) \quad (3)$$

An alternative approach is to protect the secrecy of an ambient name by wrapping the ambient in a box that carries it to destination:

$$(\nu n)(a[p[\text{out } a.\text{in } b \mid n[\langle m \rangle]]] \mid b[\text{open } p.\text{open } n.(x)P]) \quad (4)$$

The first protocol guarantees that no one can enter n , or open it and read m before n reaches b , even though the name n is revealed while n is on the move. Notice that we ignore here the orthogonal issue of authenticating b against possible malicious impersonators. In (4), name n and message m are protected by the wrapper ambient p , to be opened at the target site b . Now n need not reveal its name – even though p is forcibly opened by an attacker – because it does not move.

Whether or not these protocols are satisfactory depends on the kind of agents and networks targeted. If we look at ambients as abstract physical devices, such as laptops or PDA's, then the first approach is likely to be all we need: physical devices can easily perform access control to protect their contents in ways similar to those encompassed by co-capabilities in (3). If instead, we think of ambients as representing “soft” agents, then (3) is only appropriate in “friendly” networks, where gateways respect the privacy of the code they route to the next hop to destination.

The second approach is more robust and applies well to the case of soft agents. In particular, in (4), we may think of $n[\langle m \rangle]$ as a piece of data encrypted under the key n : this is consistent with the structure of the protocol, as ambient n need not be active while inside p , since it is the thread $\text{out } a.\text{in } b$ that routes p (hence n) to destination. On the other hand, this solution cannot be fruitfully applied to protect active agents, which cannot move autonomously when encrypted.

The solution we advocate combines the benefits of the two approaches just discussed, by introducing new abstract primitives (which can be read as) providing

for subjective access control by ways of co-capabilities, and data encryption to preserve secrecy of data while allowing agents to move autonomously.

We develop our approach for the calculus NBA of [7], a calculus of (boxed) ambients based on two ideas: direct, named communication across parent-child boundaries, and dynamic learning of incoming ambients' names. An NBA ambient owns two channels, one for local, intra-ambient interactions, and one for hierarchical 'upward' communications. For instance $(x)^n.P \mid n[\langle m \rangle^\dagger.Q]$ reduces to $P\{x := m\} \mid n[Q]$, and symmetrically with the roles of input and output swapped. Moreover, co-capabilities are binders, so that $a[\text{in}\langle b, k \rangle.P] \mid b[\overline{\text{in}}(x, k).Q]$ reduces to $b[a[P] \mid Q\{x := a\}]$, and similarly for the out capability. This means that Q inside b has learnt the name of the incoming agent a . Observe that k acts to control access to b , and must be matched by a for the move to take place. Actually, name binding and access control checking work in a way at all analogous to the exchange of names and credentials which occur when registering for a networked service (cf. [7] for a deeper discussion and for related work).

Following the intuitions highlighted above, on top of the communication and movement mechanisms *à la* NBA, we introduce a specific primitive to let an ambient 'seal' itself: $n[\text{seal } k.P \mid Q] \longrightarrow n\llbracket P \mid Q \rrbracket_k$. By exercising the capability $\text{seal } k$ in one of its internal threads, ambient n blocks all its interactions with the outside and encrypts all its messages (to be exchanged either locally or across boundaries), included those in the thread Q . The flexibility of this mechanism derives from the fact that a "sealed" ambient $n\llbracket P \mid Q \rrbracket_k$ is still (partially) active: in particular, it may still move over the network and perform limited forms of local synchronisation. On the contrary, its message exchanges are blocked and all its data encrypted, and so remain until it reaches a computational environment which knows k , the sealing key. The mechanism to unseal a sealed ambient is associated to movement and exercised through co-capabilities containing keys such as in the following process, where $n\{P'\}$ is an ambient that can be either sealed or not:

$$n\llbracket \text{in } m.P \mid Q \rrbracket_k \mid m\{\overline{\text{in}}\{x\}_k.R \mid S\} \longrightarrow m\{n[P \mid Q] \mid R\{x := n\} \mid S\}$$

The resulting model can, in some respects, be viewed as a symmetric crypto-system, with encryption associated with the sealing capability that secures the data inside an ambient, and decryption associated with the dual operation of unsealing performed at ambient boundaries.

2 Sealed Ambients

The syntax of the SBA calculus below is a proper extension of the syntax of Boxed Ambients, BA, [5], with movement co-capabilities and new 'sealing' primitives.

<i>Expressions</i>	$M, N ::= k \cdots q \mid x \cdots z \mid \text{in } M \mid \text{out } M \mid \overline{\text{in}} \mid \overline{\text{out}} \mid M.M$
<i>Locations</i>	$\eta ::= M \mid \uparrow \mid \star$
<i>Prefixes</i>	$\pi ::= M \mid (x_1, \dots, x_k)^\eta \mid \langle M_1, \dots, M_k \rangle^\eta \mid \overline{\text{in}}\{x\}_M \mid \overline{\text{out}}\{x\}_M \mid \text{seal } M$
<i>Processes</i>	$P ::= \mathbf{0} \mid \pi.P \mid (\nu n)P \mid P \mid P \mid !P \mid M[P] \mid M\llbracket P \rrbracket_N$

Names $(k \cdots q)$ and variables $(x \cdots z)$ range over two disjoint sets; we use $a \cdots d$ to denote elements from either set, when the distinction is immaterial. Messages are formed as usual over names and (sequences of) capabilities. Locations indicate the target of a communication, i.e. a process in a child ambient M , in the parent ambient ($\uparrow$), or a local process ($\star$). The operators of inactivity, composition, restriction and replication are inherited from the π -calculus [17]. The process forms $(x_1, \dots, x_k)^\eta.P$, $\langle M_1, \dots, M_k \rangle^\eta.P$ and $M[P]$ denote directed (synchronous) input/output, as in BA, and ambients, as in MA. In addition, SBA provides a new construct for the formation of *sealed* ambients, noted $M\{P\}_N$, where M is the name and N is the sealing key. Three new prefix forms provide for the operations of unsealing, $\overline{\text{in}}\{x\}_k.P$, $\overline{\text{out}}\{x\}_k.P$, and sealing $\text{seal } k.P$.

We follow the usual conventions. Parallel composition has the lowest precedence among the operators, $\pi_1.\pi_2.P$ is read as $\pi_1.(\pi_2.P)$, while $\langle \tilde{M} \rangle^\eta, (\tilde{x})$ and $(\nu \tilde{n})$ stand for $\langle M_1, \dots, M_k \rangle^\eta, (x_1, \dots, x_k)^\eta$, $(\nu n_1, \dots, n_k)$, respectively. We omit trailing and isolated dead processes, writing π for $\pi.\mathbf{0}$, $\langle \tilde{M} \rangle$ for $\langle \tilde{M} \rangle.\mathbf{0}$, and $n[\]$ for $n[\mathbf{0}]$. The superscript $\star$ for local communication, is omitted. The operators $(\nu n)P$, $\overline{\text{in}}\{x\}_a.P$, $\overline{\text{out}}\{x\}_a.P$, and $(\tilde{x})^\eta.P$ act as binders for the name n , and the variables x and $\tilde{x}$, respectively. The sets of *free names* and *free variables* of P , $fn(P)$ and $fv(P)$, are defined accordingly. A process is closed if it has no free variables (though it may have free names). In addition, we write $M\{P\}$ for $M\{P\}_N$ or $M[P]$ when the distinction may safely be disregarded; notice that in the following $M\{P\}$ always refers to the same kind of ambient on both the sides of a reduction rule.

Reduction. The operational semantics is defined as usual in terms of reduction and structural congruence. The definition of structural congruence is standard (cf. [10]). The basic idea behind the reduction relation is that ambients can be in two states, either *sealed* or *unsealed*. An ambient may be sealed at its formation, or become sealed as a result of one of its enclosed threads exercising a capability. When sealed, an ambient may move but not exchange any value, either locally or with the context. An unsealed ambient is fully operational and may move, as well as communicate. The two states for reductions are formalised by defining the reduction relation in terms of two, inter-dependent relations, formalised in Table 1.

The relation $\rightarrow$ (referred to as *silent reduction*) gives the semantics of mobility and sealing. Rules (*enter*) and (*exit*) allow any ambient, sealed or unsealed, to traverse any other ambient, sealed or unsealed: the move requires the target ambient to cooperate by offering a co-capability. Rules (*K-enter*) and (*K-exit*) provide an alternative mechanism for mobility, akin to that studied in [7]. As in *loc. cit.*, the incoming ambient is authenticated by a test on the sealing key k , and then its name registered by binding it to the variable x . In addition, the authentication mechanism of SBA has the effect of removing the seal on the incoming ambient, so as to enable it to interact with the accepting context. Rule (*seal*) shows the effect of sealing: the capability $\text{seal } k$ instructs a process to seal its enclosing ambient under a key k . Notice that encryption of individual messages remains indicated only implicitly by the $\{ \dots \}$ around the ambient; besides be-

ing a convenient notation, this abstracts away from irrelevant implementation details.

Silent reductions may occur within any context, except under prefixes. On the contrary, the reductions involving communication – which are exactly as in previous versions of (N)BA, viz. [11,7] – may only occur within unsealed ambients, as formalised by the relation $\longrightarrow$. This reflects the fact that semantically relevant local communications must involve clear-text messages and, therefore, be avoided in untrusted environments, i.e. when ambients are sealed. Finally, rule (*struct*) is standard, while rules (*silent*) and (*ambient*) guarantee that the two reduction relations are linked properly.

Remarks. For ease of presentation, the syntax and operational semantics are so defined as to guarantee that ambients cannot be sealed more than once. An alternative choice would be to separate the sealing primitives from those for mobility. Specifically, one could introduce an explicit unsealing prefix such as $\text{unseal}\{x\}_k.P$, and define its semantics by the reduction $\text{unseal}\{x\}_k.P \mid n \llbracket Q \rrbracket_k \longrightarrow P\{x := n\} \mid n \llbracket Q \rrbracket$. This, together with rules (*enter*) and (*exit*), would implement an unsealing mechanism similar to ours, albeit not atomic. However, our proposal appears to model faithfully the current practice in distributed and mobile systems, where the protocols for agent authentication and certification take place at domain (i.e. ambient) boundaries rather than after such boundaries have been crossed.

Examples. The kind of secure communication expected of the exchange of messages in (2) can now be achieved as in $(\nu n)a \llbracket p[\text{seal } n.\text{out } a.\text{in } b.\langle m \rangle^\dagger] \rrbracket \mid \overline{\text{out}} \mid b \llbracket \text{in } \{x\}_n.(y)^x.P \mid Q \rrbracket$. The public ambient p seals itself with the private key n , shared by the sender and the intended receiver, moves over the network towards its destination, gets unsealed in the act of entering it, and becomes then ready to deliver its message. Like in the spi-calculus, it is the sealing key that is private, while the name of the ambient may be left public.

Incidentally, this formulation of the message exchange fixes a minor flaw in the protocols we discussed in §1 above. Namely, in the configuration $b \llbracket \text{open } n.(x).Q \mid Q' \mid n \llbracket \overline{\text{open}} n.\langle m \rangle \rrbracket \rrbracket$, as the opening of n and the delivery of its message are distinct steps, there is no guarantee that m will be received by the intended process when multiple threads are present inside b . In particular, m could end up in Q' , even when it did not actually know the secret name n . Such behaviour is however inherent to the communication model of MA, and easily be avoided with the primitives for hierarchical communication of the present calculus

As a more realistic example, consider the case of an agent in search of vendors of a particular item over the network. The agent originates at a user site u , visits a collection s_i of network sites and reports the names of those which provide a specific item it . To protect the agent moving over the network, we use the SBA primitives as follows. Let k be a sealing key shared between user u and sites s_i . The user can be represented as the process $u \llbracket (\nu a)a \llbracket P \mid R \rrbracket \mid Q \rrbracket$, where a is an agent with two threads: a router R , which controls movements, and a

Table 1. Reduction and Silent Reduction**Silent Reduction**

Silent Reduction Context $\mathbf{S} ::= - \mid (\nu n)\mathbf{S} \mid P \mid \mathbf{S} \mid n[\mathbf{S}] \mid n\llbracket \mathbf{S} \rrbracket_k$

MOBILITY (I)

(enter) $n\llbracket \text{in } m.P \mid Q \rrbracket \mid m\{\overline{\text{in}}.R \mid S\} \dashrightarrow m\{n\llbracket P \mid Q \rrbracket \mid R \mid S\}$

(exit) $m\{n\llbracket \text{out } m.P \mid Q \rrbracket \mid R\} \mid \overline{\text{out}}.S \dashrightarrow n\llbracket P \mid Q \rrbracket \mid m\{R\} \mid S$

MOBILITY (II)

(K-enter) $n\llbracket \text{in } m.P \mid Q \rrbracket_k \mid m\{\overline{\text{in}}\{x\}_k.R \mid S\} \dashrightarrow m\{n\llbracket P \mid Q \rrbracket \mid R\{x := n\} \mid S\}$

(K-exit) $m\{P \mid n\llbracket \text{out } m.Q \mid R \rrbracket_k\} \mid \overline{\text{out}}\{x\}_k.S \dashrightarrow m\{P\} \mid n\llbracket Q \mid R \rrbracket \mid S\{x := n\}$

(seal) $n[\text{seal } k.P \mid Q] \dashrightarrow n\llbracket P \mid Q \rrbracket_k$

STRUCTURAL RULES

(struct) $P \equiv Q, Q \dashrightarrow R, R \equiv S \Rightarrow P \dashrightarrow S$

(context) $P \dashrightarrow Q \Rightarrow \mathbf{S}\{P\} \dashrightarrow \mathbf{S}\{Q\}$

Reduction

Reduction Context $\mathbf{E} ::= - \mid (\nu n)\mathbf{E} \mid P \mid \mathbf{E} \mid n[\mathbf{E}]$

COMMUNICATION

(local) $(\tilde{x})P \mid \langle \tilde{M} \rangle Q \longrightarrow P\{\tilde{x} := \tilde{M}\} \mid Q$

(input n) $(\tilde{x})^n P \mid n[\langle \tilde{M} \rangle^\dagger Q \mid R] \longrightarrow P\{\tilde{x} := \tilde{M}\} \mid n[Q \mid R]$

(output n) $\langle \tilde{M} \rangle^n P \mid n[(\tilde{x})^\dagger Q \mid R] \longrightarrow P \mid n[Q\{\tilde{x} := \tilde{M}\} \mid R]$

STRUCTURAL RULES

(silent) $P \dashrightarrow Q \Rightarrow P \longrightarrow Q$

(ambient) $P \longrightarrow Q \Rightarrow n[P] \dashrightarrow n[Q]$

(struct) $P \equiv Q, Q \longrightarrow R, R \equiv S \Rightarrow P \longrightarrow S$

(context) $P \longrightarrow Q \Rightarrow \mathbf{E}\{P\} \longrightarrow \mathbf{E}\{Q\}$

communicator P , which interacts with the visited sites. We use two locks l and r to synchronise the two threads within a .

$$a[(\nu l, r)(\text{synch}(l) \mid !\overline{\text{synch}}(l).\text{seal } k.(\text{synch}(r) \mid \langle \text{it} \rangle^\dagger.(x, y)^\dagger([x = \text{it}]\langle y \rangle \mid \text{synch}(l)))) \\ \mid \overline{\text{synch}}(r).\text{route}(u, s_1).\overline{\text{synch}}(r).\text{route}(s_1, s_2).\overline{\text{synch}}(r).\text{route}(s_2, u)] .$$

where $[a = b]P \triangleq (\nu c) (c[\langle \rangle^a \mid b[(\langle \rangle^\dagger.\langle \rangle^\dagger] \mid (\langle \rangle^b.\langle \rangle^\dagger)] \mid (\langle \rangle^c.P), (c \notin \text{fn}(P))$

and $\text{synch}(n) \triangleq n[n\llbracket \text{out } n \rrbracket_n, \overline{\text{synch}}(n) \triangleq \overline{\text{out}}\{\cdot\}_n$

The first thread is a loop that, when activated, seals the agents under the key k , activates the router, and waits for a to be routed to the destination sites. Once there, it collects the name of the vendor, if this contains the desired item. The router thread, in turn, ships a across the network to visit the sites, in this case s_1 and s_2 . However, before moving outside u or any of the s_i , it waits for the sibling thread to seal the agent using k . The reduction semantics guarantees that, whenever the ambient a is not inside a site which knows k , all data in a are sealed, hence kept secret.

To synchronise with each other, the router and the communicator use the process forms: $\text{synch}(n)$ and $\overline{\text{synch}}(n)$. Interestingly, local synchronisation be-

tween threads is available even though the ambient is sealed, since it does not rely on exchanges of messages. Finally, each of the visiting sites can be coded as $s_i[\overline{\text{in}} \{z\}_k.(x)^z.\langle f_i(x), s_i \rangle^z \mid \dots]$. When agent a enters s_i it gets unsealed, so that it may hold exchanges with the site. Here the function f_i represents a lookup performed by the site searching for item x : the result is x if s_i has x on sale, or some different value otherwise. Of course, rather than total unsealing, a policy of selective decryption of sensitive data may be desirable when agents interact with sites only partially trusted; this variation of the example can easily be implemented in SBA.

3 A Type System

The type system separates trusted and untrusted data and code while allowing safe interactions with untrusted sites. In particular, a distinct type **Un** is used to type processes for which we cannot make any assumption on structure and/or behaviour. Correspondingly, we assign a ‘default’ type **Public** to data that comes from untyped processes, and we handle such data carefully. The structure of types is defined by the following productions:

$$\begin{aligned} \textit{Expression Types} \quad W &::= \text{Amb}[E] \mid \text{Key}[E] \mid \text{Public} \\ \textit{Exchange Types} \quad E, F &::= \text{shh} \mid (W_1, \dots, W_k) \\ \textit{Process Types} \quad T &::= [E, F] \mid \text{Un} \end{aligned}$$

Untrusted processes are built upon expressions of type **Public**. In addition, the type **Public** is assigned to expressions that trusted processes may exchange with untrusted ones. Among such expressions, we include the movement (co-)capabilities, so as to enhance the flexibility of typing: there is no negative effect on safety (or security) in this choice, as the interaction among trusted components is enabled by the possession of shared keys, which are secret and hence protected from the untrusted components. The type **Key**[E] is the type of sealing keys: a key with this type may only be used to seal (trusted) ambients of type **Amb**[E]. The latter, in turn, is the type of all the trusted ambients whose upward exchanges (if any) have type E . Notice that only ambients (not generic expressions) can be sealed. However, even untrusted ambients may be sealed, but in that case the sealing key is a generic expression of type **Public** and no security guarantee is made. As for process types, $[E, F]$ is the type of all processes that can be enclosed in ambients of type **Amb**[F], with E and F denoting the local and upward exchanges of the processes in question. **Un** is the type of the untrusted processes. In order to provide the intended privacy guarantees, the types of trusted and untrusted data and processes are kept separate (there is no subsumption rule, nor any common super-type). Nevertheless, the typing rules for processes allow non-trivial forms of interaction between trusted and untrusted processes. Specifically, ambients have full migration capabilities as the type system allow trusted ambients to traverse untrusted ones and vice versa (as in the example of §2). Instead, a trusted (resp. untrusted) sealed ambient may

be unsealed only within trusted (resp. untrusted) contexts. As for communication, the following policy is adopted: (i) local exchanges are allowed everywhere except that at top level, where we disallow local exchanges between trusted and untrusted processes, and (ii) trusted and untrusted ambients may exchange values across boundaries, provided that such values have type **Public**. We proceed with the description of the typing rules, collected in Tables 2 and 3.

Typing Rules. Every (co-)capability is assigned type **Public**; accordingly, the rule (PREFIX) allows trusted ambients to traverse untrusted ones and vice versa without breaking the soundness of the type system. Note that ill-formed (paths of) capabilities, such as $a.b$ and $\text{in } (a.b)$ do type check in this system when a, b are **Public**. This is necessary to allow full flexibility in the typing of the opponent: on the other hand, we will prove that the type system provides the expected guarantees of secrecy and safety for any value exchange.

Each process form has two associated typing rules, depending on whether the process in question is to be considered trusted (Table 2) or deemed untrusted (Table 3): in the latter case, it could be an attacker or a trusted process tainted by an interaction with an untrusted component via its public names. For prefixes the two cases can be accounted for by a single rule, (PREFIX), where T stands for either $[E, F]$ or **Un**. For ambients we need four rules: rule (AMB SEAL) in Table 2, assigns a type to ambients formed with the ‘right’ key N , and enclosing a process P with the expected exchanges. Rule (AMB) in Table 2 is standard. Rules (UNTRUSTED AMB/AMB SEAL), in Table 3 are used to type untrusted, possibly ill-formed, ambients. In addition, observe that a trusted (sealed) ambient may be typed with type **Un**; this is perfectly correct and allows a trusted (sealed) ambient to traverse untrusted sites.

The same rationale applies to the prefix constructors for sealing and unsealing, as well as for local and upward communication. Three typing rules handle the case of input (output) from a sub-ambient M . As we noted above, we allow untrusted and trusted process to exchange values, as long as these have type **Public**, as required in rules (UNTRUSTED INPUT/OUTPUT M) in Table 3. Note also that in these rules we do not require that the arity of the downward communication matches that of the target ambient. This leaves full flexibility in the typing of opponent processes, as it is implied by the following proposition.

Proposition 1 (Typability). *Let P be a process with $\text{fn}(P) = \{a_1, \dots, a_n\}$ and $\text{fv}(P) = \{x_1, \dots, x_m\}$. Then $a_1 : \text{Public}, \dots, a_n : \text{Public}, x_1 : \text{Public}, \dots, x_m : \text{Public} \vdash P : \text{Un}$,*

In other words, no constraint is imposed on the structure of the opponent: only that it initially does not know any secret. In addition, one can easily prove the standard property of type preservation under reduction.

Proposition 2 (Subject Reduction). *If $\Gamma \vdash P : T$ and $P \longrightarrow Q$, then $\Gamma \vdash Q : T$.*

Table 2. Typing Rules: Trusted Processes

(EMPTY)	(ENV x)	(PROJECTION)	(PATH)
$\frac{}{\emptyset \vdash \diamond}$	$\frac{\Gamma \vdash \diamond \quad x \notin \text{Dom}(\Gamma)}{\Gamma, x : W \vdash \diamond}$	$\frac{\Gamma \vdash \diamond \quad \Gamma(M) = W}{\Gamma \vdash M : W}$	$\frac{\Gamma \vdash M_1 : \text{Public} \quad \Gamma \vdash M_2 : \text{Public}}{\Gamma \vdash M_1.M_2 : \text{Public}}$
	(CO-IN)	(CO-OUT)	(IN M)
	$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \overline{\text{in}} : \text{Public}}$	$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \overline{\text{out}} : \text{Public}}$	$\frac{\Gamma \vdash M : W \quad W \in \{\text{Amb}[E], \text{Public}\}}{\Gamma \vdash \text{in } M : \text{Public}}$
(OUT M)	(PREFIX)	(SEAL)	
$\frac{\Gamma \vdash M : W \quad W \in \{\text{Amb}[E], \text{Public}\}}{\Gamma \vdash \text{out } M : \text{Public}}$	$\frac{\Gamma \vdash M : \text{Public} \quad \Gamma \vdash P : T}{\Gamma \vdash M.P : T}$	$\frac{\Gamma \vdash M : \text{Key}[E] \quad \Gamma \vdash P : [F, E]}{\Gamma \vdash \text{seal } M.P : [F, E]}$	
(AMB)	(AMB SEAL)		
$\frac{\Gamma \vdash M : \text{Amb}[E] \quad \Gamma \vdash P : [F, E]}{\Gamma \vdash M[P] : T}$	$\frac{\Gamma \vdash N : \text{Key}[E] \quad \Gamma \vdash M : \text{Amb}[E] \quad \Gamma \vdash P : [F, E]}{\Gamma \vdash M[P]_N : T}$		
(CO-IN KEY)	(CO-OUT KEY)		
$\frac{\Gamma \vdash M : \text{Key}[E] \quad \Gamma, x : \text{Amb}[E] \vdash P : [G, H]}{\Gamma \vdash \overline{\text{in}} \{x\}_M.P : [G, H]}$	$\frac{\Gamma \vdash M : \text{Key}[E] \quad \Gamma, x : \text{Amb}[E] \vdash P : [G, H]}{\Gamma \vdash \overline{\text{out}} \{x\}_M.P : [G, H]}$		
(DEAD)	(PAR)	(NEW)	(REPL)
$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0} : T}$	$\frac{\Gamma \vdash P : T \quad \Gamma \vdash Q : T}{\Gamma \vdash P \mid Q : T}$	$\frac{\Gamma, n : W \vdash P : T}{\Gamma \vdash (\nu n)P : T}$	$\frac{\Gamma \vdash P : T}{\Gamma \vdash !P : T}$
(LOCAL INPUT)	(LOCAL OUTPUT) $i = 1, \dots, k$		
$\frac{\Gamma, x_1 : W_1, \dots, x_k : W_k \vdash P : [(W_1, \dots, W_k), E]}{\Gamma \vdash (x_1, \dots, x_k).P : [(W_1, \dots, W_k), E]}$	$\frac{\Gamma \vdash M_i : W_i \quad \Gamma \vdash P : [\tilde{W}, E]}{\Gamma \vdash \langle \tilde{M} \rangle.P : [\tilde{W}, E]}$		
(INPUT $\uparrow$)	(OUTPUT $\uparrow$) $i = 1, \dots, k$		
$\frac{\Gamma, x_1 : W_1, \dots, x_k : W_k \vdash P : [E, (W_1, \dots, W_k)]}{\Gamma \vdash (x_1, \dots, x_k)^\uparrow.P : [E, (W_1, \dots, W_k)]}$	$\frac{\Gamma \vdash M_i : W_i \quad \Gamma \vdash P : [E, (W_1, \dots, W_k)]}{\Gamma \vdash \langle M_1, \dots, M_k \rangle^\uparrow.P : [E, (W_1, \dots, W_k)]}$		
(INPUT M)	(OUTPUT M AMB)		
$\frac{\Gamma \vdash M : \text{Amb}[\tilde{W}] \quad \Gamma, \tilde{x} : \tilde{W} \vdash P : [E, F]}{\Gamma \vdash (\tilde{x})^M.P : [E, F]}$	$\frac{\Gamma \vdash N : \text{Amb}[\tilde{W}] \quad \Gamma \vdash \tilde{M} : \tilde{W} \quad \Gamma \vdash P : [E, F]}{\Gamma \vdash \langle \tilde{M} \rangle^N.P : [E, F]}$		

4 A Secrecy Theorem

We refer to a standard notion of secrecy in the literature of security protocols, namely: *a process preserves the secrecy of a piece of data M if it does not publish M , or anything that would permit the computation of M* . The formal definition is inspired by [2]. We adapt that definition to our framework by representing an attacker as a closed, but otherwise arbitrary, context. This leaves full power to an attacker, which can either take the role of an hostile context (or host) enclosing a trusted process, as in $a[Q \mid (-)]$, or the role of a malicious agent mounting an attack to a remote host, as in $a[\text{in } p.\text{in } q.Q \mid Q'] \mid (-)$. In addition, we characterise the initial knowledge of the attacker in terms of the names, the keys and the capabilities initially known to it. Interestingly, the knowledge of

Table 3. Typing Rules: Untrusted Processes

(UNTRUSTED AMB)	(UNTRUSTED AMB SEAL)
$\Gamma \vdash M : \text{Public} \quad \Gamma \vdash P : \text{Un}$	$\Gamma \vdash N : \text{Public} \quad \Gamma \vdash M : \text{Public} \quad \Gamma \vdash P : \text{Un}$
$\Gamma \vdash M[P] : T$	$\Gamma \vdash M \llbracket P \rrbracket_N : T$
(UNTRUSTED SEAL)	(UNTRUSTED CAP)
$\Gamma \vdash M : \text{Public} \quad \Gamma \vdash P : \text{Un}$	$\Gamma \vdash M : \text{Public} \quad \Gamma \vdash P : \text{Un}$
$\Gamma \vdash \text{seal } M.P : \text{Un}$	$\Gamma \vdash M.P : \text{Un}$
(UNTRUSTED CO-IN)	(UNTRUSTED CO-OUT)
$\Gamma \vdash M : \text{Public} \quad \Gamma, x:\text{Public} \vdash P : \text{Un}$	$\Gamma \vdash M : \text{Public} \quad \Gamma, x:\text{Public} \vdash P : \text{Un}$
$\Gamma \vdash \overline{\text{in}} \{x\}_M.P : \text{Un}$	$\Gamma \vdash \overline{\text{out}} \{x\}_M.P : \text{Un}$
(UNTRUSTED LOCAL INPUT)	(UNTRUSTED LOCAL OUTPUT)
$\Gamma, x_1 : \text{Public}, \dots, x_k : \text{Public} \vdash P : \text{Un}$	$\Gamma \vdash M_i : \text{Public} \quad i = 1, \dots, k \quad \Gamma \vdash P : \text{Un}$
$\Gamma \vdash (x_1, \dots, x_k).P : \text{Un}$	$\Gamma \vdash \langle M_1, \dots, M_k \rangle.P : \text{Un}$
(UNTRUSTED INPUT $\uparrow$)	(UNTRUSTED OUTPUT $\uparrow$)
$\Gamma, x_1 : \text{Public}, \dots, x_k : \text{Public} \vdash P : \text{Un}$	$\Gamma \vdash M_i : \text{Public} \quad i = 1, \dots, k \quad \Gamma \vdash P : \text{Un}$
$\Gamma \vdash (x_1, \dots, x_k)^\uparrow.P : \text{Un}$	$\Gamma \vdash \langle M_1, \dots, M_k \rangle^\uparrow.P : \text{Un}$
(INPUT M UNTRUSTED)	
$\Gamma \vdash M : \text{Public} \quad \Gamma, x_1 : \text{Public}, \dots, x_k : \text{Public} \vdash P : T$	
$\Gamma \vdash (x_1, \dots, x_k)^M.P : T$	
(OUTPUT M UNTRUSTED)	
$\Gamma \vdash M : \text{Public} \quad \Gamma \vdash M_i : \text{Public} \quad i = 1, \dots, k \quad \Gamma \vdash P : T$	
$\Gamma \vdash \langle M_1, \dots, M_k \rangle^M.P : T$	
(UNTRUSTED INPUT M)	
$\Gamma \vdash M : \text{Amb}[\text{Public}_1, \dots, \text{Public}_n] \quad \Gamma, x_1 : \text{Public}, \dots, x_k : \text{Public} \vdash P : \text{Un}$	
$\Gamma \vdash (x_1, \dots, x_k)^M.P : \text{Un}$	
(UNTRUSTED OUTPUT M)	
$\Gamma \vdash M : \text{Amb}[\text{Public}_1, \dots, \text{Public}_n] \quad \Gamma \vdash M_i : \text{Public} \quad i = 1, \dots, k \quad \Gamma \vdash P : \text{Un}$	
$\Gamma \vdash \langle M_1, \dots, M_k \rangle^M.P : \text{Un}$	

capabilities is important here, since by exercising (a sequence of) capabilities an adversary may approach an agent and interact with it, even without knowing its name. As an example, if we take the process $b[\overline{\text{in}} \{x\}_k.\langle a \rangle^x]$, an opponent may have access to the value a even without knowing the name b : knowing the capability ‘in b ’ and the key ‘ k ’ is enough.

We define a context $\mathbf{A}(-)$ to be a process that contains exactly one variable $(-)$ (i.e. a hole). We denote with $\mathbf{A}(P)$ the process resulting from substituting the variable with P in A . Also, we denote with $fc(P)$ the set of capabilities formed over the free names of P : the inductive definition of this set is straightforward.

Definition 1 (S-adversary). *Let S be a finite set of names and capabilities. The closed context $\mathbf{A}(-)$ is an S -adversary if $fn(\mathbf{A}(-)) \cup fc(\mathbf{A}(-)) \subseteq S$.*

Next, we define what it means to preserve a secret: since capabilities are public, the definition of secrecy only applies to names. Let $\Rightarrow$ be the reflexive and transitive closure of the reduction relation $\longrightarrow$.

Definition 2 (Revealing Names, Preserving their Secrecy). *Let P be a process, n a name free in P , and S a finite set of names and capabilities. P may reveal n to S iff there exists an S -adversary $\mathbf{A}(-)$, with $\mathbf{A}(P)$ closed, and a name $c \in S$ such that $\mathbf{A}(P) \Rightarrow \mathbf{C}(c[\langle n \rangle^\uparrow \mid Q])$, for some context $\mathbf{C}(-)$ and process Q , with c not bound by $\mathbf{C}(-)$. Dually, P preserves the secrecy of n from S iff it does not reveal n to S .*

The definition extends readily to private names as follows (cf. [9]): $(\nu n)P$ may reveal n to S if and only if there is a fresh name m such that $P\{n := m\}$ may reveal m to S , with $m \notin S \cup \text{fn}(P)$. Notice that an adversary may dynamically acquire new names and new capabilities (i) by creating its own fresh names, (ii) by receiving names over public channels, and (iii) by unsealing ambients sealed with a key it knows (thus learning the ambient's name). As an example, take $S = \{c\}$, and consider the process $P = c[\langle a \rangle^\uparrow \mid a[\langle k \rangle^\uparrow]]$. P does not preserve the secrecy of k from S , even though S does not include a . In fact, one can take the S -adversary $\mathbf{A}(-) = (x)^c.(y)^x.c[\langle y \rangle^\uparrow \mid (-)]$, and note that $\mathbf{A}(P) \Rightarrow c[\langle k \rangle^\uparrow \mid c[] \mid a[]]$.

The secrecy theorem below states that a well-typed process P does not leak its secrets to any adversary that initially knows all the public names in P and has the capability to move in and out any ambient of P (included its secret ambients).

Theorem 1 (Secrecy). *Let P be a process such that $\Gamma \vdash P : \text{Un}$ and $\Gamma \vdash s : W$ with $W \neq \text{Public}$. Let $S = \{a \mid \Gamma \vdash a : \text{Public}\} \cup \{\text{in } a, \text{out } a \mid a \in \text{Dom}(\Gamma)\}$. Then P preserves the secrecy of s from S .*

Notice that the theorem only holds for well-typed processes of type Un . This immediately rules out processes that exchange non-public data at top level. Indeed, for such processes no secrecy guarantee can be made, for adversaries always have free access to the anonymous top level channel of any process. On the other hand, the theorem captures precisely the security guarantees our approach was intended to provide. That follows by observing (i) that well-typed ambient processes can always be typed with type Un , and (ii) that ambients (i.e. agents) are indeed the objects of our security concerns.

5 Encoding of the Spi Calculus

We further illustrate the calculus with an encoding of spi-calculus [3]. To ease the presentation, we focus on the following fragment of the asynchronous spi-calculus, in which we disregard the construct for pairs, natural numbers and matching.

$$\begin{aligned}
 \text{Expressions } M, N &::= n \mid x \mid \{M_1, \dots, M_n\}_N \\
 \text{Processes } P, Q &::= \mathbf{0} \mid \overline{M}\langle N_1, \dots, N_n \rangle \mid M(x_1, \dots, x_n)P \\
 &\mid P \mid Q \mid (\nu n)P \mid \text{case } M \text{ of } \{x_1, \dots, x_n\}_N \text{ in } P
 \end{aligned}$$

Table 4. Encoding of the spi calculus

$\langle a \rangle_p = a$	(for a a name or a variable)
$\langle \{M_1, \dots, M_n\}_k \rangle_p = p$	
$\llbracket a \rrbracket_p = 0$	(for a a name or a variable)
$\llbracket \{M_1, \dots, M_n\}_k \rrbracket_p = (\nu q_1, \dots, q_n) \quad \{q_1, \dots, q_n\} \cap (fn(M_1, \dots, M_n) \cup \{k, p\}) = \emptyset$ $\llbracket M_1 \rrbracket_{q_1} \mid \dots \mid \llbracket M_n \rrbracket_{q_n} \mid !p[(x)^\dagger.seal\ k.in\ x.\langle \langle M_1 \rangle_{q_1}, \dots, \langle M_n \rangle_{q_n} \rangle^\dagger]$	
$\llbracket 0 \rrbracket = 0, \quad \llbracket (\nu n)P \rrbracket = (\nu n)\llbracket P \rrbracket, \quad \llbracket P \mid Q \rrbracket = \llbracket P \rrbracket \mid \llbracket Q \rrbracket$	
$\llbracket b\langle M_1, \dots, M_n \rangle \rrbracket = (\nu q_1, \dots, q_n) \quad \{q_1, \dots, q_n\} \cap (fn(M_1, \dots, M_n) \cup \{b\}) = \emptyset$ $\llbracket M_1 \rrbracket_{q_1} \mid \dots \mid \llbracket M_n \rrbracket_{q_n} \mid b[\langle \langle M_1 \rangle_{q_1}, \dots, \langle M_n \rangle_{q_n} \rangle^\dagger]$	
$\llbracket b(x_1, \dots, x_n)P \rrbracket = (x_1, \dots, x_n)^b.\llbracket P \rrbracket$	
$\llbracket case\ M\ of\ \{x_1, \dots, x_n\}_k\ in\ P \rrbracket = (\nu p)(\llbracket M \rrbracket_p \mid (\nu c)(\langle c \rangle^{\llbracket M \rrbracket_p} \mid \{p, c\} \cap \{fn(P) \cup fn(M)\} = \emptyset$ $c[\text{in } \{y\}_k.(x_1, \dots, x_n)^y.(x_1, \dots, x_n)^\dagger] \mid (x_1, \dots, x_n)^c.\llbracket P \rrbracket))$	

The operational semantics of this fragment is standard (cf. [3]): in particular, decryption is governed by the following reduction: $case\ \{M_1, \dots, M_n\}_k\ of\ \{x_1, \dots, x_n\}_k\ in\ P \longrightarrow P\{x_i := M_i\}$.

The basic idea of the encoding is to represent an encrypted message with a sealed ambient that contains that message: communicating the encrypted message is then accounted for by communicating the name of the corresponding ambient. The formal definition is given in Table 4 in terms of three translation maps: $\langle \cdot \rangle_p : Expressions \mapsto Expressions$, $\llbracket \cdot \rrbracket_p : Expressions \mapsto Processes$, and $\llbracket \cdot \rrbracket : Processes \mapsto Processes$. In the first two (subsidiary) maps, p is the name of the ambient (if any) enclosing the message to be exchanged. In particular, if M is a name or a variable, then $\langle M \rangle_p$ returns M ; if instead M is an encryption packet, $\llbracket M \rrbracket_p$ returns p , the name of the ambient that stores the packet. Correspondingly, $\llbracket M \rrbracket_p$ stores M into an ambient named p , if M is an encrypted message, and returns the inactive process otherwise. More precisely, if M is a message encrypted under a key k , the ambient generated by $\llbracket M \rrbracket_p$ first reads a name x , then gets sealed with k to move into x , where eventually gets unsealed and delivers its payload. The use of replication on the ambient encoding an encryption packet accounts for the possible non-linear usage of messages in spi.

The encoding can be shown to be sound with respect to appropriate choices of behavioural equivalences in the two calculi, noted $\cong_{spi}$ and $\cong_{SBA}$, respectively. In particular, we take $\cong_{spi}$ to be *testing* equivalence, the notion of equivalence for spi-calculus studied in [3], and for SBA, we define $\cong_{SBA}$ to be reduction barbed congruence, based on the following exhibition predicate: $P \downarrow_b^{SBA} \triangleq P \equiv (\nu \tilde{n})((\tilde{x})^b P_1 \mid P_2)$. Given these choices, one can prove that the encoding is equationally sound.

Theorem 2 (Soundness of the encoding). *If $\llbracket P \rrbracket \cong_{SBA} \llbracket Q \rrbracket$ then $P \cong_{spi} Q$.*

6 Conclusions

We have investigated new mechanisms to protect migrating agents against the untrusted networks they traverse. Our primitives are best understood as low-level primitives to be employed for a secure implementation of the abstract mechanisms for secrecy found in mainstream ambient calculi.

The resulting calculus, SBA, is derived as a natural extension of NBA, the variant of Boxed Ambients studied in [7]. In fact, NBA can be interpreted into SBA by defining the capability $\text{in}\langle n, k \rangle$ as $\text{seal } k.\text{in } n$, and similarly for $\text{out}\langle n, k \rangle$. (Observe though that this lacks the atomicity of movement and credential verification of NBA.) On the other hand, the sealing model of SBA appears to provide strictly more flexibility and expressiveness than the access control of NBA: a SBA agent can be sealed by any of its local threads. Hence, an agent can be sealed and protected from undesired interactions by firing an action in one of its local threads, and it is not clear that a corresponding mechanism can be recovered in NBA.

We have investigated the role of types in enforcing static guarantees of safety and secrecy in the presence of untyped opponents. It is worth remarking that even though our typing deals with untrusted networks, similar ideas can be used to generalise those presented in [6] and in [11] for access control and information flow security with untrusted components.

Similar studies have been conducted on other process calculi in the literature. In fact, our use of the trusted/untrusted rules is directly inspired by work on the π/spi calculus (Cardelli et al. [9], Gordon and Jeffrey [14], Abadi and Gordon [3]). Alternative approaches to the same problem have also been investigated. Among these, Hennessy and Riely [15] study an extension of the $D\pi$ -calculus with a type system that labels some location as untrusted and relies on run-time type checking to enforce security restrictions for processes coming from untrusted locations. Similar approaches have also been advocated for Mobile Ambients [4], and other calculi (notably Klaim [12]).

Several questions remain to be explored, as for instance whether the data encryption underlying the sealing mechanisms we have introduced can be implemented effectively, and efficiently. Furthermore, in its current formulation, sealing an ambient has only the effect of guaranteeing the secrecy of data. More powerful mechanisms may be necessary to protect migrating agents by further hiding their structure or encrypting subcomponents consisting of data and code. Plans for future include work in both these directions.

Acknowledgements. We would like to thank Beppe Castagna for his suggestions, and the anonymous referees for their comments.

References

1. M. Abadi. Protection in programming-language translations. In *Proceedings of ICALP'98*, number 1443 in LNCS, pages 868–883. Springer-Verlag, 1998.

2. M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Proceedings of POPL'02*, pages 33–44. ACM Press, 2002.
3. M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, 1999.
4. M. Bugliesi and G. Castagna. Secure safe ambients. In *Proceedings of POPL'01*, pages 222–235. ACM Press, 2001.
5. M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *Proceedings of TACS'01*, number 2215 in LNCS, pages 38–63. Springer-Verlag, 2001.
6. M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *Proceedings of CONCUR 2001*, number 2154 in LNCS, pages 102–120. Springer-Verlag, 2001.
7. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication interference in mobile boxed ambients. In *FST&TCS 2002*, volume 2556 of LNCS, pages 71–84. Springer-Verlag, 2002.
8. L. Cardelli. Abstractions for mobile computations. In *Secure Internet Programming*, number 1603 in LNCS, pages 51–94. Springer-Verlag, 1999.
9. L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. In *Proceedings of CONCUR'00*, number 1877 in LNCS, pages 365–379. Springer-Verlag, August 2000.
10. L. Cardelli and A. Gordon. Mobile ambients. In *FoSSaCS'98*, number 1378 in LNCS, pages 140–155. Springer-Verlag, 1998.
11. S. Crafa, M. Bugliesi, and G. Castagna. Information Flow Security for Boxed Ambients. *ENTCS*, 66(3), 2002.
12. R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: a kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24:315–330, 1998.
13. C. Fournet, J.-J. Levy, and Schmitt. A. An asynchronous, distributed implementation of mobile ambients. In *Proceedings of IFIP TCS'00*, number 1872 in LNCS. Springer-Verlag, 2000.
14. A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of CSFW 2001*, pages 145–159. IEEE Computer Society, 2001.
15. M. Hennesy and J. Riely. Type-safe execution of mobile agents in anonymous networks. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS, pages 95–115. Springer-Verlag, 1999.
16. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings of POPL'00*, pages 352–364. ACM Press, 2000.
17. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Parts I and II. *Information and Computation*, 100:1–77, September 1992.
18. T. Sander and C. Tschudin. Towards mobile cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1998.
19. D. Sangiorgi and A. Valente. A distributed abstract machine for safe ambients. In *Proc. of ICALP 2001*, pages 408–420, 2001.
20. U. G. Wilhelm, L. Buttyà, and S. Staamann. On the problem of trust in mobile agent systems. In *Symposium on Network and Distributed System Security*. Internet Society, 1998.

Locally Commutative Categories

Arkadev Chattopadhyay and Denis Thérien*

School of Computer Science, McGill University, 3480 rue University, Montréal (PQ)
H3A 2A7, Canada
{achatt3,denis}@cs.mcgill.ca

Abstract. It is known that a finite category can have all its base monoids in a variety $\mathbf{V}$ (i.e. be locally $\mathbf{V}$, denoted $\ell\mathbf{V}$) without itself dividing a monoid in $\mathbf{V}$ (i.e. be globally $\mathbf{V}$, denoted $\mathbf{gV}$). This is in particular the case when $\mathbf{V}=\mathbf{Com}$, the variety of commutative monoids. Our main result provides a combinatorial characterization of locally commutative categories. This is the first such theorem dealing with a variety for which local differs from global. As a consequence, we show that $\ell\mathbf{Com} \subset \mathbf{gV}$ for every variety $\mathbf{V}$ that strictly contains the commutative monoids.

1 Introduction

In algebraic theory of automata, a language $L \subseteq A^*$ is said to be recognized by the finite monoid M if there exist a morphism $\phi : A^* \rightarrow M$ and a subset $F \subseteq M$ such that $L = \phi^{-1}(F)$. It is well-known that languages that can be so recognized are precisely the regular languages and that for each regular language there is a unique minimal monoid, called the *syntactic monoid* of L and denoted $M(L)$, that recognizes it. One expects that combinatorial properties of L would be reflected in the algebraic structure of $M(L)$: this intuition is completely valid and a driving theme of the field is to prove theorems of the following form:

“A language L belongs to the combinatorially-defined class $\mathcal{V}$ iff the syntactic monoid $M(L)$ belongs to the algebraically-defined class $\mathbf{V}$.”

For technical, but unavoidable, reasons, one sometimes has to deal with subsets of A^+ (instead of A^*) and semigroups (instead of monoids). Most often, “algebraically-defined” means that $\mathbf{V}$ is an M-variety, that is a class of finite monoids which is closed under division (i.e. morphic image and submonoid) and direct product. The notion of S-variety is similarly defined for finite semigroups. Books such as [1,2,4] offer a comprehensive treatment of this theory. One interesting by-product of results of the above form is that when membership in $\mathbf{V}$ is decidable, one gets a decision procedure to test if L is in $\mathcal{V}$, since the monoid $M(L)$ can be effectively computed from any of the common representations used for regular languages (automaton, regular expression, grammar, logical formula). Two classical theorems of that nature are the correspondence between star-free languages and aperiodic monoids [5] and the correspondence between piecewise-testable languages and $\mathcal{J}$ -trivial monoids [6].

* Research supported in part by NSERC and FCAR grants.

Consider the situation where two automata are connected in series: for the second machine it is no longer the case that the space of inputs it can receive forms a free monoid, since the input sequence is mediated through the first machine and some combinations may never arise. Technically, the right point of view needed for analyzing the computations of the second automaton, is to view the machine as operating over a free category rather than over a free monoid. In order to understand the all-important case of serial connection of automata and its algebraic incarnation i.e the wreath product of monoids, it is essential to generalize the above setting to the level of categories, e.g. deciding if a monoid M divides a wreath product of the form $S \circ T$ amounts to decide if a certain category, constructible from M and T , divides S . In this framework, one considers languages as sets of finite-length paths in a directed multigraph (instead of finite-length sequences over a set) and such languages may be recognized by finite categories (instead of finite monoids). The notion of the syntactic category of a language appears naturally and so does the notion of a C-variety, i.e. a class of finite categories closed under division and direct product. Thus the manipulation and understanding of finite categories are essential ingredients in manipulation and understanding of regular languages as observed and formalised in the seminal work of [10].

Given a C-variety $\mathbf{W}$, it is easily seen that the monoids in $\mathbf{W}$ form an M-variety. It is thus natural to consider the following question: for a fixed M-variety $\mathbf{V}$, what are the C-varieties $\mathbf{W}$ for which the monoids in $\mathbf{W}$ are precisely those of $\mathbf{V}$? Two natural examples emerge readily: the variety $\mathbf{gV} = \{C : C \text{ divides } M \text{ for some } M \in \mathbf{V}\}$, and the variety $\ell\mathbf{V} = \{C : \text{every base monoid of } C \text{ is in } \mathbf{V}\}$ are respectively the smallest and the largest C-variety with that property. It turns out that a combinatorial description of the languages recognized by monoids in $\mathbf{V}$ immediately implies a combinatorial description of the languages recognized by categories in $\mathbf{gV}$; similarly, an algebraic description of the monoids in $\mathbf{V}$ implies an algebraic description of the categories in $\ell\mathbf{V}$. Our understanding is thus complete whenever $\mathbf{gV} = \ell\mathbf{V}$; this happens in a number of interesting cases, e.g. for every non-trivial variety of groups, for semilattices, for aperiodic monoids. But there are also cases where $\mathbf{gV} \subsetneq \ell\mathbf{V}$, e.g. for the trivial variety, for commutative monoids [9], for $\mathcal{J}$ -trivial monoids [3]; apart from the case of the trivial variety, it becomes quite a challenge to find an algebraic description of $\mathbf{gV}$ or a combinatorial description of the languages recognized by members of $\ell\mathbf{V}$.

The main result of this paper is to provide a combinatorial description of the languages recognized by members of $\ell\mathbf{Com}$, the C-variety of locally commutative categories. This is the first instance of such result for a non-trivial variety $\mathbf{V}$ where $\mathbf{gV} \neq \ell\mathbf{V}$. We give our description via congruences of finite-index and some novel ideas have to be introduced. We also show that $\ell\mathbf{Com}$ is contained in $\mathbf{gV}$ for every M-variety $\mathbf{V}$ that strictly contains all commutative monoids. We then use known techniques to derive results about the S-variety $\mathbf{LCom} = \{S : eSe \in \mathbf{Com} \text{ for every } e = e^2\}$. The paper is organized as follows: section 2 presents the basic notions that are needed, section 3 proves the main theorem about locally commutative categories and section 4 describes the consequences of that result.

2 Basic Notions

A directed multigraph $G = (V, A, \alpha, \omega)$ consists of a set V of vertices, a set A of directed edges and two mappings $\alpha, \omega : A \rightarrow V$, which assigns to each edge a the start vertex $\alpha(a)$ and the end vertex $\omega(a)$ of that edge. Two edges a, b are consecutive iff $\omega(a) = \alpha(b)$. A path of length $n > 0$ is a sequence of n consecutive edges; we extend the mappings α and ω to paths in the natural way. For each vertex v we allow an empty path 1_v of length 0 for which $\alpha(1_v) = \omega(1_v) = v$. The length of a path x will be denoted by $|x|$, and the number of occurrences of an edge a in x by $|x|_a$. Two paths x, y are coterminial, denoted $x \sim y$ if $\alpha(x) = \alpha(y)$ and $\omega(x) = \omega(y)$. A loop is a path x such that $\alpha(x) = \omega(x)$ and a loop edge is a loop that consists of a single edge; we denote by $\bar{x}$ the path obtained from x by removing its loop edges. For a path x and a vertex v , let $x[v]$ stand for the subsequence of x consisting of all edges of the path that are incident on vertex v ; note that $x[v]$ is not itself a path, and that when x is a loop $\bar{x}[v]$ has even length for each v .

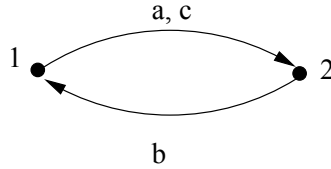
An equivalence β on the set G^* of all paths in G is a graph congruence iff $x\beta y$ implies $x \sim y$ and $x_1\beta y_1, x_2\beta y_2, \omega(x_1) = \alpha(x_2)$ imply $x_1x_2\beta y_1y_2$. The set of congruence classes, G^*/β , then forms a category. For each path x , we denote the corresponding congruence class containing x by $[x]_\beta$. We note that for every vertex v , the set $\{[x]_\beta : x \text{ is a loop on } v\}$ forms a monoid; we call these the base monoids of G^*/β . We refer the reader to [10] for the technical definition of division of categories and we define a C-variety to be a class of finite categories which is closed under division and direct product.

Monoids can be identified with 1-vertex categories in an obvious way. If we restrict a C-variety to its 1-vertex members, we then get an M-variety. In general, there may exist several C-varieties which coincide on the monoids they contain. For a given M-variety $\mathbf{V}$ it is always the case that $\mathbf{gV} = \{C : C \text{ divides } M \text{ for some } M \in \mathbf{V}\}$ is the smallest C-variety having $\mathbf{V}$ as its restriction to monoids: similarly $\ell\mathbf{V} = \{C : \text{every base monoid of } C \text{ is in } \mathbf{V}\}$ is always the largest C-variety with this property. Thus the C-variety corresponding to $\mathbf{V}$ is unique iff $\mathbf{gV} = \ell\mathbf{V}$. Although this holds in several instances, here are two examples where this is not the case.

Example 1. Let $\mathbf{V} = \mathbf{1}$ be the M-variety consisting of the 1-element monoid only. Then for every graph G , $G^*/\beta \in \mathbf{g1}$ iff β and $\sim$ coincide. On the other hand, let B be the subset of those edges of G for which the start and the end vertices belong to different strongly connected components. Define $x\gamma y$ iff $x \sim y$ and, for each $b \in B$, $x = x_0bx_1$ iff $y = y_0by_1$. Clearly G^*/β is in $\ell\mathbf{1}$ but not in $\mathbf{g1}$ if B is non-empty. In fact, it is an exercise to show that $G^*/\beta \in \ell\mathbf{1}$ iff $\gamma \subseteq \beta$. An interesting consequence of this observation is that $\ell\mathbf{1} \subset \mathbf{gV}$ whenever $\mathbf{1} \subset \mathbf{V}$. Indeed an edge b of B can appear in a path zero or one time only: if M is a non-trivial monoid, i.e. M contains an element $m \neq 1$, it can be used to distinguish paths in which b occurs from paths in which it does not, by mapping b to m and every other edge of the graph to 1. Taking a direct product of $|B|$ copies of M

insures that we can recover the equivalence class (in γ) of a path from its value in $M^{|B|}$.

Example 2. Let $\mathbf{V} = \mathbf{Com}$, the variety of all commutative finite monoids. On any graph G , define $x \gamma_{t,q} y$ iff $x \sim y$ and for each $a \in A$ either $(|x|_a < t$ and $|x|_a = |y|_a)$ or $(|x|_a \geq t, |y|_a \geq t$ and $|x|_a \equiv_q |y|_a$, where $\equiv_q$ denotes modulo q equality). It can be shown that $G^*/\beta \in \mathbf{gCom}$ iff $\gamma_{t,q} \subseteq \beta$ for some $t \geq 0, q \geq 1$. On the other hand, consider the following graph G :



define $x \beta y$ iff $x \sim y$ and $(|x| \leq 3$ and $x = y)$ or $(|x| > 3$ and $x \sim y)$. Then $G^*/\beta \in \ell\mathbf{Com}$ but not in $\mathbf{gCom}$.

This example is in some sense generic as [9] proves that a category C is in $\mathbf{gCom}$ iff it satisfies $xyz = zyx$ whenever x and z are coterminial; this result is combinatorially quite delicate to obtain. By definition, a category C is in $\ell\mathbf{Com}$ iff $xy = yx$ for every two loops x, y on the same vertex. The above example shows that knowing the number of occurrences of each edge in a path is not enough information to characterize the value of the path in a locally commutative category. Our present paper will provide a combinatorial description of the information that is missing in order to do so.

3 Combinatorial Characterization of Locally Commutative Categories

3.1 Free Locally Commutative Categories

Let G be a graph and define on G^* the congruence $x \gamma_\infty y$ iff $x \sim y$ and $|x|_a = |y|_a$ for every edge a . Let also θ_∞ be the coarsest congruence satisfying the equation $xyz \theta_\infty zyx$ whenever $x \sim z$. It was shown in [9] that $\gamma_\infty = \theta_\infty$.

The free locally commutative congruence on G^* , which we denote θ_∞^ℓ , is the coarsest congruence satisfying $xy \theta_\infty^\ell yx$ whenever x and y are loops on the same vertex. Obviously, θ_∞^ℓ refines $\theta_\infty = \gamma_\infty$. We also observe that $x \theta_\infty^\ell y$ iff $|x|_a = |y|_a$ for every loop edge a and $\bar{x} \theta_\infty^\ell \bar{y}$, i.e. the presence of loop edges cannot affect the congruence relation provided they are in equal number in both paths. There is another combinatorial property that is preserved by commutation of loops; let v be a vertex such that $|xy|_a = 0$ for each loop edge a on v and such that $|xy[v]|_a \leq 1$ for each a ; then the subsequence $xy[v]$ is an even permutation of the subsequence $yx[v]$. We now proceed to show that these combinatorial properties, the last one suitably modified, characterize θ_∞^ℓ .

In general, it is not the case that every edge appears at most once in a path. Suppose $|x|_a = k$; we make the k occurrences of a in x formally distinct by labelling them, in the order they appear, as $a_{\lambda(1)}, \dots, a_{\lambda(k)}$, where λ is a permutation of $\{1, \dots, k\}$. A *labelling* $\Lambda(x)$ of a path x is the result of applying this process to each edge. Thus the edges forming $\Lambda(x)$ can be viewed as being distinct. We will write $I(x)$ when the labelling is based on identity permutations for each edge, i.e. for each a , if $|x|_a = k$, the occurrences of a in x are renamed $a_1, \dots, a_k$ in that order.

We define on G^* $x \gamma_\infty^\ell y$ iff $x \gamma_\infty y$ and there exists a labelling Λ for $\bar{y}$ such that for every vertex v the sequence $\Lambda(\bar{y})[v]$ is an even permutation of the sequence $I(\bar{x})[v]$. It can be checked that γ_∞^ℓ is a congruence relation.

We state a useful property of γ_∞^ℓ

Proposition 1. *Let $x = x_1 \rho x_2$ and $y = y_1 \rho y_2$ be two paths in a graph such that $x \gamma_\infty^\ell y$ and ρ is a loop on some vertex v such that for each edge a in ρ we have $|x|_a = |y|_a = 1$. Then $x_1 x_2 \gamma_\infty^\ell y_1 y_2$.*

Proof. Clearly $x_1 x_2 \gamma_\infty y_1 y_2$. For the second property that we need to prove, we can assume that x and y do not contain any loop edge, since this property is dealing with $\bar{x}$ and $\bar{y}$. From the definition of γ_∞^ℓ there exists a labelling function Λ such that for each vertex v , $\Lambda(y)[v]$ is an even permutation of $I(x)[v]$. But every edge that appears in ρ is unique and so Λ and I must have labelled ρ exactly the same way. Also for every vertex v , $\Lambda(\rho)[v]$ and $I(\rho)[v]$ have the same length, which is even since ρ is a loop. This implies that $\Lambda(y_1 y_2)[v]$ is an even permutation of $I(x_1 x_2)[v]$ for every vertex v , as required. $\square$

An immediate corollary follows

Corollary 1. *If two paths x and y satisfy $x \gamma_\infty^\ell y$ and there are n loops $\rho_1, \dots, \rho_n$ appearing in x and y where for each edge a in a loop ρ_i we have $|x|_a = |y|_a = 1$ then the paths obtained by deleting these loops from x and y (say x' and y'), satisfy $x' \gamma_\infty^\ell y'$*

Proof. This follows by repeatedly applying proposition 1 once for every loop ρ_i . $\square$

Lemma 1. *For two paths x and y , $x \gamma_\infty^\ell y$ iff $x \theta_\infty^\ell y$.*

Proof. The implication from right to left is easy and left to the reader.

Now for the other direction we assume $x \gamma_\infty^\ell y$. Since every loop edge appears the same number of times in the two paths, it suffices to show $\bar{x} \theta_\infty^\ell \bar{y}$, so we now suppose that x and y have no loop edges. Because of the labelling involved in the definition of γ_∞^ℓ , we can think of x and y as having at most one occurrence of any edge.

We will prove our claim by induction on the length of the paths. For the base case of $|x| = 1$ the lemma is trivially true. Also note that if x and y are two coterminal paths that start with the same edge a and $x = ax'$ and $y = ay'$,

then $x \gamma_\infty^\ell y$ implies $x' \gamma_\infty^\ell y'$, since the occurrence of a is unique. Thus from the inductive hypothesis we obtain $x' \theta_\infty^\ell y'$ and this proves $x \theta_\infty^\ell y$.

Assume next that x and y start with different edges. Let $x = ax_0bx_1, y = by_0ay_1, v = \alpha(x) = \alpha(y)$; If v appears in y_1 , i.e. $y_1 = y_{10}y_{11}$ with $\omega(y_{10}) = v$, then we can commute by_0 and ay_{10} and we are back at the previous case. A similar argument holds if x_1 contains v . Otherwise the vertex v , which is the common end vertex of x_0 and y_0 , must appear at least once more in those two subpaths because $x[v]$ is an even permutation of $y[v]$. This implies the presence of an edge c in x_0 with start vertex v . This edge also appears in y , hence must appear in y_0 . We can thus use loop commuting to bring the c as first edge in each path, and so $x \theta_\infty^\ell cx' \gamma_\infty^\ell cy' \theta_\infty^\ell y$ for some x' and y' (note that this follows from the already proven fact that θ_∞^ℓ refines γ_∞^ℓ). Now we are back to the case handled before. $\square$.

The lemma above combinatorially captures the algebraic congruence θ_∞^ℓ and so provides a tool for describing the language recognized by free locally commutative categories. But it is impossible to work directly with the congruence γ_∞^ℓ for the case of finite categories since we have to deal with paths that are equivalent even though their lengths are different and so the concept of even permutations does not work anymore. This motivates us to find another way of characterising θ_∞^ℓ .

Consider the following special case. Let $x = ax_1yx_2zx_3a$ be a path where a is an edge which is coterminal with the subpaths y and z . One verifies that $x \theta_\infty^\ell zx_3yx_2ax_1a \theta_\infty^\ell zx_3yx_1ax_2a \theta_\infty^\ell zx_1ax_3yx_2a \theta_\infty^\ell ax_3zx_1yx_2a \theta_\infty^\ell ax_1yx_3zx_2a \theta_\infty^\ell ax_1zx_2yx_3a$. Thus we are able to interchange in x the coterminal subpaths y and z by using commutation of loops, because x contains an edge twice which is coterminal with these subpaths. The equivalence between exchange of coterminal paths and commutation of loops holds under a more general condition that we formalize below.

For a path x , define Γ_x^* as the reflexive and transitive closure of the relation Γ_x defined on the vertices by $v_1\Gamma_xv_2$ whenever there is an edge a such that $|x|_a \geq 2$ and $\alpha(a) = v_1, \omega(a) = v_2$ or $\alpha(a) = v_2, \omega(a) = v_1$.

Lemma 2. *For any path $x = x_1x_2x_3x_4x_5$ in G , if $\alpha(x_2)\Gamma_x^*\omega(x_2)$ and $x_2 \sim x_4$ then $x_1x_2x_3x_4x_5 \theta_\infty^\ell x_1x_4x_3x_2x_5$.*

Proof. Let $x = x_1x_2x_3x_4x_5, y = x_1x_4x_3x_2x_5, v_a = \alpha(x_2) = \alpha(x_4), v_b = \omega(x_2) = \omega(x_4)$. If $v_a = v_b$, the result is immediate. Otherwise we prove the lemma by showing that the hypothesis implies $x \gamma_\infty^\ell y$. Clearly $|x|_a = |y|_a$ for each a . Consider now $\bar{x}$ and $\bar{y}$, or equivalently assume that x and y contain no loop edges. We have to show that there exists a labelling Λ which will make $\Lambda(y)[v]$ an even permutation of $I(x)[v]$ for every vertex v . Since y is obtained by interchanging subpaths of x , we get naturally from $I(x)$ a first labelling Λ for y . For each vertex $v \neq v_a, v_b$, we have that $x_2[v]$ and $x_4[v]$ have even length. Since $\Lambda(y)[v]$ is obtained from $I(x)[v]$ by interchanging two blocks of even length, it must be an even permutation. The problem is that $x_2[v_a]$ and $x_4[v_a]$ have odd length, hence the permutation $\Lambda(y)[v_a]$ is odd, and the same for v_b . Since $v_a \Gamma_x^* v_b$

there exists some $n > 0$ such that $v_a = v_0 \Gamma_x v_1 \Gamma_x v_2 \dots \Gamma_x v_{n-1} \Gamma_x v_n = v_b$. Using the definition of Γ_x let e_i be the edge connecting v_{i-1} and v_i for $i > 0$. Each e_i is directed and its direction is arbitrary. Also there are at least two occurrences of e_i in both x and y . Let us create a new labelling Λ' that switches the labels (as given by Λ) of two arbitrarily chosen instances of e_i for each i . For all other edge occurrences, Λ' is the same as Λ . For each of $v_1, \dots, v_{n-1}$, $\Lambda'(y)[v_i]$ differs from $\Lambda(y)[v_i]$ by two transpositions, hence it remains even. For v_0 and for v_n , the difference between Λ and Λ' is one transposition, hence these become even as well. $\square$

An edge e in a path x is called a *special edge for x* iff $\alpha(e)$ and $\omega(e)$ are not related by Γ_x^* . A maximal subpath in a path x that is completely contained inside an equivalence class of Γ_x^* is called a *component of x* . So special edges always connect components that are over different equivalence classes of Γ_x^* . Note that a component could consist of just the identity path in which case two special edges would be adjacent to each other. Clearly every special edge occurs exactly once in a path x . Every path x in G^* is thus now uniquely decomposed as $x_0 e_1 x_1 \dots e_n x_n$, where the e_i 's are the special edges for x and the x_i 's are its components. The lemma above then gives the following result

Corollary 2. *If a path x has no special edges then for any path y , $x \theta_\infty^\ell y$ iff $x \theta_\infty y$ iff $x \gamma_\infty y$.*

In order to take into account the presence of special edges, we define, for each path x , a *reduced graph* $G_x = (V_x, A_x, \alpha_x, \omega_x)$ where $V_x = V/\Gamma_x^*$, A_x is the set of special edges for x , and α_x, ω_x are defined in the obvious way. The path x induces a path $Red(x)$ in the graph G_x by taking $Red(x)$ to be the sequence of special edges in the order they appear in x . Note that $Red(x)$ is a permutation of A_x and that $x \gamma_\infty y$ implies that $\Gamma_x^* = \Gamma_y^*$, hence that the graphs G_x and G_y are identical; furthermore we then have that $Red(x) \sim Red(y)$ in this graph.

We now define a congruence on G^* by $x \delta_\infty^\ell y$ iff $x \gamma_\infty y$ and $Red(x) \gamma_\infty^\ell Red(y)$.

Lemma 3. *For two paths x and y in G if $x \delta_\infty^\ell y$ and $Red(x) = Red(y)$ then $x \theta_\infty^\ell y$.*

Proof. Let $x = x_0 e_1 x_1 \dots e_n x_n$, $y = y_0 e_1 y_1 \dots e_n y_n$. Observe that this forces $x_i \sim y_i$ for each i . Fix an equivalence class C in V/Γ_x^* and let $0 \leq i_0 < i_1 < \dots, i_t \leq n$ be the indices for which x_{i_j} is a component of x over C ; the same sequence of indices gives the components of y that are over C . For each j replace the subpath of x between $x_{i_{j-1}}$ and x_{i_j} by a “meta-edge” E_j that goes from $\omega(x_{i_{j-1}})$ to $\alpha(x_{i_j})$. Do the same for y . Consider the paths $X = x_{i_0} E_1 x_{i_1} \dots E_t x_{i_t}$ and $Y = y_{i_0} E_1 y_{i_1} \dots E_t y_{i_t}$. We have that $X \gamma_\infty Y$ and these two paths now have no special edges since the two endpoints of each E_j are in C . By corollary 2, X can be transformed into Y by commuting loops. The corresponding sequence of operations will transform x into a path $x' = x'_0 e_1 x'_1 \dots e_n x'_n$ where $x'_i = y_i$ for $i \in \{i_0, \dots, i_t\}$ and $x'_i = x_i$ otherwise. Doing this for each class of V/Γ_x^* in turn will transform x into y . $\square$

We are now in a position to prove the equivalence of δ_∞^ℓ and θ_∞^ℓ .

Lemma 4. *For any two paths x and y in G $x \delta_\infty^\ell y$ iff $x \theta_\infty^\ell y$.*

Proof. The implication from right to left is easy and left to the reader. We prove the second implication. Suppose $x = x_0 e_1 x_1 \dots e_n x_n$; We fix in each equivalence class C of V/Γ_x^* a vertex v_C , and for each special edge e_i going from vertex v in C to a vertex v' in C' , we augment the graph G by introducing four new edges: e_i^C going from v to v_C , f_i^C going from v_C to v , g_i^C going from v' to $v_{C'}$ and h_i^C going from $v_{C'}$ to v' . We create from x a new path x' in the augmented graph by the following process: if e_j is a special edge for x going from vertex v in C to a vertex v' in C' , we replace e_j by $e_j^C f_j^C e_j g_j^{C'} h_j^{C'}$. If any loop edges have been added we remove them. We create y' from y similarly. $\text{Red}(x') \gamma_\infty^\ell \text{Red}(y')$ comes trivially from the fact that $x \delta_\infty^\ell y$ (since $\text{Red}(x) = \text{Red}(x')$ and $\text{Red}(y) = \text{Red}(y')$). Hence also $\text{Red}(x') \theta_\infty^\ell \text{Red}(y')$ by lemma 1. By construction, if there is a loop on vertex C appearing in $\text{Red}(x')$ in the reduced graph, there is a corresponding loop on vertex v_C appearing in x' in the augmented graph. Thus, corresponding to the sequence of loop commutations that transforms $\text{Red}(x')$ to $\text{Red}(y')$ in the reduced graph, there is a sequence of loop transformations, in the augmented graph, that transforms x' into a path (say w) in which the special edges appear in the same order as those of y' . Hence using lemma 3 it follows $x' \theta_\infty^\ell w \theta_\infty^\ell y'$. So $x' \gamma_\infty^\ell y'$ and by recalling that we obtained x' (y') from x (y) by adding a certain number of loops around every vertex v_C we apply proposition 1 and corollary 1 to get $x \gamma_\infty^\ell y$. Hence $x \theta_\infty^\ell y$. $\square$

Thus δ_∞^ℓ provides an alternative characterisation of locally commutative free categories. We will see in the next section that this characterisation can be naturally adapted to the case of finite categories.

3.2 Locally Commutative Finite Categories

We recall from [9] that the algebraic description of finite globally commutative categories is given by a path congruence $\theta_{t,q}$ generated by equations: $xyz_{t,q}zyx$ for $x \sim z$ and $x^t \theta_{t,q} x^{t+q}$ where x is a loop.

The corresponding combinatorial congruence $\gamma_{r,q}$ is induced by relations: for $x \sim y$ we say $x \gamma_{r,q} y$ iff for all edge $a \in A$, either $(|x|_a, |y|_a < r \text{ and } |x|_a = |y|_a)$ or $(|x|_a, |y|_a \geq r \text{ and } |x|_a \equiv_q |y|_a)$

We summarize the main result from [9] in the lemma below:

Lemma 5. *For every $t \geq 0$ and graph G there exists s such that for two paths x and y , $x \gamma_{s,q} y$ implies $x \theta_{t,q} y$.*

Observe that $\theta_{t,q}$ can be thought of as a rewriting system. If a path y can be obtained from path x using only loop commuting ($uw \rightarrow wu$) and loop replication ($u^t \rightarrow u^{t+q}$) without using loop deletion ($u^{t+q} \rightarrow u^t$), then we write $x \leq_{t,q} y$. It is a trivial observation that $x \leq_{t,q} y$ implies $x \theta_{t,q} y$. Clearly $x \leq_{t,q} y$ implies for all $a \in A$, $|x|_a \leq |y|_a$.

We now state a result that follows from the argumentation given in [8] for proving his lemma B.3.10 in Appendix B.

Proposition 2. *For paths x and y and for $t > 0$ if $\forall a \in A(G)$, $|x|_a \leq |y|_a$ and $x \gamma_{t+1,q} y$ then $x \leq_{t,q} y$.*

Proof. This follows from the argument used in the proof of lemma B.3.10 given in [8] (and is left as an exercise for the reader). $\square$

As an extension of ideas from free locally commutative categories we introduce $\theta_{t,q}^\ell$ to be the finite index path congruence generated by the conditions: $xy \theta_{t,q}^\ell yx$ where x and y are loops and $x^t \theta_{t,q}^\ell x^{t+q}$ where x is a loop. Analogous to the global case we write $x \leq_{t,q}^\ell y$ when $x \theta_{t,q}^\ell y$ and y can be obtained from x by just loop commuting and loop replication.

We also extend our combinatorial characterisation from the last section to $\delta_{t,q}^\ell$ meaning for two paths x and y $x \delta_{t,q}^\ell y$ iff $x \gamma_{t,q} y$ and $\text{Red}(x) \gamma_\infty^\ell \text{Red}(y)$, where γ_∞^ℓ gets defined on the reduced graph G_x . Note that this congruence only depends on the permutation of reduced paths which are of fixed length.

Using the definition of $\theta_{t,q}^\ell$ and the lemma 2 we can conclude the following:

Corollary 3. *For paths with no special edges, $\theta_{t,q}$ and $\theta_{t,q}^\ell$ are equivalent.*

This corollary along with lemma 5 gives us the intuition to expect the following result

Lemma 6. *If $x \delta_{s,q}^\ell y$ and $\text{Red}(x) = \text{Red}(y)$ then $x \theta_{t,q}^\ell y$ where s and t are related according to lemma 5.*

Proof. We direct the attention of the reader to the proof of lemma 3. Employing exactly the same technique as in that proof, fixing an equivalence class C in V/Γ_x^* we add “meta-edges” connecting two successive components of that class and obtain paths X and Y respectively from x and y . In our case here, $X \gamma_{s,q} Y$. Therefore using lemma 5 it follows $X \theta_{t,q} Y$ and since X and Y have no special edges from corollary 3 X can be transformed into Y by transformations preserving $\theta_{t,q}^\ell$. We apply the same operations on x to get a new path x' and then repeat the procedure with x' for each class of V/Γ_x^* to finally get y . $\square$

We can now combine the lemma above and proposition 2 to obtain the following corollary

Corollary 4. *If $x \delta_{t+1,q}^\ell y$ and $\text{Red}(x) = \text{Red}(y)$ with $|x|_a \leq |y|_a$, then $x \leq_{t,q}^\ell y$.*

Lemma 7. *For every $t \geq 2$ and $q \geq 1$, there exists $R \geq t + 1$ such that $x \delta_{R,q}^\ell y$ implies that there exists a path ρ satisfying $x \delta_{t+1,q}^\ell \rho$, where $\rho \theta_{t,q}^\ell y$ and for all edges $a \in A$, $|x|_a \leq |\rho|_a$.*

Proof. We will use lemmas 3.3 and 3.8 from [9] to prove this. Specifically let $R = m(G, t + 1)(|E| + 1) + 1$ where $m(G, t + 1) = |V| + (t + 1)(2^{|E|} - 1) + 2$ as defined in [9]. So for each edge a such that $|x|_a > |y|_a$ we have $|y|_a \geq R$ and since y can have at most $(|E| + 1)$ components there is at least one component that has at least $m(G, t + 1)$ occurrences of a . We can now straight away apply the argument used to prove lemma 3.8 in [9] and obtain the result of the present lemma. $\square$

Lemma 8. *If for two paths x and y , $|x|_a \leq |y|_a$ for all $a \in A$ and $x \delta_{t+1,q}^\ell y$, then $x \theta_{t,q}^\ell y$ for $t \geq 2$ and $q \geq 1$.*

Proof. We ask the reader to recall the technique used to prove lemma 4. We mimic the steps in that proof to augment the graph G by introducing four new edges for each special edge e_i and then modify the paths x and y to x' and y' respectively as prescribed there. (Note: we are using the same notation as in that proof.) Also let A' represent the set of edges of the augmented graph. The same argumentation of the earlier proof carries over to establish the existence of a path w such that $\text{Red}(w) = \text{Red}(y')$ and $x' \theta_\infty^\ell w \delta_{t+1,q}^\ell y'$. From corollary 4 it follows that $w \leq_{t,q}^\ell y'$ and hence $x' \leq_{t,q}^\ell y'$. This implies that there exists a series of loop commuting and loop duplicating transformations to obtain y' from x' . Let the loops that got duplicated, be called $\rho_1, \dots, \rho_n$ and let them be around vertices $v_1, \dots, v_n$ in G respectively. Also let n_i be the number of times ρ_i was duplicated. It is a trivial observation that every vertex v_i occurs somewhere in the path x and every loop ρ_i contains edges strictly from the unaugmented original graph G (since for each edge $a \in A' \setminus A$ we have $|x'|_a = |y'|_a$). Also no loop ρ_i contains any special edges as their count is one in both x' and y' . Hence every loop ρ_i could be added n_i times to path x to obtain a path u in G such that $u \delta_\infty^\ell y$ since $\text{Red}(x) = \text{Red}(u)$. This implies $x \delta_{t+1,q}^\ell y$ and hence from corollary 4 we have $x \theta_{t,q}^\ell u$. Now applying lemma 4 to u and y we get $u \theta_\infty^\ell y$ and hence $x \theta_{t,q}^\ell y$. $\square$

We now state the main result of this paper

Theorem 1. *β is a $\ell\mathbf{Com}$ -congruence iff there exist $R \geq 2$, $q \geq 1$ such that $\delta_{R,q}^\ell \subseteq \beta$.*

Proof. The direction from right to left is trivial and left as an exercise for the reader (it can be verified that $\delta_{R,q}^\ell$ is a $\ell\mathbf{Com}$ -congruence). For $t \geq 2$ we choose $R = m(G, t+1)(|E| + 1) + 1$ according to lemma 7. Then $x \delta_{R,q}^\ell y$ implies there exists a path z with $|x|_a \leq |z|_a$ for each edge $a \in A$ and $x \delta_{t+1,q}^\ell z \theta_{t,q}^\ell y$. Using lemma 8 we get $x \theta_{t,q}^\ell y$. We recall that for the cases $t = 0$ and $t = 1$, [9] proves $\ell\mathbf{Com}$ coincides with $\mathbf{gCom}$. $\square$

4 Consequences of the Main Result

In this section, we sketch some consequences of the combinatorial description obtained above. When an M -variety $\mathbf{V}$ is such that the C -varieties $\mathbf{gV}$ and $\ell\mathbf{V}$ differ, then $\ell\mathbf{V}$ cannot be equal to $\mathbf{gW}$ for any M -variety $\mathbf{W}$. How big should $\mathbf{W}$ be to insure $\ell\mathbf{V} \subset \mathbf{gW}$? In example 1 of section 2, we observed that for the trivial M -variety we have $\ell\mathbf{1} \subset \mathbf{gW}$ for every non-trivial $\mathbf{W}$. We now argue that a similar phenomenon occurs for $\mathbf{Com}$.

Theorem 2. *$\ell\mathbf{Com} \subset \mathbf{gW}$ for every M -variety $\mathbf{W}$ that strictly contains $\mathbf{Com}$.*

Proof. Our main result shows that in every locally commutative category, the value of a path is determined by the number of occurrences of each edge (threshold t , modulo q for some $t \geq 0, q \geq 1$) and the ordering of the so-called “special” edges. The first condition can be determined by using for each edge a cyclic counter of appropriate cardinality. For the second condition, let M be any non-commutative monoid, i.e. M contains two elements m and m' such that $mm' \neq m'm$. Fix two edges of the graph, a and b , map a to m , b to m' and every other edge to 1. If a path x contains at most one occurrence of each of a and b , which is necessarily the case when these two edges are special for x , the value of the path in M is in $\{1, m, m', mm', m'm\}$. In particular if both edges occur once, the order in which they appear can be recovered from the value in the monoid. If the graph has k edges, we can use the direct product of k cyclic counters to count occurrences of each edge, and $O(k^2)$ copies of M , one for each pair of edges. The value of the counters will determine the first condition and also which edges are special for a given path; we can then look up the appropriate copies of M to know in which order the special edges have appeared, hence recover the $\delta_{t,q}^\ell$ -value of the path. $\square$

Next, we transfer the last theorem to the S-variety $\mathbf{LCom} = \{S : eSe \in \mathbf{Com} \text{ for all } e = e^2\}$. For any semigroup S , consider the graph $G = (V, A, \alpha, \omega)$, where V is the set of idempotents of S , $A = V \times A \times V$, $\alpha(e, s, f) = e$, $\omega(e, s, f) = f$. Define the congruence β on G^* by identifying coterminal paths that multiply out to the same element in S . This construction trivially insures that $S \in \mathbf{LCom}$ iff $G^*/\beta \in \ell\mathbf{V}$. It follows from work of [7] that $S \in \mathbf{V} * \mathbf{D}$, where $\mathbf{D} = \{S : Se = e \text{ for all } e = e^2\}$ and $*$ denotes wreath product of varieties, iff $G^*/\beta \in \mathbf{gV}$. We thus get the following

Theorem 3. $\mathbf{LCom} \subset \mathbf{V} * \mathbf{D}$ for every M-variety $\mathbf{V}$ that strictly includes the commutative monoids.

5 Conclusion

In this paper we have proved a combinatorial description for the languages that can be recognized by finite locally commutative categories. This is the first result of that kind for a non-trivial M-variety for which the induced global and local C-varieties are different. We derived as a consequence the upper bound that for each M-variety $\mathbf{V}$ properly including the commutative monoids, the inclusion $\ell\mathbf{V} \subset \mathbf{gV}$ holds, which is similar to the situation for the trivial M-variety. It is easily checked that all these results can be proved, mutatis mutandis, for the C-variety of locally aperiodic commutative monoids. There is another famous case of an M-variety for which the induced global and local C-varieties are different, namely the variety $\mathbf{J}$ of $\mathcal{J}$ -trivial monoids. However, Jorge Almeida has pointed out to us that there exists a C-variety $\mathbf{gV}$ where $\mathbf{V}$ is a M-variety of aperiodic monoids that strictly contains $\mathbf{J}$, such that $\mathbf{gV}$ does not contain $\ell\mathbf{J}$. It would be interesting to find an upper bound for $\ell\mathbf{J}$ in terms of globally defined C-varieties.

References

1. J. Almeida. *Finite Semigroups and Universal Algebra*. World Scientific, 1994.
2. S. Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, New York, 1976.
3. R. Knast. Some theorems on graph congruences. *RAIRO Inform. Théor.*, 17:331–342, 1983.
4. J. E. Pin. *Varieties of Formal Languages*. Plenum, London, 1986.
5. M. Schützenberger. On finite monoids having only trivial subgroups. *Inform. and Control*, 8:190–194, 1965.
6. I. Simon. Piecewise testable events. In *2nd GI Conference*, volume 33 of *Lect. Notes in Comp. Sci.*, pages 214–222, Berlin, 1975. Springer.
7. H. Straubing. Finite semigroup varieties of the form $\mathbf{V} * \mathbf{D}$. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.
8. H. Straubing. *Finite Automata, Formal Logic and Circuit Complexity*. Birkhäuser, 1994.
9. D. Thérien and A. Weiss. Graph congruences and wreath products. *Journal of Pure and Applied Algebra*, 36:205–212, 1985.
10. B. Tilson. Categories as algebra: An essential ingredient in the theory of monoids. *Journal of Pure and Applied Algebra*, 48:83–198, 1987.

Semi-pullbacks and Bisimulations in Categories of Stochastic Relations

Ernst-Erich Doberkat

Chair for Software Technology
University of Dortmund
doberkat@acm.org

Abstract. The problem of constructing a semi-pullback in a category is intimately connected to the problem of establishing the transitivity of bisimulations. Edalat shows that a semi-pullback can be constructed in the category of Markov processes on Polish spaces, when the underlying transition probability functions are universally measurable, and the morphisms are measure preserving continuous maps. We demonstrate that the simpler assumption of Borel measurability suffices. Markov processes are in fact a special case: we consider the category of stochastic relations over Standard Borel spaces. At the core of the present solution lies a selection argument from stochastic dynamic optimization. An example demonstrates that (weak) pullbacks do not exist in the category of Markov processes.

Keywords: Bisimulation, semi-pullback, stochastic relations, labelled Markov processes, Hennessy-Milner logic.

1 Introduction

The existence of semi-pullbacks in a category makes sure that the bisimulation relation is transitive, provided bisimulation between objects is defined as a span of morphisms [10]. Edalat investigates this question for categories of Markov processes and shows that semi-pullbacks exist [6]. The category he focusses on has as objects universally measurable transition probability functions on Polish spaces, the morphisms are continuous, surjective, and probability preserving maps. His proof is constructive and makes essentially use of techniques of analytic spaces (which are continuous images of Polish spaces). The result implies that the semi-pullback of those transition probabilities which are measurable with respect to the Borel sets of the Polish spaces under consideration may in fact be universally measurable rather than simply Borel measurable. This then demands some unpleasant technical machinery when logically characterizing bisimulation for labelled Markov processes, cf. [2].

The distinction between measurability and universal measurability (both terms are defined in Sect. 2) may seem negligible at first. Measurability is the natural concept in measurable spaces (like continuity in topological spaces, or homomorphisms in groups), thus stochastic concepts are usually formulated in

terms of it. Universal measurability requires a completion process using all (σ) -finite measures on the measure space under consideration. In a Polish space the Borel sets are generated by the open sets, so the generators are well known. Comparable generators for the universally measurable sets are not that easy identified, let alone put to use. Thus it appears to be sensible to search for solutions for the problem of constructing semi-pullbacks for stochastic relations or labelled Markov processes first within the realm of Borel sets.

We show that the semi-pullback of Borel Markov processes exists within the category of these processes, when the underlying space is Polish (like the real line). Edalat considers transition probability functions from one Polish space into itself, this paper considers the slightly more general notion of a stochastic relation, cf. [1,4], i.e., transition sub-probability functions from one Polish space to another one. Rather than constructing the function explicitly, we show that the problem can be formulated in terms of measurable set-valued maps for which a measurable selector exists.

The paper's contributions are as follows. First it is shown that one can in fact construct semi-pullbacks in a category of stochastic relations between Polish spaces (and, by the way, an example shows that weak pullbacks do not exist). The second contribution is the reduction of an existential argument to a selection argument, a technique borrowed from dynamic optimization. Third it is shown that the solution for characterizing bisimulations for labelled Markov processes proposed by Desharnais, Edalat and Panangaden [2] can be carried over to Standard Borel spaces with their simple Borel structure.

This note is organized as follows: Sect. 2 collects some basic facts from topology, and from measure theory. It is shown that assigning a Polish space its set of sub-probability measures is an endofunctor on this category. Sect. 3 defines the category of stochastic relations, shows how to formulate the problem in terms of a set-valued function, and proves that a selector exists. This also implies the existence of semi-pullbacks for some related categories. A counterexample destroys the hope for strengthening this results to weak pullbacks. Finally, we show in Sect. 4 that the bisimulation relation is transitive for the category of stochastic relations, and that bisimilar labelled Markov processes are characterized through a weak negation free logic. Sect. 5 wraps it all up by summarizing the results and indicating areas of further work.

2 A Small Dose Measure Theory

This Section collects some basic facts from topology and measure theory for the reader's convenience and for later reference.

A *Polish space* $(X, \mathcal{T})$ is a topological space which has a countable dense subset, and which is metrizable through a complete metric, a measurable space $(X, \mathcal{A})$ is a set X with a σ -algebra $\mathcal{A}$. The *Borel sets* $\mathcal{B}(X, \mathcal{T})$ for the topology $\mathcal{T}$ is the smallest σ -algebra on X which contains $\mathcal{T}$. A *Standard Borel space* $(X, \mathcal{A})$ is a measurable space such that the σ -algebra $\mathcal{A}$ equals $\mathcal{B}(X, \mathcal{T})$ for some Polish topology $\mathcal{T}$ on X . Although the Borel sets are determined uniquely through the

topology, the converse does not hold, as we will see in a short while. Given two measurable spaces $(X, \mathcal{A})$ and $(Y, \mathcal{B})$, a map $f : X \rightarrow Y$ is $\mathcal{A} - \mathcal{B}$ -measurable whenever $f^{-1}[\mathcal{B}] \subseteq \mathcal{A}$ holds, where $f^{-1}[\mathcal{B}] := \{f^{-1}[B] \mid B \in \mathcal{B}\}$ is the set of inverse images $f^{-1}[B] := \{x \in X \mid f(x) \in B\}$ of elements of $\mathcal{B}$. Note that $f^{-1}[\mathcal{B}]$ is in any case an σ -algebra. If the σ -algebras are the Borel sets of some topologies on X and Y , resp., then a measurable map is called *Borel measurable* or simply a *Borel map*. The real numbers $\mathbb{R}$ carry always the Borel structure induced by the usual topology.

A map $f : X \rightarrow Y$ between the topological spaces $(X, \mathcal{T})$ and $(Y, \mathcal{S})$ is $\mathcal{T} - \mathcal{S}$ -continuous iff the inverse image of an open set from $\mathcal{S}$ is an open set in $\mathcal{T}$. Thus a continuous map is also measurable with respect to the Borel sets generated by the respective topologies.

When the context is clear, we will write down Polish spaces without their topologies, and the Borel sets are always understood with respect to the topology.

The following statement will be most helpful in the sequel. It states that, given a measurable map between Polish spaces, we can find a finer Polish topology on the domain, which has the same Borel sets, and which renders the map continuous; formally [11, Cor. 3.2.5, Cor. 3.2.6]:

Proposition 1. *Let $(X, \mathcal{T})$ and $(Y, \mathcal{S})$ be Polish spaces. If $f : X \rightarrow Y$ is a Borel measurable map, then there exists a Polish topology $\mathcal{T}'$ on X such that $\mathcal{T}'$ is finer than $\mathcal{T}$ (hence $\mathcal{T} \subseteq \mathcal{T}'$), $\mathcal{T}$ and $\mathcal{T}'$ have the same Borel sets, and f is $\mathcal{T}' - \mathcal{S}$ continuous.*

Given two measurable spaces X and Y , a *stochastic relation* $K : X \rightsquigarrow Y$ is a Borel map from X to the set $\mathbf{S}(Y)$, the latter denoting the set of all sub-probability measures on (the Borel sets) of Y . The latter set carries the *weak*- σ -algebra*. This is the smallest σ -algebra on $\mathbf{S}(Y)$ which renders all maps $\mu \mapsto \mu(B)$ measurable, where $B \subseteq Y$ is measurable. Hence $K : X \rightsquigarrow Y$ is a stochastic relation iff $K(x)$ is a sub-probability measure on (the Borel sets of) Y for all $x \in X$, such that $x \mapsto K(x)(B)$ is a measurable map for each Borel set $B \subseteq Y$.

Let Y be a Polish space, then $\mathbf{S}(Y)$ is usually equipped with the topology of weak convergence. This is the smallest topology on $\mathbf{S}(Y)$ which makes the map $\mu \mapsto \int_Y f \, d\mu$ continuous for each continuous and bounded $f : Y \rightarrow \mathbb{R}$. It is well known that this topology is Polish [9, Thm. II.6.5], and that its Borel sets is just the weak*- σ -algebra. If X is a Standard Borel space, then $\mathbf{S}(X)$ is also one: select a Polish topology $\mathcal{T}$ on X which induces the measurable structure, then $\mathcal{T}$ will give rise to the Polish topology of weak convergence on $\mathbf{S}(X)$ which in turn has the weak*- σ -algebra as its Borel sets.

A Borel map $f : X \rightarrow Y$ between the Polish spaces X and Y induces a Borel map $\mathbf{S}(f) : \mathbf{S}(X) \rightarrow \mathbf{S}(Y)$ upon setting $(\mu \in \mathbf{S}(X), B \subseteq Y \text{ Borel})$ $\mathbf{S}(f)(\mu)(B) := \mu(f^{-1}[B])$

It is easy to see that a continuous map f induces a continuous map $\mathbf{S}(f)$, and we will see in a moment that $\mathbf{S}(f) : \mathbf{S}(X) \rightarrow \mathbf{S}(Y)$ is onto, provided $f : X \rightarrow Y$ is. Denote by $\mathbf{P}(X)$ the subspace of all probability measures on X .

Let $\mathcal{F}(X)$ be the set of all closed and non-empty subsets of the Polish space X , and call for Polish Y a relation, i.e., a set-valued map $F : X \rightarrow \mathcal{F}(Y)$ $\mathcal{C}$ -

measurable iff, for any compact set $C \subseteq Y$, the weak inverse $\exists F(C) := \{x \in X \mid F(x) \cap C \neq \emptyset\}$ is measurable. A *selector* s for such a relation F is a single-valued map $s : X \rightarrow Y$ such that $\forall x \in X : s(x) \in F(x)$ holds. $\mathcal{C}$ -measurable relations have Borel selectors:

Proposition 2. *Let X and Y be Polish spaces. Then each $\mathcal{C}$ -measurable relation F has a measurable selector.*

Proof. Since closed subsets of Polish spaces are complete, the assertion follows from [8, Theorem 3]. $\square$

As a first application it is shown that $\mathbf{S}$ actually constitutes an endofunctor on the category of Standard Borel spaces with surjective measurable map as morphisms. This implies that $\mathbf{S}$ is the functorial part of a monad similar to the one studied by Giry [7].

Lemma 1. *$\mathbf{S}$ is an endofunctor on the category $\mathfrak{SB}$ of Standard Borel spaces with surjective Borel maps as morphisms.*

Proof. 1. Let X and Y be Standard Borel spaces, and endow these spaces with a Polish topology the Borel sets of which form the respective σ -algebras. Since $\mathbf{S}(X)$ is a Polish space under the topology of weak convergence, and since a Borel map $f : X \rightarrow Y$ induces a Borel map $\mathbf{S}(f) : \mathbf{S}(X) \rightarrow \mathbf{S}(Y)$ with all the compositional properties a functor should have, only surjectivity of the induced map has to be shown.

2. In view of Prop. 1 it is no loss of generality to assume that f is continuous. Continuity and surjectivity together imply that $y \mapsto f^{-1}[\{y\}]$ has closed and non-empty values in X . It constitutes a $\mathcal{C}$ -measurable relation, which has a measurable selector $g : Y \rightarrow X$ by Prop. 2, so that $f(g(y)) = y$ always holds. Let $\nu \in \mathbf{S}(Y)$, and define $\mu \in \mathbf{S}(X)$ as $\mu := \mathbf{S}(g)(\nu)$. Since $g^{-1}[f^{-1}[B]] = B$ for $B \subseteq Y$, it is now easy to establish that $\mathbf{S}(f)(\mu) = \nu$ holds. $\square$

Finally, the concept of universal measurability is needed. Let $\mu \in \mathbf{S}(X, \mathcal{A})$ be a sub-probability on the measurable space $(X, \mathcal{A})$, then $A \subseteq X$ is called μ -measurable iff there exist $M_1, M_2 \in \mathcal{A}$ with $M_1 \subseteq A \subseteq M_2$ and $\mu(M_1) = \mu(M_2)$. The μ -measurable subsets of X form a σ -algebra $\mathcal{M}_\mu(\mathcal{A})$. The σ -algebra $\mathcal{U}(\mathcal{A})$ of universally measurable sets is defined by

$$\mathcal{U}(\mathcal{A}) := \bigcap \{\mathcal{M}_\mu(\mathcal{A}) \mid \mu \in \mathbf{S}(X, \mathcal{A})\}$$

(in fact, one considers usually all finite or σ -finite measures, but these definitions lead to the same universally measurable sets). If $f : X_1 \rightarrow X_2$ is an $\mathcal{A}_1$ - $\mathcal{A}_2$ -measurable map between the measurable spaces $(X_1, \mathcal{A}_1)$ and $(X_2, \mathcal{A}_2)$, then it is well known that f is also $\mathcal{U}(\mathcal{A}_1)$ - $\mathcal{U}(\mathcal{A}_2)$ -measurable; the converse does not hold, and one usually cannot conclude that a map $g : X_1 \rightarrow X_2$ which is $\mathcal{U}(\mathcal{A}_1)$ - $\mathcal{A}_2$ -measurable is also $\mathcal{A}_1$ - $\mathcal{A}_2$ -measurable.

3 Semi-pullbacks

The category $\mathfrak{S}\mathfrak{R}\mathfrak{el}$ of stochastic relations has as objects triplets $\langle X, Y, K \rangle$, where X and Y are Standard Borel spaces, and $K : X \rightsquigarrow Y$ is a stochastic relation. A morphism $\langle \varphi, \psi \rangle : \langle X, Y, K \rangle \rightarrow \langle X', Y', K' \rangle$ is a pair of surjective Borel maps $\varphi : X \rightarrow X'$ and $\psi : Y \rightarrow Y'$ such that $K' \circ \varphi = \mathbf{S}(\psi) \circ K$ holds. Thus we have for $x \in X, B' \subseteq Y'$ Borel the equality $K'(\varphi(x))(B') = K(x)(\psi^{-1}[B'])$, so that morphisms are in particular measure preserving. Morphisms compose componentwise.

The category of Markov processes is a subcategory of $\mathfrak{S}\mathfrak{R}\mathfrak{el}$: it has as objects pairs $\langle X, K \rangle$, where X is a Standard Borel space, and $K : X \rightsquigarrow X$ is a stochastic relation, i.e., a Borel measurable transition probability function. Morphisms are surjective and measurable measure preserving maps.

Edalat [6] investigates a similar category, called $\mathfrak{M}\mathfrak{P}\mathfrak{roc}$ for easier reference: the objects are pairs $\langle X, K \rangle$ such that X is a Polish space, and K is a universally measurable transition sub-probability function. This requires that for each Borel set $A \subseteq X$ the map $x \mapsto K(x)(A)$ is $\mathcal{U}(\mathcal{B}(X))$ -measurable, and that $K(x) \in \mathbf{S}(X, \mathcal{B}(X))$ for each $x \in X$. Morphisms in $\mathfrak{M}\mathfrak{P}\mathfrak{roc}$ are surjective and continuous maps which are measure preserving. Note that an object $\langle X, K \rangle$ in $\mathfrak{M}\mathfrak{P}\mathfrak{roc}$ has the property that the set $\{x \in X | K(x)(A) \leq r\}$ is universally measurable for each Borel set $A \subseteq X$ and for each $r \in \mathbb{R}$; since each Borel set is measurable, this is a weaker condition than the one we will be investigating.

Assume that $\langle \varphi_i, \psi_i \rangle : \langle X_i, Y_i, K_i \rangle \rightarrow \langle X, Y, K \rangle$ ($i = 1, 2$) are morphisms in $\mathfrak{S}\mathfrak{R}\mathfrak{el}$, then a *semi-pullback* for this pair of morphisms is an object $\langle A, B, N \rangle$ together with morphisms $\langle \alpha_i, \beta_i \rangle : \langle A, B, N \rangle \rightarrow \langle X_i, Y_i, K_i \rangle$ ($i = 1, 2$) so that this diagram is commutative in $\mathfrak{S}\mathfrak{R}\mathfrak{el}$:

$$\begin{array}{ccc}
 \langle A, B, N \rangle & \xrightarrow{\langle \alpha_1, \beta_1 \rangle} & \langle X_1, Y_1, K_1 \rangle \\
 \downarrow \langle \alpha_2, \beta_2 \rangle & & \downarrow \langle \varphi_1, \psi_1 \rangle \\
 \langle X_2, Y_2, K_2 \rangle & \xrightarrow{\langle \varphi_2, \psi_2 \rangle} & \langle X, Y, K \rangle
 \end{array}$$

This means in particular that $K_1 \circ \alpha_1 = \mathbf{S}(\beta_1) \circ N$ and $K_2 \circ \alpha_2 = \mathbf{S}(\beta_2) \circ N$ should hold, so that a bisimulation is to be constructed (cf. Def. 1). The condition that $\langle A, B, N \rangle$ is the object underlying a semi-pullback may be formulated in terms of measurable maps as follows: N is a map from the Standard Borel space A to the Standard Borel space $\mathbf{S}(B)$ so that N is also a measurable selector for the set-valued function

$$b \mapsto \{\mu \in \mathbf{S}(B) | (K_1 \circ \alpha_1)(b) = \mathbf{S}(\beta_1)(\mu), (K_2 \circ \alpha_2)(b) = \mathbf{S}(\beta_2)(\mu)\}.$$

This translates the problem of finding the object $\langle A, B, N \rangle$ of a semi-pullback to a selection problem for set-valued maps, provided the spaces A and B together with the morphisms are identified.

It should be noted that the notion of a semi-pullback depends only on the measurable structure of the Standard Borel spaces involved. The topological structure enters only through Borel sets, and Borel measurability. From Prop. 1 we see that there are certain degrees of freedom for selecting a Polish topology that generates the Borel sets. They will be capitalized upon in the sequel.

Our goal is to establish:

Theorem 1. *$\mathfrak{S}\mathfrak{R}\mathfrak{el}$ has semi-pullbacks for each pair of morphisms*

$$\langle X_1, Y_1, K_1 \rangle \xrightarrow{\langle \varphi_1, \psi_1 \rangle} \langle X, Y, K \rangle \xleftarrow{\langle \varphi_2, \psi_2 \rangle} \langle X_2, Y_2, K_2 \rangle$$

with a common range.

We begin with a rather technical measure-theoretic observation: in terms of probability theory, it states that there exists under certain conditions a common distribution for two random variables with values in a Polish space with preassigned marginal distributions. This is a cornerstone for the construction leading to the proof of Theorem 1, it shows in particular where Edalat's work enters the present discussion.

Proposition 3. *Let Z_1, Z_2, Z be Polish spaces, $\zeta_i : Z_i \rightarrow Z$ ($i = 1, 2$) continuous and surjective maps, define $S := \{\langle x_1, x_2 \rangle \in Z_1 \times Z_2 \mid \zeta_1(x_1) = \zeta_2(x_2)\}$, and let $\nu_1 \in \mathbf{P}(Z_1), \nu_2 \in \mathbf{P}(Z_2), \nu \in \mathbf{P}(S)$ such that $\mathbf{P}(\pi_i)(\nu)(E_i) = \nu_i(E_i)$ holds for all $E_i \in \zeta_i^{-1}[\mathcal{B}(Z)]$ ($i = 1, 2$), where $\pi_1 : S \rightarrow Z_1, \pi_2 : S \rightarrow Z_2$ are the projections; S carries the trace of the product topology. Then there exists $\mu \in \mathbf{P}(S)$ such that $\mathbf{P}(\pi_i)(\mu)(E_i) = \nu_i(E_i)$ is true for all $E_i \in \mathcal{B}(Z_i) : (i = 1, 2)$.*

Proof. $\zeta_i : Z_i \rightarrow Z$ are morphisms in Edalat's category of probability measures on Polish spaces. The assertion then follows from the proof of [6, Cor. 5.4]. $\square$

In important special cases, there are other ways of establishing the Proposition, as will be discussed briefly.

Remark 1. 1. If $\zeta_i : Z_i \rightarrow Z$ are bijections, then the Blackwell-Mackey Theorem [11, Thm. 4.5.7] shows that $\zeta_i^{-1}[\mathcal{B}(Z)] = \mathcal{B}(Z_i)$. In this case the given measure $\nu \in \mathbf{P}(S)$ is the desired one.

2. If Z_1, Z_2, Z are not only Polish but also locally compact (like the real line $\mathbb{R}$), then a combination of the Riesz Representation Theorem and the equally famous Hahn-Banach Theorem can be used to construct the desired measure directly. This is the line of attack in [5]. Consequently, the somewhat heavy machinery of regular conditional distributions on analytic spaces need not be used (on the other hand, the Hahn-Banach Theorem relies on the Axiom of Choice which is not listed among the light weight tools either). —

The preparations for establishing that $\mathfrak{S}\mathfrak{R}\mathfrak{el}$ has semi-pullbacks are complete.

Proof. of Theorem 1

1. In view of Prop. 1 we may assume that the respective σ -algebras on X_1 and X_2 are obtained from Polish topologies which render φ_1 and K_1 as well as φ_2 and K_2 continuous. These topologies are fixed for the proof. Put

$$A := \{\langle x_1, x_2 \rangle \in X_1 \times X_2 \mid \varphi_1(x_1) = \varphi_2(x_2)\},$$

$$B := \{\langle y_1, y_2 \rangle \in Y_1 \times Y_2 \mid \psi_1(y_1) = \psi_2(y_2)\},$$

then both A and B are closed, hence Polish. $\alpha_i : A \rightarrow X_i$ and $\beta_i : B \rightarrow Y_i$ are the projections, $i = 1, 2$. We know that for $x_i \in X_i$ the equalities $K(\varphi_1(x_1)) = \mathbf{S}(\psi_1)(K_1(x_1))$ and $K(\varphi_2(x_2)) = \mathbf{S}(\psi_2)(K_2(x_2))$ hold. The construction implies that $(\psi_1 \circ \beta_1)(y_1, y_2) = (\psi_2 \circ \beta_2)(y_1, y_2)$ is true for $\langle y_1, y_2 \rangle \in B$, and $\psi_1 \circ \beta_1 : B \rightarrow Y$ is surjective.

2. Fix $\langle x_1, x_2 \rangle \in A$. Lemma 1 shows that $\mathbf{S}$ is an endofunctor on $\mathfrak{S}\mathfrak{B}$, in particular that the image of a surjective map under $\mathbf{S}$ is onto again, so that there exists $\mu \in \mathbf{S}(S)$ with $\mathbf{S}(\psi_1 \circ \beta_1)(\mu) = K(\varphi_1(x_1))$, consequently, $\mathbf{S}(\psi_i \circ \beta_i)(\mu) = \mathbf{S}(\psi_i)(K_i(x_i))$ ($i = 1, 2$). But this means that $\mathbf{S}(\beta_i)(\mu)(E_i) = K_i(x_i)(E_i)$ holds for all $E_i \in \psi_i^{-1}[\mathcal{B}(Y)]$. Put

$$\Gamma(x_1, x_2) := \{\mu \in \mathbf{S}(B) \mid \mathbf{S}(\beta_1)(\mu) = K_1(x_1) \wedge \mathbf{S}(\beta_2)(\mu) = K_2(x_2)\},$$

then Prop. 3 shows that $\Gamma(x_1, x_2) \neq \emptyset$.

3. Since K_1 and K_2 are continuous, $\Gamma(x_1, x_2) \subseteq \mathbf{S}(B)$ is closed, and the set $\exists \Gamma(C)$ is closed in A for compact $C \subseteq \mathbf{S}(B)$. In fact, let $(\langle x_1^{(n)}, x_2^{(n)} \rangle)_{n \in \mathbb{N}}$ be a sequence in this set with $x_i^{(n)} \rightarrow x_i$, as $n \rightarrow \infty$ for $i = 1, 2$, thus $\langle x_1, x_2 \rangle \in A$. There exists $\mu_n \in C$ such that $\mathbf{S}(\beta_i)(\mu_n) = K_i(x_i^{(n)})$. Because C is compact, there exists a converging subsequence $\mu_{s(n)}$ and $\mu \in C$ with $\mu = \lim_{n \rightarrow \infty} \mu_{s(n)}$ in the topology of weak convergence. Continuity of K_i implies that $\mathbf{S}(\beta_i)(\mu) = K_i(x_i)$, consequently $\langle x_1, x_2 \rangle \in \exists \Gamma(C)$, thus this set is closed, hence measurable. From Prop. 2 it can now be inferred that there exists a measurable map $N : A \rightarrow \mathbf{S}(B)$ such that $N(x_1, x_2) \in \Gamma(x_1, x_2)$ holds for every $\langle x_1, x_2 \rangle \in A$. Thus $N : A \rightsquigarrow B$ is a stochastic relation with $K_1 \circ \alpha_1 = \mathbf{S}(\beta_1) \circ N$, and $K_2 \circ \alpha_2 = \mathbf{S}(\beta_2) \circ N$. Hence $\langle A, B, N \rangle$ is the desired semi-pullback. $\square$

Specializing Theorem 1, we list some categories of stochastic relations which have semi-pullbacks. Whenever continuity enters the game, its proof shows that the semi-pullback has the continuity property, too.

Corollary 1. *The following categories have semi-pullbacks:*

1. *Objects are Standard Borel spaces with a sub-probability measure attached, morphisms are measure preserving and surjective Borel maps (continuous maps, resp.).*
2. *Objects are Markov processes over Standard Borel spaces (Polish spaces), morphisms are measure preserving and surjective Borel maps (continuous maps, resp.).*
3. *Objects are stochastic relations over Polish spaces, morphisms $\langle \varphi, \psi \rangle$ are as in $\mathfrak{S}\mathfrak{Rel}$ with ψ continuous. In the subcategory in which φ is also continuous semi-pullbacks exists, too*

Hence we know that the semi-pullback $\langle X, K \rangle$ for morphisms involving Markov processes is a Markov process again (whereas Edalat's main result [6, Cor. 5.2] permits only to conclude that K is a universally measurable transition sub-probability function).

Remark 2. One might be tempted now and ask for pullbacks or at least for weak pullbacks in the categories involved, now that the upper left hand corner of a pullback diagram can be filled. Recall that in a category a pair $\langle \tau_1 : c \rightarrow a_1, \tau_2 : c \rightarrow a_2 \rangle$ is a weak pullback for the pair $\rho_1 : a_1 \rightarrow b, \rho_2 : a_2 \rightarrow b$ of morphisms iff it is a semi-pullback (so that $\rho_1 \circ \tau_1 = \rho_2 \circ \tau_2$ holds), and if $\langle \tau'_1 : c' \rightarrow a_1, \tau'_2 : c' \rightarrow a_2 \rangle$ is another semi-pullback for that pair, then there exists a morphism $\theta : c' \rightarrow c$ so that $\tau'_i = \tau_i \circ \theta$ ($i = 1, 2$) holds. If the factor θ is unique, then the weak pullback is called a pullback.

The following example shows that even the category of Standard Borel spaces with probability measures where the morphisms are surjective and measure preserving measurable maps does not have always weak pullbacks: Let μ be the uniform distribution on $A := \{1, 2, 3\}$, put $B := \{a, b\}$ with $\nu(a) := \frac{2}{3}, \nu(b) := \frac{1}{3}$. Let $f : A \rightarrow B$ with $f(1) := f(2) := a, f(3) := b$. Then $f : \langle A, \mu \rangle \rightarrow \langle B, \nu \rangle$ is a morphism. Now compute the semi-pullback $\langle P, \gamma \rangle$ for the kernel pair represented by f . Then

$$P = \{\langle x, x' \rangle \mid f(x) = f(x')\} = \{\langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 2, 2 \rangle, \langle 3, 3 \rangle\},$$

and a suitable instance for γ is determined easily (e.g., $\gamma(\langle 3, 3 \rangle) = \frac{1}{3}$, all other pairs in P can be assigned $\frac{1}{6}$). The identity $\iota : \langle A, \mu \rangle \rightarrow \langle A, \mu \rangle$ has the property $f \circ \iota = f$. If a weak pullback exists, then we know about the factor ρ that $\rho(a) = \langle a, a \rangle$ holds for all $a \in A$; since f is not injective, ρ cannot be onto. This is a contradiction.

The reason for this is evidently that a weak pullback in e.g. $\mathfrak{S}\mathfrak{R}\mathfrak{el}$ would induce a weak pullback in the category of sets with ordinary maps as morphisms, but that it cannot be guaranteed there that the factor is onto, even if the morphisms for which the pullback is computed are.

Consequently, semi-pullbacks are the best we can do in $\mathfrak{S}\mathfrak{R}\mathfrak{el}$.

4 Bisimulation

This section demonstrates that the bisimulation relation on objects of $\mathfrak{S}\mathfrak{R}\mathfrak{el}$ is transitive, and serves as an application for the result that semi-pullbacks exist in this category. A final application is provided by proving the well known result due to Desharnais, Edalat and Panangaden that bisimilarity of labelled Markov processes may be characterized through a simple negation-free modal logic; the processes are based on Standard Borel spaces with measurable — rather than universally measurable — transition sub-probability functions.

We define a bisimulation for two objects in $\mathfrak{S}\mathfrak{R}\mathfrak{el}$ through a span of morphisms in that category [10]. This is similar to the notion of 1-bisimulation investigated in [4] for the comma category $\mathbf{1}_{\mathcal{M}} \downarrow \mathbf{S}$, were $\mathcal{M}$ is the category of all measurable spaces with measurable maps as morphisms.

Definition 1. An object P in $\mathfrak{SRel}$ together with morphisms $\langle \sigma_1, \tau_1 \rangle : P \rightarrow Q_1$ and $\langle \sigma_2, \tau_2 \rangle : P \rightarrow Q_2$ is called a bisimulation of objects Q_1 and Q_2 .

We apply the semi-pullback for establishing the fact that the bisimulation relation is transitive in $\mathfrak{SRel}$.

Proposition 4. The bisimulation relation between objects in the category $\mathfrak{SRel}$ of stochastic relations is transitive. The same is true for the subcategories of Markov processes introduced in Cor. 1.

Finally the characterization of bisimulations for labelled Markov processes through a Hennessy-Milner logic will be discussed. This follows the lines of [2] (a completely different approach is pursued in [3]). We will capitalize on the possibility to construct semi-pullbacks in categories of Markov processes over Polish spaces with Borel (rather than universally) measurable transition sub-probabilities. Hence we can characterize bisimulation in what seems to be a much more natural way from a probabilistic point of view, albeit for a restricted class of Markov processes for which the argumentation can be kept within the realm of Standard Borel spaces.

Fix a countable set $\mathbf{L}$ of actions.

Definition 2. Let S be a Standard Borel space, and assume that $k_a : S \rightsquigarrow S$ is a stochastic relation for each $a \in \mathbf{L}$. Then $(S, (k_a)_{a \in \mathbf{L}})$ is called a labelled Markov process.

S serves as a state space for the process. If the process is in state $s \in S$, and action $a \in \mathbf{L}$ is taken, then $k_a(s, B)$ is the probability for the next state to be a member of Borel set $B \subseteq S$.

Before proceeding, recall that a subset $A \subseteq X$ of a Polish space X is called *analytic* iff there exists a Polish space P and a continuous map $f : P \rightarrow X$ such that $A = f[P]$ holds. If A is equipped with the trace of the Borel sets of X , viz., $\{A \cap B \mid B \in \mathcal{B}(X)\}$ then A together with this σ -algebra is called an *analytic space*. The definition of a labelled Markov process found in [2] resembles the one given above, but assumes that the state space is analytic; *generalized* labelled Markov processes are introduced in which the transition sub-probability is assumed to be universally measurable.

Returning to Def. 2, let $(S, (k_a)_{a \in \mathbf{L}})$ and $(S', (k'_a)_{a \in \mathbf{L}})$ be labelled Markov processes with the same set $\mathbf{L}$ of actions. A *morphism* $f : (S, (k_a)_{a \in \mathbf{L}}) \rightarrow (S', (k'_a)_{a \in \mathbf{L}})$ is a surjective Borel map $f : S \rightarrow S'$ such that $k'_a \circ f = \mathbf{S}(f) \circ k_a$ holds for all $a \in \mathbf{L}$. Hence f is probability preserving for each action. Thus we have for each action $a \in \mathbf{L}$ a morphism between the objects (S, k_a) and (S', k'_a) in the category described in Cor. 1.(2). Applying Cor. 1 for each action separately and collecting the results yields:

Corollary 2. The category of labelled Markov processes with morphisms described above has semi-pullbacks.

From now on we omit the set $\mathbf{L}$ of actions when writing down labelled Markov processes.

In essentially the same way bisimulations are introduced through a span of morphisms: the labelled Markov processes $(S, (k_a))$ and $(S', (k'_a))$ are called *bisimilar* iff there exists a labelled Markov process $(T, (\ell_a))$ and morphisms $(S, (k_a)) \leftarrow (T, (\ell_a)) \rightarrow (S', (k'_a))$.

We follow [2] in introducing syntax and semantics of the Hennessy-Milner logic $\mathcal{L}$. The syntax is given by

$$\top \mid \phi_1 \wedge \phi_2 \mid \langle a \rangle_q \phi$$

Here $a \in \mathbf{L}$ is an action, and q is a rational number. Fix a labelled Markov process $(S, (k_a))$, then satisfaction of a state s for a formula ϕ is defined inductively. This is trivial for $\top$ and for formulas of the form $\phi_1 \wedge \phi_2$. The more complicated case is making an a -move: $s \models \langle a \rangle_q \phi$ holds iff we can find a measurable set $A \subseteq S$ such that $\forall s' \in A : s' \models \phi$ and $k_a(s, A) \geq q$ both hold. Intuitively, we can make an a -move in a state s to a state that satisfies ϕ with probability greater than q .

Denote by Φ the set of all formulas, and put $\llbracket \phi \rrbracket_S := \llbracket \phi \rrbracket := \{s \in S \mid s \models \phi\}$ as usual as the set of states that satisfy formula ϕ (we omit the subscript S if the context is clear). Let $(S', (k'_a))$ be another labelled Markov process, then define for $s \in S, s' \in S'$ the relation $s \approx s'$ iff s and s' satisfy all the same formulas. Formally, $s \approx s'$ holds iff $1_{\llbracket \phi \rrbracket}(s) = 1_{\llbracket \phi \rrbracket}(s')$ holds for all $\phi \in \Phi$, 1_A denoting the indicator function for the set A . Now define for labelled Markov processes the relation $\sim$ which indicates that two labelled Markov processes satisfy exactly the same formulas for logic $\mathcal{L}$: $(S, (k_a)) \sim (S', (k'_a))$ iff $[\forall s \in S \exists s' \in S' : s \approx s' \text{ and } \forall s' \in S' \exists s \in S : s' \approx s]$.

We will establish for labelled Markov processes the equivalence of bisimilarity and satisfying the same formulas, and we will follow essentially the line of attack pursued in [2]. But we want to stay within the realm of Standard Borel spaces. Working as in [2] with the set of equivalence classes with the final Borel structure for the quotient map for $\approx$ would bring us into the realm of analytic spaces. Instead we will work with a Borel set which intersects each equivalence class in exactly one element (what is usually called a *Borel cross section*, cf. [11, p. 186]). With T comes a surjective Borel map $f_T : S \rightarrow T$ which may stand in for the quotient map, so that we can construct from $(S, (k_a))$ another labelled Markov process $(T, (h_a))$ with f_T now acting as morphism. This is then applied to the case that $(S, (k_a)) \sim (S', (k'_a))$ by forming the sum of the processes and constructing from this sum through relation $\approx$ morphisms the semi-pullback of which will yield the desired bisimulation. So the plan is very similar to that in [2], but the terrain will be operated on in a slightly different manner.

We will restrict ourselves to processes for which the existence of a cross section is guaranteed:

Definition 3. *The $(S, (k_a))$ be a labelled Markov process is called small iff there exists a Borel cross section T for relation $\approx$, i.e., a Borel set $T \subseteq S$ which intersects each equivalence class in exactly one state.*

If S is locally compact, and each k_a is weakly continuous, then it can be shown that the process is small.

The cross section is a Standard Borel space in its own right, taking the Borel sets of S that are contained in T as its σ -algebra. Define $f_T : S \rightarrow T$ so that $f_T(x)$ is the unique element of $[x] \cap T$. Hence f_T picks from each class a representative, so that it may be interpreted as a selection map, because $s \approx f_T(s)$ always holds. f_T is a surjective Borel map [11, Prop. 5.1.9].

For the rest of the paper we assume that the processes involved are small.

Lemma 2. *Let $(S, (k_a))$ be small, and T, f_T as above. The σ -algebra on T is generated by the family $\mathcal{B}_0 := \{f_T[[\phi]] \mid \phi \in \Phi\}$, and $\mathcal{B}_0$ is closed under finite intersections.*

From these data a labelled Markov process can be constructed:

Corollary 3. *Let S, T, f_T be as above, and put for $a \in \mathbf{L}, s \in S$ and the measurable $B \subseteq T$ $h_a(f_T(s))(B) := k_a(s)(f_T^{-1}[B])$. This defines a labelled Markov process $(T, (h_a))$ such that $f_T : (S, (k_a)) \rightarrow (T, (h_a))$ is a morphism.*

We can now prove that satisfying the same formulas and bisimilarity are equivalent, following the trail laid out in [2].

Theorem 2. *Two small labelled Markov processes are bisimilar iff they satisfy the same formulas.*

Proof. 1. The “only if”-part follows from [2, Cor. 9.3], so only the “if”-part needs to be established. We proceed as in the proof of [2, Theorem 9.10] by constructing from the labelled Markov processes $(S, (k_a))$ and $(S', (k'_a))$ a diagram of the form

$$(S, (k_a)) \longrightarrow (T, (h_a)) \longleftarrow (S', (k'_a))$$

2. Let S_0 be the sum of the Standard Borel spaces S and S' , hence S_0 is a Standard Borel space again. Put for $a \in \mathbf{L}, s \in S_0$ and the Borel set $B \subseteq S_0$

$$\ell_a(s)(B) := \begin{cases} k_a(s)(S \cap B), & s \in S \\ k'_a(s)(S' \cap B), & s \in S' \end{cases}$$

Thus $(S_0, (\ell_a))$ is a labelled Markov process.

Since $(S, (k_a))$ is small, it has a Borel cross section T which is also a Borel cross section for S_0 , since both processes satisfy the same formulas. Thus $(S_0, (\ell_a))$ is small. Construct the associated Borel map $f_T : S_0 \rightarrow T$ for T , and define the labelled Markov process $(T, (h_a))$ as in Lemma 3. Let $i : S \rightarrow S_0, i' : S' \rightarrow S_0$ be the embeddings of S resp. S' into S_0 . Then $f_T \circ i : S \rightarrow T, f_T \circ i' : S' \rightarrow T$ are surjective. Both are morphisms. $\square$

Acknowledgements. The author wants to thank Georgios Lajios for his helpful and constructive comments. The referees' suggestions are gratefully acknowledged.

5 Conclusion

We show that one can construct a semi-pullback in the category of stochastic relations over Standard Borel spaces with Borel measurable and measure preserving maps as morphisms. The bisimulation relation is shown to be transitive. It is finally shown that the characterization of bisimulation through satisfiability in a simple logic may be derived in this conceptually simpler context, too.

We rely on selection arguments from the theory of set-valued relations. This technique permits drawing from the rich well of topology, in particular utilizing the weak topology on the space of all sub-probability measures. Selection arguments may be a helpful way of constructing objects; we illustrate this by showing that the map which assigns each Polish space its sub-probabilities and each surjective Borel measurable map the corresponding measure transform is actually a functor which may be difficult to establish otherwise.

Further work will investigate congruences on Markov processes. They arise in a natural fashion from morphisms and generalize relation $\approx$. Conditions on the smallness of these processes will be also of interest.

References

- [1] S. Abramsky, R. Blute, and P. Panangaden. Nuclear and trace ideal in tensored $*$ -categories. *Journal of Pure and Applied Algebra*, 143(1–3):3–47, 1999.
- [2] J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation of labelled markov-processes. *Information and Computation*, 179(2):163–193, 2002.
- [3] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating labeled Markov processes. In *Proc. 15th Symposium on Logic in Computer Science*, pages 95–106. IEEE, 2000.
- [4] E.-E. Doberkat. The demonic product of probabilistic relations. In Mogens Nielsen and Uffe Engberg, editors, *Proc. Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Computer Science*, pages 113–127, Berlin, 2002. Springer-Verlag.
- [5] E.-E. Doberkat. A remark on A. Edalat’s paper *Semi-Pullbacks and Bisimulations in Categories of Markov-Processes*. Technical Report 125, Chair for Software Technology, University of Dortmund, July 2002.
- [6] A. Edalat. Semi-pullbacks and bisimulation in categories of Markov processes. *Math. Struct. in Comp. Science*, 9(5):523–543, 1999.
- [7] M. Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, volume 915 of *Lecture Notes in Mathematics*, pages 68–85, Berlin, 1981. Springer-Verlag.
- [8] C. J. Himmelberg and F. Van Vleck. Some selection theorems for measurable functions. *Can. J. Math.*, 21:394–399, 1969.
- [9] K. R. Parthasarathy. *Probability Measures on Metric Spaces*. Academic Press, New York, 1967.
- [10] J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. Special issue on modern algebra and its applications.
- [11] S. M. Srivastava. *A Course on Borel Sets*. Number 180 in Graduate Texts in Mathematics. Springer-Verlag, Berlin, 1998.

Quantitative Analysis of Probabilistic Lossy Channel Systems

Alexander Rabinovich

School of Computer Science, Tel Aviv University, Israel
`rabinoa@post.tau.ac.il`

Abstract. Many protocols are designed to operate correctly even in the case where the underlying communication medium is faulty. To capture the behaviour of such protocols, *lossy channel systems (LCS)* [3] have been proposed. In an LCS the communication channels are modelled as FIFO buffers which are unbounded, but also unreliable in the sense that they can nondeterministically lose messages.

Recently, several attempts [5,1,4,6] have been made to study *Probabilistic Lossy Channel Systems (PLCS)* in which the probability of losing messages is taken into account and the following qualitative model checking problem is investigated: to verify whether a given property holds with probability one.

Here we consider a more challenging problem, namely to calculate the probability by which a certain property is satisfied. Our main result is an algorithm for the following *Quantitative model checking problem*:

Instance: A PLCS, its state s , a finite state ω -automaton $\mathcal{A}$, and a rational $\theta > 0$.

Task: Find a rational r such that the probability of the set of computations that start at s and are accepted by $\mathcal{A}$ is between r and $r + \theta$.

1 Introduction

Finite state machines which communicate through unbounded buffers (CFSM) have been popular in the modelling of communication protocols. A CFSM defines in a natural way an infinite state transition system. The fact that Turing machines can be simulated by CFSMs [7] implies that all the nontrivial verification problems are undecidable for CFSMs. Many protocols are designed to operate correctly even in the case where the underlying communication medium is faulty. To capture the behaviour of such protocols, *lossy channel systems (LCS)* [3] have been proposed as an alternative model. In an LCS the communication channels are modelled as FIFO buffers which are unbounded, but also unreliable in the sense that they can nondeterministically lose messages. Though an LCS defines in a natural way an infinite state transition system, it has been shown that the reachability problem for LCS is decidable [3], while progress properties are undecidable [2].

Probabilistic Lossy Channel Systems. Since we are dealing with unreliable communication media, it is natural to deal with models in which the probability

of losing messages is taken into account. Recently, several attempts [10,5,1,4,6] have been made to study *probabilistic Lossy Channel Systems (PLCS)* which introduce randomization into the behaviour of LCS. The works in [10,5,1,4,6] define different semantics for PLCS, depending on the manner in which the messages may be lost inside the channels. All these models associate in a natural way a countable Markov Chain (M.C.) to a PLCS.

Baier and Engelen [5] consider a model which assumes that at most a single message may be lost during each step of the execution of the system. They showed decidability of the following problems under the assumption that the probability of losing messages is at least 0.5.

Qualitative Probabilistic Reachability

Instance: A PLCS M and its states s_1, s_2 .

Question: Is s_2 reached from s_1 with probability one?

Qualitative Probabilistic Model-checking

Instance: A PLCS M , its state s and a finite state ω -automaton $\mathcal{A}$.

Question: Is the probability of the set of computations that start at s and are accepted by $\mathcal{A}$ equal to one?

The model in [1] assumes that messages can only be lost during send operations. Once a message is successfully sent to a channel, it continues to reside inside the channel until it is removed by a receive operation. Even the qualitative reachability problem was shown to be undecidable for this model of PLCS and losing probability $\lambda < 0.5$.

In [4,6] another semantics for PLCS was considered which is more realistic than that in [5,1]. More precisely, it was assumed that, during each step in the execution of the system, each message may be lost with a certain predefined probability. This means that the probability of losing a certain message will not decrease with the length of the channels (as it is the case with [5,1]).

For this model, the decidability of both the qualitative reachability and the qualitative model-checking problems was independently established in [4,6].

Our Contribution. All the above mentioned papers consider qualitative properties of PLCS. Here we consider a more challenging problem, namely to calculate the probability by which a certain property is satisfied.

Unfortunately, we were unable to prove that the probability of reaching a state s_2 from a state s_1 in PLCS is an algebraic number, or it is explicitly expressible by standard mathematical functions.

Therefore, we will approximate the probability by which a certain property is satisfied. Our main result is that the following two problems are computable.

Quantitative Probabilistic Reachability

Instance: A PLCS $\mathcal{L}$, its states s_1, s_2 and a rational $\theta > 0$.

Task: Find a rational r such that s_2 is reached from s_1 with the probability between r and $r + \theta$.

Quantitative Probabilistic Model-checking

Instance: A PLCS $\mathcal{L}$, its state s , a finite state ω -automaton $\mathcal{A}$, and a rational $\theta > 0$.

Task: Find a rational r such that the probability of the set of computations that start at s and are accepted by $\mathcal{A}$ is between r and $r + \theta$.

In order to approximate the probability p of the set of computations from a state s with a property φ in a PLCS $\mathcal{L}$ one can try to compute this probability p_n for the finite sub-chain $M_n = (S_n, P_n)$ of the countable Markov chain M generated by $\mathcal{L}$, where S_n is the set of states with at most n messages. There are two problems in this approach: (a) a state which was recurrent in M might become transient in M_n ; (b) how to find n which will ensure that the result p_n approximates up to θ the probability p in M .

In order to overcome problem (a) we analyze the structure of the recurrent classes of the Markov chain generated by a PLCS $\mathcal{L}$. For problem (b) the value for n is computed from an appropriate reduction of the Markov chains generated by PLCSs to one dimensional random walks.

Outline. In the next two sections we give basics of transition systems and countable Markov chains respectively. In Sect. 4 the quantitative probabilistic reachability problem over countable Markov chains is considered. In Sections 5 and 6 the semantics of lossy channel systems and probabilistic lossy channel systems are described. In Sect. 7 the algorithm for the quantitative probabilistic reachability problem over PLCS is presented and its complexity is analyzed. In Sect. 8 we generalize our results to the verification of the properties definable by ω -behavior of finite state automata.

2 Transition Systems

In this section, we recall some basic concepts of transition systems.

A *transition system* T is a pair $(S, \longrightarrow)$ where S is a (potentially) infinite set of *states*, and $\longrightarrow$ is a binary relation on S . We write $s_1 \longrightarrow s_2$ to denote that $(s_1, s_2) \in \longrightarrow$ and use $\xrightarrow{*}$ to denote the reflexive transitive closure of $\longrightarrow$. We say that s_2 is *reachable* from s_1 if $s_1 \xrightarrow{*} s_2$. For sets $Q_1, Q_2 \subseteq S$, we say that Q_2 is *reachable* from Q_1 , denoted $Q_1 \xrightarrow{*} Q_2$, if there are $s_1 \in Q_1$ and $s_2 \in Q_2$ with $s_1 \xrightarrow{*} s_2$. A *path* or *computation* π (from s_0) is a (finite or infinite) sequence $s_0, s_1, \dots, s_n, \dots$ of states with $s_i \longrightarrow s_{i+1}$ for $i \geq 0$. We use $\pi(i)$ to denote s_i , and write $s \in \pi$ to denote that there is an $i \geq 0$ with $\pi(i) = s$. For states s and s' , we say that π *leads* from s to s' , written, $s \xrightarrow{\pi} s'$, if $s = s_0$ and $s' \in \pi$.

For $Q \subseteq S$, we define the *graph* of Q , denoted $\text{Graph}(Q)$, to be the transition system $(Q, \longrightarrow')$ where $s_1 \longrightarrow' s_2$ iff $s_1 \xrightarrow{*} s_2$. A *strongly connected component* (SCC) in T is a maximal set $C \subseteq S$ such that $s_1 \xrightarrow{*} s_2$ for each $s_1, s_2 \in C$.

We say that C is a *bottom SCC (BSCC)* if there is no other SCC C_1 in T with $C \xrightarrow{*} C_1$. In other words, the BSCCs are the leaves in the acyclic graph of SCCs (ordered by reachability).

3 Markov Chains

In this section, we recall basic properties of *Markov chains*. We also introduce attractors which play an important role in our analysis of recurrent classes of Markov chains.

A *Markov chain* M is a pair (S, P) where S is a (potentially infinite) set of states and P is a mapping from $S \times S$ to the set $[0, 1]$, such that $\sum_{s' \in S} P(s, s') = 1$, for each $s \in S$.

A Markov chain induces a transition system where the transition relation consists of pairs of states related by positive probabilities. Formally, the *underlying transition system* of M is $(S, \longrightarrow)$ where $s_1 \longrightarrow s_2$ iff $P(s_1, s_2) > 0$. In this manner, the concepts defined for transition systems can be lifted to Markov chains. For instance, an SCC in M is a SCC in the underlying transition system.

A Markov chain (S, P) induces a natural measure on the set of computations from every state s .

Let us recall some basic notions from probability theory. A *measurable space* is a pair (Ω, Δ) consisting of a non empty set Ω and a σ -algebra Δ of its subsets that are called *measurable sets* and represent random events. A σ -algebra over Ω contains Ω and is closed under complementation and countable union. Adding to a measurable space a *probability measure* $Prob : \Delta \rightarrow [0, 1]$ such that $Prob(\Omega) = 1$ and that is countably additive, we get a *probability space* $(\Omega, \Delta, Prob)$.

Consider a state s of a Markov chain (S, P) . On the sets of computations that start at s , the probabilistic space $(\Omega, \Delta, Prob)$ is defined as follows (see [12]): $\Omega = sS^\omega$ is the set of all ω -sequences of states starting from s , Δ is the σ -algebra generated by the basic cylindric sets $D_u = uS^\omega$, for every $u \in sS^*$, and the probability measure $Prob$ is defined by $Prob(D_u) = \prod_{i=0, \dots, n-1} P(s_i, s_{i+1})$ where $u = s_0 s_1 \dots s_n$; it is well-known that this measure is extended in a unique way to the elements of the σ -algebra generated by the basic cylindric sets.

Consider a set $Q \subseteq S$ of states and a path π . We say that π *reaches* Q if there is an $i \geq 0$ with $\pi(i) \in Q$. We say that π *repeatedly reaches* Q if there are infinitely many i with $\pi(i) \in Q$. Let s be a state in S . We say that a state s is *recurrent* if $Prob\{\pi : \pi \text{ is a path from } s \text{ and } \pi \text{ repeatedly reaches } s\} = 1$. We say that a state s is *transient* if $Prob\{\pi : \pi \text{ is a path from } s \text{ and } \pi \text{ repeatedly reaches } s\} = 0$.

The next theorem summarizes standard properties of countable Markov chains [13].

Theorem 3.1. 1. *Every state is either transient or recurrent.*

2. *If s is recurrent then all the states reachable from s are recurrent.*

3. *Let C be a strongly connected component of a Markov chain. Then, either all the states in C are transient or all the states in C are recurrent.*

4. Let C be a recurrent strongly connected component of a Markov chain and $s_1, \in C$. Then $\text{Prob}\{\pi : \pi \text{ starts at } s_1 \text{ and repeatedly reaches every state of } C\} = 1$. For every state s and non-empty subset $B \subseteq C$ the probability to repeatedly reach every state of B from s is the same as the probability to reach B from s and is the same as the probability to reach s_1 from s .
5. A recurrent strongly connected component is always a bottom strongly connected component.

A recurrent (transient) SCC is often called a recurrent (transient) class. We introduce a central concept which we use in our solution for the probabilistic reachability problem, namely that of *attractors*.

Definition 3.2 (attractor). A set $A \subseteq S$ is said to be an attractor if, for each $s \in S$, the set A is reachable from s with probability one.

In other words, regardless of the state in which we start, we are guaranteed that we will reach the attractor with probability one. It is clear that an attractor has a state in every recurrent class. The Lemma below follows from Theorem 3.1 and describes properties of the BSCCs of the graph of a finite attractor A .

Lemma 3.3. Assume that a Markov chain M has a finite attractor A . Then (1) each BSCC C of $\text{Graph}(A)$ is a subset of a recurrent component in M . (2) A state is recurrent if and only if it is reachable from a BSCC C of $\text{Graph}(A)$. (3) For every s in M the set of recurrent states is reached from s with probability one.

4 Approximating Probability for Countable Markov Chains

Let M be a M.C. and let s_1, s_2 be its states. We use $\text{Prob}_M(s_1 \xrightarrow{*} s_2)$ for the probability with which s_2 is reached from s_1 in M . Let $\text{Comp}_n(s_1)$ be the set of all the computations of length n in M from s_1 . Partition $\text{Comp}_n(s_1)$ into three sets:

$$\text{Reach}_n(s_1, s_2) = \{\pi : \pi \in \text{Comp}_n(s_1) \wedge \exists i \leq n. \pi(i) = s_2\}$$

$$\text{Escape}_n(s_1, s_2) = \{\pi : \pi \in \text{Comp}_n(s_1) \setminus \text{Reach}_n(s_1, s_2) \wedge s_2$$

$$\text{is unreachable from } \pi(n)\}$$

$$\text{Endecided}_n(s_1, s_2) = \text{Comp}_n(s_1) \setminus \text{Reach}_n(s_1, s_2) \setminus \text{Escape}_n(s_1, s_2)$$

All the computations in $\text{Reach}_n(s_1, s_2)$ reach s_2 , and no computation in $\text{Escape}_n(s_1, s_2)$ extends to a computation that reaches s_2 . Note that $\text{Prob}(\text{Comp}_n(s_1)) = 1$. Let $p_n^+ = \text{Prob}(\text{Reach}_n(s_1, s_2))$, $p_n^- = \text{Prob}(\text{Escape}_n(s_1, s_2))$ and $p_n^? = \text{Prob}(\text{Endecided}_n(s_1, s_2))$. Observe that p_n^+ and p_n^- are increasing sequences, while $p_n^?$ is decreasing and

$$p_n^+ \leq \lim p_n^+ = \text{Prob}_M(s_1 \xrightarrow{*} s_2) \leq p_n^+ + p_n^? \quad (1)$$

Path Enumeration (PE) scheme for approximation $Prob_M(s_1 \xrightarrow{*} s_2)$ is based on (1).

Path Enumeration Scheme for Approximating Probabilistic Reachability

Instance: A M.C. M , its states s_1, s_2 and a $\theta > 0$.

Task: Find r such that s_2 is reached from s_1 with the probability between r and $r + \theta$.

begin

1. $n := 0; \Delta := 1;$

2. **while** $(\Delta > \theta)$ **do**

3. $n := n + 1;$ Compute $r := p_n^+;$ Compute $\Delta := p_n^?$

end while

4. **return**(r)

end

In the above problem, we do not assume that M is finite. Hence, these are not instances of an algorithmic problem. In Sect. 7 we consider the quantitative reachability problem when countable Markov chains are described by probabilistic lossy channel systems. For such finite descriptions we investigate the corresponding algorithmic problem.

If M has finite branching and is presented effectively, then p_n^+ is computable. Moreover, if in addition the reachability problem for the transition system underlying M is decidable, then $\text{Escape}_n(s_1, s_2)$, $\text{Endecided}_n(s_1, s_2)$ and $p_n^?$ can be computed. Hence, in this case the scheme can be implemented. Observe that PE scheme terminates only if $\lim p_n^? < \theta$. Therefore,

Lemma 4.1. *If $\lim p_n^? = 0$ then PE scheme terminates.*

It is well-known that for a finite state Markov chains $\lim p_n^? = 0$. This property holds for Markov chains with finite attractors [15] as well.

Lemma 4.2. *If M has a finite attractor then $\lim p_n^? = 0$.*

Another class of Markov chains for which PE scheme terminates is the class of chains which satisfy the following property.

Definition 4.3. *A Markov chain $M = (S, P)$ has δ -reachability property for $\delta > 0$ if $\forall s_1, s_2 \in S$ (s_2 is reachable from s_1) $\Rightarrow Prob_M(s_1 \xrightarrow{*} s_2) > \delta$.*

Lemma 4.4. *If M has δ -reachability property then $\lim p_n^? = 0$.*

Theorem 4.5. *The PE scheme terminates over the class of Markov chains with finite attractor and over the class of Markov chains with δ -reachability property.*

A variant of PE scheme was suggested in [10] for the following decision problem.

A decision problem for Probabilistic Reachability

Instance: A M.C. M , its states s_1, s_2 , $\theta > 0$ and p .

Question: Is $p - \theta < Prob_M(s_1 \xrightarrow{*} s_2) < p + \theta$?

It was claimed in [10] that Eq. (1) implies that (a) when the scheme terminates it produces a correct answer (b) it terminates for the Markov chains defined by

PLCS under the semantics of [10]. However, assertion (a) was incorrect. Also, the Markov chains assigned to PLCs in [10] do not have finite attractor property and the termination assertion (b) is unsound. It is an open question whether the above problem is decidable for the PLCs defined in Sect. 6 (or considered in [5,4,6]) which have finite attractor property.

PE scheme is conceptually very simple, however, no information about the number of iterations before it terminates can be extracted from Theorem 4.5. For finite state M. C. standard algebraic methods allow to find the exact value of $Prob_M(s_1 \xrightarrow{*} s_2)$ in polynomial time; however, in this case PE scheme finds an approximation in time $|M|^{\Omega(\ln(\frac{1}{\delta}))}$. An alternative approach for approximation of $Prob_{M'}(s_1 \xrightarrow{*} s_2)$ is to “approximate” a countable M.C. M by a finite state M.C. M' and then to evaluate $Prob_{M'}(s_1 \xrightarrow{*} s_2)$ by standard algebraic methods. Below is a simple transformation which allows to reduce the size of Markov chains.

Let $M = (S, P)$ and let $U \subseteq S$ and let u be a new state. The chain $M' = (S', P')$ which is obtained from M by collapsing U into an absorbing state u is denoted by $M^{U,u}$ and is defined as follows: $S' = S \setminus U \cup \{u\}$ and

$$P'(s, s') = \begin{cases} \sum_{d \in U} P(s, d) & \text{if } s \neq u \wedge s' = u, \\ P(s, s') & \text{if } s \neq u \wedge s' \neq u, \\ 1 & \text{if } s = u = s', \\ 0 & \text{otherwise.} \end{cases}$$

The following two lemmas are immediate, but useful for reductions of the size of M.C.

Lemma 4.6. *Let M be a M.C., let s_1, s_2 be states of M , let $u \notin S$, let C be a recurrent class such that $s_1 \notin C$ and let $M' = M^{C,u}$.*

1. *If $s_2 \in C$ then $Prob_M(s_1 \xrightarrow{*} s_2) = Prob_{M'}(s_1 \xrightarrow{*} u)$.*
2. *If $s_2 \notin C$ then $Prob_M(s_1 \xrightarrow{*} s_2) = Prob_{M'}(s_1 \xrightarrow{*} s_2)$.*

Lemma 4.7. *Let M be a M.C., let s_1, s_2 be states of M . Assume that $D \subseteq S \setminus \{s_1, s_2\}$ is such that either (1) $Prob\{\pi : \pi \text{ starts at } s_1 \text{ and reaches } D\} \leq \theta$ or*

(2) $\forall s \in D. Prob\{\pi : \pi \text{ starts at } s \text{ and reaches } s_2\} \leq \theta$. Let $M' = M^{D,d}$. Then $Prob_{M'}(s_1 \xrightarrow{*} s_2) \leq Prob_M(s_1 \xrightarrow{*} s_2) \leq Prob_{M'}(s_1 \xrightarrow{*} s_2) + \theta$.

In order to find D which satisfies the assumption of Lemma 4.7 we provide a reduction to a one-dimensional random walk. The following lemma is easily derived from standard properties of one dimensional random walks.

Lemma 4.8. *Let $M = (S, P)$ be a Markov chain where $S = \{0, 1, 2, 3, \dots\}$, and*

- $P(0, 0) = 1$.
- $P(i, i+1) = \nu_i$, $P(i, i-1) = \mu_i$, and $P(i, i) = 1 - \mu_i - \nu_i$, for $i \geq 1$.
- There is $q > 0.5$ such that $\mu_i > q$ for all $i \geq 2$.

Let $N(\mu_1, q, \theta) = \lceil \frac{\ln(1-\mu_1) - \ln(\mu_1 \cdot \theta)}{\ln q - \ln(1-q)} \rceil + 1$, where $\lceil x \rceil$ stands for the smallest integer which is greater than or equal to x . Then, for each $\theta > 0$ and $n \geq N(\mu_1, q, \theta)$, the probability of reaching a state n from 1 is less than θ .

The main technical lemma for the correctness and the complexity analysis of the algorithm presented in Sect. 7 is a generalization of Lemma 4.8.

Lemma 4.9 (Main Lemma). *Consider a Markov chain $M = (S, P)$ such that*

1. *S is the union of disjoint sets S_i ($i \in \mathcal{N}$).*
2. *If $s \in S_i$, $s' \in S_j$, and $P(s, s') > 0$, then $j \leq i + 1$.*
3. *$S_0 = C \cup R$ and*
 - *For every state $s \in R$, only states in R are reachable from s .*
 - *For every state $s \in S_1$ there is a finite path to R with the probability $> \delta$ which is inside $C \cup R$.*
4. *There is $\alpha < \frac{1}{2}$ such that $\nu_i + \gamma_i < \alpha$ for each $i \geq 2$, where*

$$\nu_i = \sup_{s \in S_i} \left(\sum_{s' \in S_{i+1}} P(s, s') \right) \text{ and } \gamma_i = \sup_{s \in S_i} \left(\sum_{s' \in S_i} P(s, s') \right).$$

Let $N_0 = N(\delta, 1 - \alpha, \theta)$, where N is defined as in Lemma 4.8. Then, for every $s \in S_0 \cup S_1$ the probability of reaching $\bigcup_{n \geq N_0} S_n$ from s is less than θ .

Hence,

Lemma 4.10. *Let M , S_i and N_0 be as in Lemma 4.9 and assume that $s_1, s_2 \in S_0$. Let $U = \bigcup_{n \geq N_0} S_n$. Let $M' = M^{U, u}$. Then $\text{Prob}_{M'}(s_1 \xrightarrow{*} s_2) \leq \text{Prob}_M(s_1 \xrightarrow{*} u) \leq \text{Prob}_{M'}(s_1 \xrightarrow{*} s_2) + \theta$.*

5 Lossy Channel Systems

In this section we consider lossy channel systems: processes with a finite set of local states operating on a number of unbounded and unreliable channels.

A *Lossy Channel System (LCS)* consists of a finite state process operating on a finite set of channels each of which behaves as a FIFO buffer which is unbounded and unreliable in the sense that it can nondeterministically lose messages. Formally, a *lossy channel system (LCS)* $\mathcal{L}$ is a tuple (S, C, M, T) where S is a finite set of *local states*, C is a finite set of *channels*, M is a finite *message alphabet*, and T is a set of *transitions* each of the form (s_1, op, s_2) , where $s_1, s_2 \in S$, and op is an *operation* of one of the forms $c!m$ (sending message m to channel c), or $c?m$ (receiving message m from channel c). A *global state* s is of the form (s, w) where $s \in S$ and w is a mapping from C to M^* .

For words $x, y \in M^*$, we use $x \bullet y$ to denote the concatenation of x and y . We write $x \preceq y$ to denote that x is a (not necessarily contiguous) substring of y . We use $|x|$ to denote the length of x , and use $x(i)$ to denote the i^{th} element of x where $i : 1 \leq i \leq |x|$. For $w_1, w_2 \in (C \mapsto M^*)$, we use $w_1 \preceq w_2$ to denote that $w_1(c) \preceq w_2(c)$ for each $c \in C$, and define $|w| = \sum_{c \in C} |w(c)|$. We also extend $\preceq$ to a relation on $S \times (C \mapsto M^*)$, where $(s_1, w_1) \preceq (s_2, w_2)$ iff $s_1 = s_2$ and $w_1 \preceq w_2$.

The LCS $\mathcal{L}$ induces a transition system $(S, \longrightarrow)$, where S is the set of global states, i.e., $S = (S \times (C \mapsto M^*))$, and $(s_1, w_1) \longrightarrow (s_2, w_2)$ iff one of the following conditions are satisfied:

- There is a $\mathbf{t} \in \mathbf{T}$, where $\mathbf{t}$ is of the form $(\mathbf{s}_1, \mathbf{c}!\mathbf{m}, \mathbf{s}_2)$ and $\mathbf{w}_2$ is the result of appending $\mathbf{m}$ to the end of $\mathbf{w}_1(\mathbf{c})$.
- There is a $\mathbf{t} \in \mathbf{T}$, where $\mathbf{t}$ is of the form $(\mathbf{s}_1, \mathbf{c}?\mathbf{m}, \mathbf{s}_2)$ and $\mathbf{w}_1$ is the result of removing $\mathbf{m}$ from the head of $\mathbf{w}_2(\mathbf{c})$.
- Furthermore, if $(\mathbf{s}_1, \mathbf{w}_1) \longrightarrow (\mathbf{s}_2, \mathbf{w}_2)$ according to one of the previous two rules, then $(\mathbf{s}_1, \mathbf{w}_1) \longrightarrow (\mathbf{s}'_2, \mathbf{w}'_2)$ for each $(\mathbf{s}'_2, \mathbf{w}'_2) \preceq (\mathbf{s}_2, \mathbf{w}_2)$.

In the first two cases we define $\mathbf{t}(\mathbf{s}_1, \mathbf{w}_1) = (\mathbf{s}_2, \mathbf{w}_2)$.

A transition $(\mathbf{s}_1, \mathbf{op}, \mathbf{s}_2)$ is said to be *enabled* at $(\mathbf{s}, \mathbf{w})$ if $s = s_1$ and either $\mathbf{op}$ is of the form $\mathbf{c}!\mathbf{m}$; or $\mathbf{op}$ is of the form $\mathbf{c}?\mathbf{m}$ and $\mathbf{w}(\mathbf{c}) = \mathbf{m} \bullet x$, for some $x \in \mathbf{M}^*$. We define $\text{enabled}(\mathbf{s}, \mathbf{w}) = \{\mathbf{t} : \mathbf{t} \text{ is enabled at } (\mathbf{s}, \mathbf{w})\}$.

Remark on notation. We use $\mathbf{s}$ and $\mathbf{S}$ to range over local states and sets of local states respectively. On the other hand, s and S range over states and sets of states of the induced transition system (states of the transition system are global states of the LCS).

A set $Q \subseteq S$ is said to be *upward closed* if $s_1 \in Q$ and $s_1 \preceq s_2$ imply $s_2 \in Q$. The upward closure $Q \uparrow$ of a set Q is the set $\{s : \exists s' \in Q. s' \preceq s\}$.

Theorems in [3,9] imply the following decidability results for LCS:

Lemma 5.1. (1) *It is decidable whether a state s_2 is reachable from a state s_1 .*
 (2) *It is decidable whether the upward closure of a finite set Q is reachable from a state s .* (3) *There is an algorithm $\text{Find-a-path}(s_1, s_2, \mathcal{L})$ which returns a path from s_1 to s_2 in the lossy channel system $\mathcal{L}$ or returns “No” if s_2 is not reachable from s_1 .* (4) *$\text{Graph}(A)$ is computable for every finite set of global states A of an LCS.*

6 Probabilistic Lossy Channel Systems

We introduce a probabilistic behaviour into LCS obtaining *Probabilistic Lossy Channel Systems*. This semantics was considered in [4,6] and differs from that in [10,5]

A *probabilistic lossy channel system (PLCS)* $\mathcal{L}$ is of the form $(\mathbf{S}, \mathbf{C}, \mathbf{M}, \mathbf{T}, \lambda, w)$, where $(\mathbf{S}, \mathbf{C}, \mathbf{M}, \mathbf{T})$ is an LCS, $\lambda \in (0, 1)$, and w is a mapping from $\mathbf{T}$ to the natural numbers. Intuitively, we derive a Markov chain from the PLCS $\mathcal{L}$ by assigning probabilities to the transitions of the underlying transition system $(\mathbf{S}, \mathbf{C}, \mathbf{M}, \mathbf{T})$. The probability of performing a transition $\mathbf{t}$ from a global state $(\mathbf{s}, \mathbf{w})$ is determined by the weight $w(\mathbf{t})$ of $\mathbf{t}$ compared to the weights of the other transitions which are enabled at $(\mathbf{s}, \mathbf{w})$. Furthermore, after performing each transition, each message which resides inside one of the channels may be lost, independently of the other messages, with the probability λ . This means that the probability of the transition from $(\mathbf{s}_1, \mathbf{w}_1)$ to $(\mathbf{s}_2, \mathbf{w}_2)$ is equal to (the sum over all $(\mathbf{s}_3, \mathbf{w}_3)$ of) the probability of reaching some $(\mathbf{s}_3, \mathbf{w}_3)$ from $(\mathbf{s}_1, \mathbf{w}_1)$ through performing a transition $\mathbf{t}$ of the underlying LCS, multiplied by the probability of reaching $(\mathbf{s}_2, \mathbf{w}_2)$ from $(\mathbf{s}_3, \mathbf{w}_3)$ through the loss of messages (see [4] for detailed calculations of the probabilities of the transitions).

To simplify the presentation, we assume from now on that PLCSs have no deadlock states, i.e., from every state a transition is enabled. The only probabilis-

tic properties of PLCSs which we use are summarized in the next two lemmas from [4].

Lemma 6.1. *Let s be a state with m messages. The probability of the transitions from s to the set of states with $> m + 1$ messages is 0. The probability of the transitions from s to the set of states with $m + 1$ messages is $\leq (1 - \lambda)^{m+1}$. The probability of the transitions from s to the set of states with m messages is $\leq m\lambda(1 - \lambda)^m$.*

Lemma 6.2. *For each λ , w , and PLCS (S, C, M, T, λ, w) the set of the states with the empty set of messages is a finite attractor.*

The next lemma plays a key role in the algorithm presented in Sect. 7

Lemma 6.3. *For each PLCS $\mathcal{L} = (S, C, M, T, \lambda, w)$ there are $V_1, \dots, V_k$ such that V_i are finite sets of global states and k is the number of the recurrent classes of $\mathcal{L}$ and for each state s : s is in the i -th recurrent class of $\mathcal{L}$ iff s is not in the upward closure of V_i . Moreover, $V_1, \dots, V_k$ are computable from the underlying LCS (S, C, M, T) .*

7 Algorithm for Approximating the Probability of Reachability

Lemmas 6.2, 5.1(1) and Theorem 4.5 imply that there is an algorithm based on PE scheme for the quantitative probabilistic reachability problem. However, no information about the complexity of this algorithm can be extracted from Theorem 4.5. In this section we provide an algorithm with a parametric complexity $f(\mathcal{L}, s_1, s_2) \times \frac{1}{\theta^3}$ for the quantitative probabilistic reachability problem.

The idea of the algorithm is to take the set $B_{\leq n}$ of states with at most n messages of the Markov chain M generated by PLCS $\mathcal{L}$. Construct a finite Markov chain $\widehat{M}$ by restricting the transition of M to $B_{\leq n}$, and then for each recurrence class D_i of M collapse the set of D_i states in $B_{\leq n}$ into one state of $\widehat{M}$. Finally, calculate the probability of reaching s_2 from s_1 in the finite M.C. $\widehat{M}$. The crucial fact in the correctness of our algorithm is that relying on Lemma 4.9 we can compute n big enough which will ensure that the probability of reaching s_2 from s_1 in the finite Markov chain $\widehat{M}$ approximates up to θ the probability of reaching s_2 from s_1 in the infinite Markov chain M .

In the rest of this section we describe the algorithm with a justification of its correctness and provide an analysis of its complexity.

Algorithm – for Quantitative Probabilistic Reachability Problem

Input PLCS $\mathcal{L} = (S, C, M, T, \lambda, w)$ with an underlying Markov chain $M = (S, P)$, states $s_1, s_2 \in S$, and a rational θ .

Output: a rational r such that $r \leq \text{Prob}_M(s_1 \xrightarrow{*} s_2) \leq r + \theta$.

Let A be the (finite) set of all states with 0 messages. A is an attractor by Lemma 6.2. By Lemma 5.1(4) we can construct $\text{Graph}(A)$. Then we can find the bottom

strongly connected components $C_1, \dots, C_k$ in $\text{Graph}(A)$ and for $1 \leq i \leq k$ by Lemma 3.3 and by Lemma 6.3 we can compute finite sets of states V_i such that

$$\forall s \in S (s \text{ is in the recurrent class of } C_i) \text{ iff } s \text{ is not in the upward closure of } V_i \quad (2)$$

Hence, we can check whether s_1 (or any other state s) is in the i -th recurrent class.

In the case when s_1 is recurrent we proceed as follows: If s_2 is recurrent and in the same recurrent class as s_1 then output 1 else output 0. (The correctness of this answer follows by Lemma 3.1(4-5).)

Below we consider the case when s_1 is not recurrent. By Lemma 5.1(3) we can find l such that for every $u, v \in A \cup \{s_1, s_2\}$ if u is reachable from v then there is a path from v to u which passes only through nodes with at most l messages. Let m be such that $\forall n. m \leq n \rightarrow (1 - \lambda)^n (1 - \lambda + n\lambda) < \frac{1}{3}$ i.e., the probability to move from every state with $n \geq m$ messages to the set of states with at least n messages is less than $\frac{1}{3}$ (by Lemma 6.1). Let $h = \max(l, m) + 1$.

Notations: Below we denote by B_i (respectively by $B_{\leq i}$) the set of states with i (respectively with at most i) messages.

For every state $s \in B_{\leq h}$ there is a path π_s which first chooses a lossy transition which leads to a state s' with 0 messages and then follows by a path from s' which is inside $B_{\leq l} \subseteq B_{\leq h}$ to a BSCC of $\text{Graph}(A)$. Let $\delta_s = \text{Prob}(\pi_s) > 0$ and let $0 < \delta = \min(\delta_s : s \in B_{\leq h})$. Note that up to this point all our computations were independent of θ and their complexity depended only on $\mathcal{L}$, s_1 and s_2 .

Observe that if we denote by R the set of recurrent states of M , by C the set of transient states with $< h$ messages; by S_0 the set $R \cup C$ and by S_i ($i > 0$) the set of transient states with $h + i - 1$ messages, then the assumptions of Lemma 4.9 are satisfied. Let $N_0 = N(\delta, \frac{2}{3}, \theta)$, where N is the function from Lemma 4.8 and let $n = h + N_0$. Note that n depends linearly on $\ln(\frac{1}{\theta})$. By Lemma 4.9 the probability to reach from s_1 the set $U = \bigcup_{n \geq N_0} S_n$ of transient states with $\geq n$ messages is at most θ . Therefore, by Lemma 4.10 we derive that $\text{Prob}_{M'}(s_1 \xrightarrow{*} s_2) \leq \text{Prob}_M(s_1 \xrightarrow{*} s_2) \leq \text{Prob}_{M'}(s_1 \xrightarrow{*} s_2) + \theta$ for $M' = M^{U, \text{dead}}$ obtained by collapsing U into a fresh state *dead*. The chain M' might be infinite. Below we are going to construct a finite state M.C. $\widehat{M}$ of size bounded by $|B_{\leq n}|$ such that $\text{Prob}_{M'}(s_1 \xrightarrow{*} s_2) = \text{Prob}_{\widehat{M}}(s_1 \xrightarrow{*} s_2)$. Hence, $\text{Prob}_{\widehat{M}}(s_1 \xrightarrow{*} s_2)$ will approximate up to θ the value of $\text{Prob}_M(s_1 \xrightarrow{*} s_2)$ which we are trying to compute.

The complexity of the construction of $\widehat{M}$ will be $O(|B_{\leq n}|^2)$ and by standard algebraic methods we can compute $\text{Prob}_{\widehat{M}}(s_1 \xrightarrow{*} s_2)$ in time $O(|B_{\leq n}|^3)$. Since n depends linearly on $\ln(\frac{1}{\theta})$, it follows that $|B_{\leq n}|$ depends linearly on $\frac{1}{\theta}$ and the complexity of the entire algorithm is $f(\mathcal{L}, s_1, s_2) \times \frac{1}{\theta^3}$.

We define $\widehat{M}$ by replacing every recurrent class of M' by an absorbing state. From Lemma 4.6 we will derive that this transformation preserves the probability of reaching s_2 from s_1 . Formally a (finite) M. C. $\widehat{M} = (\widehat{S}, \widehat{P})$ is defined as follows.

Let D_i ($i = 1, \dots, k$) be the states with $\leq n$ messages, which are in the i -th recurrent class. (These sets can be computed by By Eq. (2) in time $O(|B_{\leq n}|)$. Let D be $B_{< n} \setminus \cup D_i$.

1. $\widehat{S} = D \cup \{d_1, \dots, d_k\} \cup \{dead\}$. The states d_i correspond to the recurrent classes and the state $dead$ corresponds to the set of transient states of M with $\geq n$ messages.
2. The states $d_1, \dots, d_k, dead$ are absorbing, i.e., for $d \in \{d_1, \dots, d_k\} \cup \{dead\}$:

$$P'(d, d') = \begin{cases} 1 & \text{if } d' = d, \\ 0 & \text{otherwise.} \end{cases}$$

3. $\widehat{P}(d, d') = P(d, d')$ for $d, d' \in D$
4. $\widehat{P}(d, dead) = \sum_{s \in B_n \setminus \cup D_i} P(d, s)$ for $d \in D$.
5. $\widehat{P}(d, d_i) = \sum_{d' \in D_i} P(d, d')$, for $d \in D$ and $i : 1 \leq i \leq k$.

Recall that we treated already the case when s_1 is recurrent, hence s_1 is in D . Compute the output r which approximates $Prob_M(s_1 \xrightarrow{*} s_2)$ up to θ by the following cases:

1. if $s_2 \in D$ then compute by standard algebraic methods the probability r of reaching s_2 from s_1 in (a finite Markov chain) $\widehat{M}$.
2. if $s_2 \in D_i$ then compute the probability r of reaching d_i from s_1 in $\widehat{M}$.

We completed the presentation of the algorithm, established its correctness and proved that its complexity is $f(\mathcal{L}, s_1, s_2) \times \frac{1}{\theta^3}$. It was shown in [14] that the complexity of the reachability problem for LCSs is not bounded by any primitive recursive function in the size of LCS. Therefore, f is not primitive recursive in the size of PLCS.

8 Probability of Automata Definable Properties

In this section we consider more general properties than reachability. Let φ be a property of computations. We will be interested in approximating

$$Prob\{\pi : \pi \text{ is a computation from } s \text{ in PLCS } \mathcal{L} \text{ and } \pi \text{ satisfies } \varphi\}.$$

We show that if the properties of computations are specified by (the ω -behavior of) finite state automata or equivalently by formulas of the monadic second-order logic of order, then the above problem is computable.

We consider an extension of PLCS by adding a labeling function. A *state labeled* PLCS is an PLCS together with a finite alphabet Σ and a labeling function lab from the local states to Σ . We lift the labeling to the global states: the label of every global state is the same as the label of its local state component. Similarly, with a computation $s_0, s_1, \dots$ we associate the ω -string $lab(s_1)lab(s_2)\dots$ over the alphabet Σ .

The next lemma reduces the Quantitative Probabilistic Model-checking Problem to the Quantitative Probabilistic Reachability Problem for PLCSs (see Sect. 1).

Lemma 8.1. *There exists an algorithm which for a state labeled PLCS $\mathcal{L}$, a global state s in $\mathcal{L}$ and a finite state ω -automaton $\mathcal{A}$ produces a PLCS $\mathcal{L}'$, a global state s' in $\mathcal{L}'$ and a set $C_1, C_2, \dots, C_p$ of BSCC for a finite attractor of $\mathcal{L}'$ such that the following are equivalent:*

1. *The probability that a computation of $\mathcal{L}$ that starts at s is accepted by $\mathcal{A}$ is r .*
2. *The probability that a computation of $\mathcal{L}'$ that starts at s' reaches $\bigcup_{i=1}^p C_i$ is r .*

Theorem 8.2. *The Quantitative Probabilistic Model-checking problem can be solved in time $g(\mathcal{L}, \mathcal{A}, s_1, s_2) \times \frac{1}{\theta^3}$.*

Proof. First apply the algorithm from Lemma 8.1. Observe that for $i \neq j$ no path reaches both C_i and C_j . For $i = 1, \dots, p$ choose a state $s_i \in C_i$. Lemma 3.1(4) and Lemma 3.3 imply that the probability to reach s_i is the same as the probability to reach C_i . By the algorithm of Sect. 7 compute r_i which approximates up to $\frac{\theta}{p}$ the probability to reach s_i from s' in $\mathcal{L}'$. From Lemma 8.1 it follows that $r_1 + r_2 + \dots + r_p$ approximates up to θ the probability that a computation of $\mathcal{L}$ that starts at s is accepted by $\mathcal{A}$.

Acknowledgments. We would like to thank Philippe Schnoebelen and an anonymous referee for pointing out that the path enumeration scheme terminates over Markov chains with finite attractors. We thank Parosh Abdulla, Danièle Beauquier, Philippe Schnoebelen and Anatol Slissenko for fruitful discussions and their useful comments.

References

1. P. A. Abdulla, C. Baier, P. Iyer, and B. Jonsson. Reasoning about probabilistic lossy channel systems. In *Proc. CONCUR 2000*, volume 1877 of *Lecture Notes in Computer Science*, 2000.
2. P. A. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996.
3. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
4. P. A. Abdulla and A. Rabinovich. Verification of probabilistic systems with faulty communication, 2003. In FOSSACS'03, volume 2620 of *LNCS*, pages 39–53. Springer Verlag, 2003.
5. C. Baier and B. Engelen. Establishing qualitative properties for probabilistic lossy channel systems. In *ARTS'99*, volume 1601 of *LNCS*, pages 34–52. Springer Verlag, 1999.
6. N. Bertrand and Ph. Schnoebelen. Model checking lossy channels systems is probably decidable. In FOSSACS'03, volume 2620 of *LNCS*, pages 120–135 Springer Verlag, 2003.
7. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 2(5):323–342, 1983.
8. Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.

9. A. Finkel and Ph. Schnoebelen. Well structured transition systems everywhere!. *Theoretical Computer Science*, 256(1-2):63-92, 2001.
10. P. Iyer and M. Narasimha. Probabilistic Lossy Channel Systems. In Proc of *TAP-SOFT '97* LNCS 1214, 667-681 1997.
11. S. Karlin. *A First Course in Stochastic Processes*. Academic Press, 1966.
12. J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. D Van Nostrand Co., 1966.
13. J. R. Norris. *Markov Chains*. Cambridge University Press, 1997.
14. Ph. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251-261, 2002.
15. Ph. Schnoebelen. Personal communication, Jan. 2003.

Discounting the Future in Systems Theory^{*}

Luca de Alfaro¹, Thomas A. Henzinger², and Rupak Majumdar²

¹ Department of Computer Engineering, UC Santa Cruz

`luca@soe.ucsc.edu`

² Department of Electrical Engineering and Computer Sciences, UC Berkeley

`{tah, rupak}@eecs.berkeley.edu`

Abstract. Discounting the future means that the value, today, of a unit payoff is 1 if the payoff occurs today, a if it occurs tomorrow, a^2 if it occurs the day after tomorrow, and so on, for some real-valued discount factor $0 < a < 1$. Discounting (or inflation) is a key paradigm in economics and has been studied in Markov decision processes as well as game theory. We submit that discounting also has a natural place in systems engineering: for nonterminating systems, a potential bug in the far-away future is less troubling than a potential bug today. We therefore develop a systems theory with discounting. Our theory includes several basic elements: discounted versions of system properties that correspond to the ω -regular properties, fixpoint-based algorithms for checking discounted properties, and a quantitative notion of bisimilarity for capturing the difference between two states with respect to discounted properties. We present the theory in a general form that applies to probabilistic systems as well as multicomponent systems (games), but it readily specializes to classical transition systems. We show that discounting, besides its natural practical appeal, has also several mathematical benefits. First, the resulting theory is robust, in that small perturbations of a system can cause only small changes in the properties of the system. Second, the theory is computational, in that the values of discounted properties, as well as the discounted bisimilarity distance between states, can be computed to any desired degree of precision.

1 Introduction

In systems theory, one models systems and analyzes their properties. Nonterminating discrete-time models, such as transition systems and games, are important in many computer science applications, and the ω -regular properties offer an accomplished theory for their analysis. The theory is expressive from a practical point of view [22,27], computational (algorithmic) [5,28], and abstract (language-independent) [21,34]. In its general setting, the theory considers games with ω -regular winning conditions [17,28], provides fixpoint-based algorithms for their solution [13,15], and property-preserving equivalence relations

^{*} This research was supported in part by the NSF CAREER award CCR-0132780, the DARPA grant F33615-C-98-3614, the NSF grants CCR-9988172, CCR-0234690 and CCR-0225610, and the ONR grant N00014-02-1-0671.

between structures [4,24]. From a systems engineering point of view, however, the theory has a significant drawback: it is too exact [1]. Since the ω -regular properties generalize finite behavior by considering behavior at infinity, they can distinguish behavior differences that occur arbitrarily late. This exactness becomes even more pronounced for probabilistic models [6,29,33], whose behaviors are specified using numerical quantities, because the theory can distinguish arbitrarily small perturbations of a system.

We propose an alternative formalism that is (in a certain sense) as expressive as the ω -regular properties, and yet achieves continuity in the Cantor topology by sacrificing exactness. In other words, we introduce an *approximate* theory of nonterminating discrete-time systems. The approximation is in two directions. First, instead of giving boolean answers to logical questions, we consider the value of a property to be a real number in the interval $[0,1]$ [19]. Second, we generalize, as in [11,12,18], the classical notions of state equivalences to pseudometrics on states. Both are achieved by defining a *discounted* version of the classical theory. Discounting is inspired by similar ideas in Markov decision processes, economics, and game theory [16,31], and captures the natural engineering intuition that the far-away future is not as important as the near future. Consider, for example, the safety property that no unsafe state is visited. In the classical theory, this property is either true or false. In the discounted theory, its value is 1 if no unsafe state is visited ever, and $1 - a^k$, for some discount factor $0 < a < 1$, if no unsafe state is visited for k steps: the longer the system stays in safe states, the greater the value of the property. Our theory is *robust*, in that small perturbations of a system imply small differences in the numerical values of properties, and *computational*, in that numerical approximation schemes are available which converge geometrically to property values from both directions.

The key insight of this work is that discounting is most naturally and fundamentally applied not to properties, nor to state equivalences, but to the μ -calculus [20]. We introduce the *discounted μ -calculus*, a *quantitative* fixpoint calculus: rather than computing with sets of states, as the traditional μ -calculus does, we compute with functions that assign to each state a value between 0 and 1. A quantitative μ -calculus was introduced in [9] to compute the values of probabilistic ω -regular games by iterating a quantitative version of the predecessor (pre) operator. The discounted μ -calculus is obtained from the calculus of [9] by discounting the pre operator through multiplication with a discount factor $a < 1$. In the classical setting, there is a connection between (linear-time) ω -regular properties, (branching-time) μ -calculus, and games. By discounting the μ -calculus while maintaining this connection, we obtain a notion of discounted ω -regular properties, as well as algorithms for solving games with discounted ω -regular objectives. In the classical setting, the connection is as follows. The solution of a game with an ω -regular winning condition can be written as a μ -calculus formula [13,14]. The fixpoint formula defines the property: when evaluated on linear traces, it holds exactly on the initial states of the traces that satisfy the property. We extend this correspondence to the discounted setting by considering discounted versions of the μ -calculus formula: the discounted fixpoint

formula, evaluated on linear traces, defines a discounted version of the original ω -regular property. At the same time, we show that the discounted formula, when evaluated on a game structure, computes the value of the game whose payoff is given by the discounted ω -regular property.

We develop our theory on the system model of concurrent probabilistic game structures [9,16]. These structures generalize several standard models of computation, including nondeterministic transition systems, Markov decision processes [10], and deterministic two-player games [2,32]. The use of discounting gives our theory two main features: *computationality* and *robustness*. Computationality is due to the fact that discount factors strictly less than 1 ensure the geometric convergence of each fixpoint computation by successive approximation (Picard iteration). This enables us to compute every fixpoint value to any desired degree of precision. Moreover, discounting entails the uniqueness of fixpoints. Together, the monotonicity of the μ -calculus operators, the geometric convergence of Picard iteration, and the uniqueness of fixpoints mean that we can iteratively compute geometrically converging lower and upper bounds for the value of every discounted μ -calculus formula. The existence of such approximation schemes is in sharp contrast to the situation for the undiscounted μ -calculus, where least and greatest fixpoints generally differ, where each (least or greatest) fixpoint can be approximated in one direction only (from below, or from above), and where in the quantitative case, no rate of convergence is known.

In the classical setting, the μ -calculus characterizes bisimilarity: two states are bisimilar iff they satisfy the same μ -calculus formulas. To extend this connection to the discounted setting, we define a quantitative, discounted notion of bisimilarity, which assigns a real-valued *distance* in the interval $[0,1]$ to every pair of states: the distance between two states is 1 if they satisfy different propositions, and otherwise it is coinductively computed from discounted distances between successor states. We show that in the discounted setting, the bisimilarity distance between two states is equal to the supremum, over all μ -calculus formulas, of the difference between the values of a formula at the two states. This is in fact the characterization of discounted bisimilarity from [11,12] extended to games. However, while in [11,12] the above characterization is taken to be the definition of discounted bisimilarity, in our case it is a theorem that can be proved from the coinductive definition. The theorem demonstrates the *robustness* of the theory: small perturbations in the numerical values of transition probabilities, as well as (small or large) perturbations that come far in the future, correspond to small bisimilarity distance, and hence to small differences in the numerical values of discounted properties. The numerical computation of discounted bisimilarity by successive approximation enjoys the same properties as the numerical evaluation of discounted μ -calculus formulas; in particular, geometrically-converging approximation schemes are available for computing both lower and upper bounds.

2 Systems: Concurrent Game Structures

For a countable set U , a *probability distribution* on U is a function $p: U \mapsto [0, 1]$ such that $\sum_{u \in U} p(u) = 1$. We write $\mathcal{D}(U)$ for the set of probability distributions on U . A two-player (*concurrent*) *game structure* [2,7] $\mathcal{G} = \langle Q, M, \Gamma_1, \Gamma_2, \delta \rangle$ consists of the following components:

- A finite set Q of states.
- A finite set M of moves.
- Two move assignments $\Gamma_1, \Gamma_2: Q \mapsto 2^M \setminus \emptyset$. For $i \in \{1, 2\}$, the assignment Γ_i associates with each state $s \in Q$ the nonempty set $\Gamma_i(s) \subseteq M$ of moves available to player i at state s .
- A probabilistic transition function $\delta: Q \times M^2 \mapsto \mathcal{D}(Q)$. For a state $s \in Q$ and moves $\gamma_1 \in \Gamma_1(s)$ and $\gamma_2 \in \Gamma_2(s)$, the function δ provides a probability distribution of successor states. We write $\delta(t \mid s, \gamma_1, \gamma_2)$ for the probability $\delta(s, \gamma_1, \gamma_2)(t)$ that the successor state is $t \in Q$.

At every state $s \in Q$, player 1 chooses a move $\gamma_1 \in \Gamma_1(s)$, and simultaneously and independently player 2 chooses a move $\gamma_2 \in \Gamma_2(s)$. The game then proceeds to the successor state $t \in Q$ with probability $\delta(t \mid s, \gamma_1, \gamma_2)$. The outcome of the game is a path. A *path* of $\mathcal{G}$ is an infinite sequence $s_0, s_1, s_2, \dots$ of states in $s_k \in Q$ such that for all $k \geq 0$, there are moves $\gamma_1^k \in \Gamma_1(s_k)$ and $\gamma_2^k \in \Gamma_2(s_k)$ with $\delta(s_{k+1} \mid s_k, \gamma_1^k, \gamma_2^k) > 0$. We write Σ for the set of all paths.

The following are special cases of concurrent game structures. The structure $\mathcal{G}$ is *deterministic* if for all states $s \in Q$ and moves $\gamma_1 \in \Gamma_1(s)$, $\gamma_2 \in \Gamma_2(s)$, there is a state $t \in Q$ with $\delta(t \mid s, \gamma_1, \gamma_2) = 1$; in this case, with abuse of notation we write $\delta(s, \gamma_1, \gamma_2) = t$. The structure $\mathcal{G}$ is *turn-based* if at every state at most one player can choose among multiple moves; that is, for all states $s \in Q$, there exists at most one $i \in \{1, 2\}$ with $|\Gamma_i(s)| > 1$. The turn-based deterministic game structures coincide with the games of [32]. The structure $\mathcal{G}$ is *one-player* if at every state only player 1 can choose among multiple moves; that is, $|\Gamma_2(s)| = 1$ for all states $s \in Q$. The one-player game structures coincide with Markov decision processes (MDPs) [10]. The one-player deterministic game structures coincide with transition systems: in every state, each available move of player 1 determines a possible successor state.

A *strategy* for player $i \in \{1, 2\}$ is a function $\pi_i: Q^+ \mapsto \mathcal{D}(M)$ that associates with every nonempty finite sequence $\sigma \in Q^+$ of states, representing the history of the game, a probability distribution $\pi_i(\sigma)$, which is used to select the next move of player i . Thus, the choice of the next move can be history-dependent and randomized. We require that the strategy π_i can prescribe only moves that are available to player i ; that is, for all sequences $\sigma \in Q^*$ and states $s \in Q$, if $\pi_i(\sigma s)(\gamma) > 0$, then $\gamma \in \Gamma_i(s)$. We write Π_i for the set of strategies for player i . The strategy π_i is *deterministic* if for all sequences $\sigma \in Q^+$, there exists a move $\gamma \in M$ such that $\pi(\sigma)(\gamma) = 1$. Thus, deterministic strategies are functions from Q^+ to M . The strategy π_i is *memoryless* if for all sequences $\sigma, \sigma' \in Q^*$ and states $s \in Q$, we have $\pi(\sigma s) = \pi(\sigma' s)$. Thus, the moves chosen by memoryless strategies depend only on the current state and not on the history of the game.

Given a starting state $s \in Q$ and two strategies π_1 and π_2 for the two players, the game is reduced to an ordinary stochastic process, denoted $\mathcal{G}_s^{\pi_1, \pi_2}$, which defines a probability distribution on the set Σ of paths. An *event* of $\mathcal{G}_s^{\pi_1, \pi_2}$ is a measurable set $A \subseteq \Sigma$ of paths. For an event $A \subseteq \Sigma$, we write $\Pr_s^{\pi_1, \pi_2}(A)$ for the probability that the outcome of the game belongs to A when the game starts from s and the players use the strategies π_1 and π_2 . A *payoff function* $v: \Sigma \mapsto [0, 1]$ is a measurable function that associates with every path a real in the interval $[0, 1]$. Payoff functions define the rewards of the two players for each outcome of the game. For a payoff function v , we write $E_s^{\pi_1, \pi_2}\{v\}$ for the expected value of v on the outcome when the game starts from s and the strategies π_1 and π_2 are used. If v defines the reward for player 1, then the (*player 1*) *value* of the game is a function that maps every state $s \in Q$ to the maximal expected reward $\sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} E_s^{\pi_1, \pi_2}\{v\}$ that player 1 can achieve no matter which strategy player 2 chooses.

3 Algorithms: Discounted Fixpoint Expressions

Quantitative region algebra. The classical μ -calculus specifies algorithms for iterating boolean and predecessor (pre) operators on regions, where a region is a set of states. In our case a region is a function from states to reals. This notion of quantitative region admits the analysis both of probabilistic transitions and of real-valued discount factors. Consider a concurrent game structure $\mathcal{G} = \langle Q, M, \Gamma_1, \Gamma_2, \delta \rangle$. A (*quantitative*) *region* of $\mathcal{G}$ is a function $f: Q \mapsto [0, 1]$ that maps every state to a real in the interval $[0, 1]$. For example, for a given payoff function, the value of a game on the structure $\mathcal{G}$ is a quantitative region. We write $\mathcal{F}$ for the set of quantitative regions. By **0** and **1** we denote the constant functions in $\mathcal{F}$ that map all states in Q to 0 and 1, respectively. Given two regions $f, g \in \mathcal{F}$, define $f \leq g$ if $f(s) \leq g(s)$ for all states $s \in Q$, and define the regions $f \wedge g$ and $f \vee g$ by $(f \wedge g)(s) = \min\{f(s), g(s)\}$ and $(f \vee g)(s) = \max\{f(s), g(s)\}$, for all states $s \in Q$. The region $\mathbf{1} - f$ is defined by $(\mathbf{1} - f)(s) = 1 - f(s)$ for all $s \in Q$; this has the role of complementation. Given a set $T \subseteq Q$ of states, with abuse of notation we denote by T also the indicator function of T , defined by $T(s) = 1$ if $s \in Q$, and $T(s) = 0$ otherwise. Let $\mathcal{F}_{\mathbb{B}} \subseteq \mathcal{F}$ be the set of indicator functions (also called *boolean regions*). Note that in $\mathcal{F}_{\mathbb{B}}$, the operators $\wedge$, $\vee$, and $\leq$ correspond respectively to intersection, union, and set inclusion.

An operator $F: \mathcal{F} \mapsto \mathcal{F}$ is *monotonic* if for all regions $f, g \in \mathcal{F}$, if $f \leq g$, then $F(f) \leq F(g)$. The operator F is *Lipschitz continuous* if for all regions $f, g \in \mathcal{F}$, we have $|F(f) - F(g)| \leq |f - g|$, where $|\cdot|$ is the L_∞ norm. Note that Lipschitz continuity implies continuity: for all infinite increasing sequences $f_1 \leq f_2 \leq \dots$ of regions in $\mathcal{F}$, we have $\lim_{n \rightarrow \infty} F(f_n) = F(\lim_{n \rightarrow \infty} f_n)$. The operator F is *contractive* if there exists a constant $0 < c < 1$ such that for all regions $f, g \in \mathcal{F}$, we have $|F(f) - F(g)| \leq c \cdot |f - g|$. For $i \in \{1, 2\}$, we consider so-called *pre operators* $\text{Pre}_i: \mathcal{F} \mapsto \mathcal{F}$ with the following properties: (1) Pre_1 and Pre_2 are monotonic and Lipschitz continuous, and (2) for all regions $f \in \mathcal{F}$, we have $\text{Pre}_1(f) = \mathbf{1} - \text{Pre}_2(\mathbf{1} - f)$; that is, the operators Pre_1 and Pre_2 are

dual. The following pre operators have natural interpretations on (subclasses of) concurrent game structures. The *quantitative pre operator* [9] $\text{Qpre}_1: \mathcal{F} \mapsto \mathcal{F}$ is defined for every quantitative region $f \in \mathcal{F}$ and state $s \in Q$ by

$$\text{Qpre}_1(f)(s) = \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} E_s^{\pi_1, \pi_2} \{v_{\circ f}\},$$

where $v_{\circ f}: \Sigma \mapsto [0, 1]$ is the payoff function that maps every path $s_0, s_1, \dots$ in Σ to the value $f(s_1)$ of f at the second state of the path. In words, $\text{Qpre}_1(f)(s)$ is the maximal expectation for the value of f that player 1 can achieve in a successor state of s . The value $\text{Qpre}_1(f)(s)$ can be computed by solving a matrix game:

$$\text{Qpre}_1(f)(s) = \text{val}_1 \left[\sum_{t \in Q} f(t) \cdot \delta(t \mid s, \gamma_1, \gamma_2) \right]_{\gamma_1 \in \Gamma_1(s), \gamma_2 \in \Gamma_2(s)}$$

where $\text{val}_1[\cdot]$ denotes the player 1 value (i.e., maximal expected reward for player 1) of a matrix game. The minmax theorem guarantees that this matrix game has optimal strategies for both players [35]. The matrix game can be solved by linear programming [9, 26]. The player 2 operator Qpre_2 is defined symmetrically. The minmax theorem permits the exchange of the sup and inf in the definition, and thus ensures the duality of the two pre operators.

By specializing the quantitative pre operators Qpre_i to turn-based deterministic game structures, we obtain the *controllable pre operators* [2] $\text{Cpre}_i: \mathcal{F}_{\mathbb{B}} \mapsto \mathcal{F}_{\mathbb{B}}$, which are closed on boolean regions. In particular, for every boolean region $f \in \mathcal{F}_{\mathbb{B}}$ and state $s \in Q$, $\text{Cpre}_1(f)(s) = 1$ iff $\exists \gamma_1 \in \Gamma_1(s). \forall \gamma_2 \in \Gamma_2(s). f(\delta(s, \gamma_1, \gamma_2)) = 1$. In words, for a set $T \subseteq Q$ of states, $\text{Cpre}_1(T)$ is the set of states from which player 1 can ensure that the next state lies in T . For one-player game structures, this characterization further simplifies to $\text{Epre}_1(f)(s) = 1$ iff $\exists \gamma_1 \in \Gamma_1(s). f(\delta(s, \gamma_1, \cdot)) = 1$. This is the traditional definition of the existential pre operator on a transition system: for a set $T \subseteq Q$ of states, $\text{Epre}_1(T)$ is the set of predecessor states.

Discounted μ -calculus. We define a fixpoint calculus that permits the iteration of pre operators. The calculus is discounted, in that every occurrence of a pre operator is multiplied by a *discount factor* from $[0, 1]$. If the discount factor of a pre operator is less than 1, this has the effect that each additional application of the operator in a fixpoint iteration carries less weight. We use a fixed set Θ of *propositions*; every proposition $T \in \Theta$ denotes a boolean region $\llbracket T \rrbracket \in \mathcal{F}_{\mathbb{B}}$. For a state $s \in Q$ with $\llbracket T \rrbracket(s) = 1$, we write $s \models T$ and say that s is a *T-state*. The formulas of the *discounted μ -calculus* are generated by the grammar

$$\begin{aligned} \phi ::= & T \mid \neg T \mid x \mid \phi \vee \phi \mid \phi \wedge \phi \mid \alpha \cdot \text{pre}_1(\phi) \mid \alpha \cdot \text{pre}_2(\phi) \mid \\ & (1 - \alpha) + \alpha \cdot \text{pre}_1(\phi) \mid (1 - \alpha) + \alpha \cdot \text{pre}_2(\phi) \mid \mu x. \phi \mid \nu x. \phi \end{aligned}$$

for propositions $T \in \Theta$, variables x from some fixed set X , and parameters α from some fixed set Λ . The syntax defines formulas in positive normal form. The

definition of negation in the calculus, which is given below, makes it clear that we need two discounted pre modalities, $\alpha \cdot \text{pre}_i(\cdot)$ and $(1 - \alpha) + \alpha \cdot \text{pre}_i(\cdot)$, for each player $i \in \{1, 2\}$. A formula ϕ is *closed* if every variable x in ϕ occurs in the scope of a least-fixpoint quantifier μx or greatest-fixpoint quantifier νx .

A variable valuation $\mathcal{E}: X \mapsto \mathcal{F}$ is a function that maps every variable $x \in X$ to a quantitative region in $\mathcal{F}$. We write $\mathcal{E}[x \mapsto f]$ for the function that agrees with $\mathcal{E}$ on all variables, except that x is mapped to f . A formula may contain several different discount factors. A parameter valuation $\mathcal{P}: \Lambda \mapsto [0, 1]$ is a function that maps every parameter $\alpha \in \Lambda$ to a real-valued discount factor in the interval $[0, 1]$. Given a real $r \in [0, 1]$, the parameter valuation $\mathcal{P}$ is *r-bounded* if $\mathcal{P}(\alpha) \leq r$ for all parameters $\alpha \in \Lambda$. An *interpretation* is a pair that consists of a variable valuation and a parameter valuation. Given an interpretation $(\mathcal{E}, \mathcal{P})$, every formula ϕ of the discounted μ -calculus defines a quantitative region $\llbracket \phi \rrbracket_{\mathcal{E}, \mathcal{P}}^{\mathcal{G}} \in \mathcal{F}$ (the superscript $\mathcal{G}$ is omitted if the game structure is clear from the context):

$$\begin{aligned} \llbracket T \rrbracket_{\mathcal{E}, \mathcal{P}} &= \llbracket T \rrbracket \\ \llbracket \neg T \rrbracket_{\mathcal{E}, \mathcal{P}} &= \mathbf{1} - \llbracket T \rrbracket \\ \llbracket x \rrbracket_{\mathcal{E}, \mathcal{P}} &= \mathcal{E}(x) \\ \llbracket \alpha \cdot \text{pre}_i(\phi) \rrbracket_{\mathcal{E}, \mathcal{P}} &= \mathcal{P}(\alpha) \cdot \text{Qpre}_i(\llbracket \phi \rrbracket_{\mathcal{E}, \mathcal{P}}) \\ \llbracket (1 - \alpha) + \alpha \cdot \text{pre}_i(\phi) \rrbracket_{\mathcal{E}, \mathcal{P}} &= (1 - \mathcal{P}(\alpha)) + \mathcal{P}(\alpha) \cdot \text{Qpre}_i(\llbracket \phi \rrbracket_{\mathcal{E}, \mathcal{P}}) \end{aligned}$$

$$\begin{aligned} \llbracket \phi_1 \bigvee \phi_2 \rrbracket_{\mathcal{E}, \mathcal{P}} &= \llbracket \phi_1 \rrbracket_{\mathcal{E}, \mathcal{P}} \bigvee \llbracket \phi_2 \rrbracket_{\mathcal{E}, \mathcal{P}} \\ \llbracket \mu_\nu x. \phi \rrbracket_{\mathcal{E}, \mathcal{P}} &= \left\{ \inf_{\sup} \right\} \{ f \in \mathcal{F} \mid f = \llbracket \phi \rrbracket_{\mathcal{E}[x \mapsto f], \mathcal{P}} \} \end{aligned}$$

The existence of the required fixpoints is guaranteed by the monotonicity and continuity of all operators. The region $\llbracket \phi \rrbracket_{\mathcal{E}, \mathcal{P}}$ is in general not boolean even if the game structure is turn-based deterministic, because the discount factors introduce real numbers. The discounted μ -calculus is closed under negation: if we define the negation of a formula ϕ inductively using $\neg(\alpha \cdot \text{pre}_1(\phi')) = (1 - \alpha) + \alpha \cdot \text{pre}_2(\neg\phi')$ and $\neg((1 - \alpha) + \alpha \cdot \text{pre}_1(\phi')) = \alpha \cdot \text{pre}_2(\neg\phi')$, then $\llbracket \neg\phi \rrbracket_{\mathcal{E}, \mathcal{P}} = \mathbf{1} - \llbracket \phi \rrbracket_{\mathcal{E}, \mathcal{P}}$. This generalizes the duality $\mathbf{1} - \text{Qpre}_1(f) = \text{Qpre}_2(\mathbf{1} - f)$ of the undiscounted pre operators.

A parameter valuation $\mathcal{P}$ is *contractive* if $\mathcal{P}$ maps every parameter to a discount factor strictly less than 1. A fixpoint quantifier μx or νx occurs *syntactically contractive* in a formula ϕ if a pre modality occurs on every syntactic path from the quantifier to a quantified occurrence of the variable x . For example, in the formula $\mu x. (T \vee \alpha \cdot \text{pre}_i(x))$ the fixpoint quantifier occurs syntactically contractive; in the formula $(1 - \alpha) + \alpha \cdot \text{pre}_i(\mu x. (T \vee x))$ it does not. Under a contractive parameter valuation, every syntactically contractive occurrence of a fixpoint quantifier defines a contractive operator on the values of the free variables that are in the scope of the quantifier. Hence, by the Banach fixpoint theorem, the fixpoint is unique. In such cases, since there are unique fixpoints, we need not distinguish between μ and ν quantifiers, and we use a

single (self-dual) fixpoint quantifier λ . Fixpoints can be computed by Picard iteration: $\llbracket \mu x. \phi \rrbracket_{\mathcal{E}, \mathcal{P}} = \lim_{k \rightarrow \infty} f_k$ where $f_0 = \mathbf{0}$, and $f_{k+1} = \llbracket \phi \rrbracket_{\mathcal{E}[x \mapsto f_k], \mathcal{P}}$ for all $k \geq 0$; and $\llbracket \nu x. \phi \rrbracket_{\mathcal{E}, \mathcal{P}} = \lim_{k \rightarrow \infty} f_k$ where $f_0 = \mathbf{1}$, and f_{k+1} is defined as in the μ case. If the fixpoint is unique, then both sequences converge to the same region $\llbracket \lambda x. \phi \rrbracket_{\mathcal{E}, \mathcal{P}}$, one from below, and the other from above.

Approximating the undiscounted semantics. If $\mathcal{P}(\alpha) = 1$, then both discounted pre modalities $\alpha \cdot \text{pre}_i(\cdot)$ and $(1 - \alpha) + \alpha \cdot \text{pre}_i(\cdot)$ collapse, and are interpreted as the quantitative pre operator $\text{Qpre}_i(\cdot)$, for $i \in \{1, 2\}$. In this case, we may omit the parameter α from formulas, writing instead the undiscounted modality $\text{pre}_i(\cdot)$. The *undiscounted semantics* of a formula ϕ is the quantitative region $\llbracket \phi \rrbracket_{\mathcal{E}, 1}$ obtained from the parameter valuation 1 that maps every parameter in Λ to 1 . The undiscounted semantics coincides with the quantitative μ -calculus of [9,23]. In the case of turn-based deterministic game structures, it coincides with the alternating-time μ -calculus of [2], and in the case of transition systems, with the classical μ -calculus of [19]. The following theorem justifies discounting as an approximation theory: the undiscounted semantics of a formula can be obtained as the limit of the discounted semantics as all discount factors tend to 1 .¹

Theorem 1. *Let $\phi(x)$ be a formula of the discounted μ -calculus with a free variable x and parameter α , which always occur in the context $\alpha \cdot \text{pre}_i(x)$, for $i \in \{1, 2\}$. Then*

$$\lim_{\alpha \rightarrow 1} \llbracket \lambda x. \phi(\alpha \cdot \text{pre}_i(x)) \rrbracket_{\mathcal{E}, \mathcal{P}[\alpha \mapsto a]} = \llbracket \mu x. \phi(\text{pre}_i(x)) \rrbracket_{\mathcal{E}, \mathcal{P}}.$$

Furthermore, if x and α always occur in the context $(1 - \alpha) + \alpha \cdot \text{pre}_i(x)$, then

$$\lim_{\alpha \rightarrow 1} \llbracket \lambda x. \phi((1 - \alpha) + \alpha \cdot \text{pre}_i(x)) \rrbracket_{\mathcal{E}, \mathcal{P}[\alpha \mapsto a]} = \llbracket \nu x. \phi(\text{pre}_i(x)) \rrbracket_{\mathcal{E}, \mathcal{P}}.$$

Note that the assumption of the theorem ensures that the fixpoint quantifiers on x occur syntactically contractive on the discounted left-hand side, and therefore define unique fixpoints. Depending on the form of the discounted pre modality, the unique discounted fixpoints approximate either the least or the greatest undiscounted fixpoint. This implies that, in general, limits of discount factors are not interchangeable. Consider the formula

$$\varphi = \lambda y. \lambda x. ((\neg T \wedge \beta \cdot \text{pre}_1(x)) \vee (T \wedge ((1 - \alpha) + \alpha \cdot \text{pre}_1(y)))).$$

Then $\lim_{\alpha \rightarrow 1} \lim_{\beta \rightarrow 1} \varphi$ is equivalent to $\nu y. \mu x. ((\neg T \wedge \text{pre}_1(x)) \vee (T \wedge \text{pre}_1(y)))$, which characterizes, in the turn-based deterministic case, the player 1 winning

¹ It may be noted that Picard iteration itself offers an approximation theory for fixpoint calculi: the longer the iteration sequence, the closer the approximation of the fixpoint. This approximation scheme, however, is neither syntactically robust nor compositional, because it is not closed under the unrolling of fixpoint quantifiers. By contrast, for every discounted μ -calculus formula $\kappa x. \phi(x)$, where $\kappa \in \{\mu, \nu\}$, we have $\llbracket \kappa x. \phi(x) \rrbracket_{\mathcal{E}, \mathcal{P}} = \llbracket \phi(\kappa x. \phi(x)) \rrbracket_{\mathcal{E}, \mathcal{P}}$.

states of a Büchi game (infinitely many T -states must be visited). The inner (β) limit ensures that a T -state will be visited; the outer (α) limit ensures that this remains always the case. On the other hand, $\lim_{\beta \rightarrow 1} \lim_{\alpha \rightarrow 1} \varphi$ is equivalent to $\mu x. \nu y. ((\neg T \wedge \text{pre}_1(x)) \vee (T \wedge \text{pre}_1(y)))$, which characterizes the player 1 winning states of a coBüchi game (eventually only T -states must be visited). This is because the inner (α) limit ensures that only T -states are visited, and the outer (β) limit ensures that this will happen.

4 Properties: Discounted ω -Regular Winning Conditions

In the classical setting, the ω -regular languages can be used to specify system properties (or winning conditions of games), while the μ -calculus provides algorithms for verifying the properties (or computing the winning states). In our discounted approach, the discounted μ -calculus provides the algorithms; what, then, are the properties? We establish a connection between the semantics of a discounted fixpoint expression over a concurrent game structure, and the semantics of the same expression over the paths of the structure. This provides a trace semantics for the discounted μ -calculus, thus giving rise to a notion of “discounted ω -regular properties.”

Reachability and safety conditions. A *discounted reachability game* consists of a concurrent game structure $\mathcal{G}$ (with state space Q) together with a winning condition $\Diamond_a T$, where $T \in \Theta$ is a proposition and $a \in [0, 1]$ is a discount factor. Starting from a state $s \in Q$, player 1 has the objective to reach a T -state as quickly as possible, while player 2 tries to prevent this. The reward for player 1 is a^k if a T -state is visited for the first time after k moves, and 0 if no T -state is ever visited. Formally, we define the payoff function $v_{\Diamond T}^a: \Sigma \mapsto [0, 1]$ on paths by $v_{\Diamond T}^a(s_0, s_1, \dots) = a^k$ for $k = \min\{i \mid s_i \models T\}$, and $v_{\Diamond T}^a(s_0, s_1, \dots) = 0$ if $s_k \not\models T$ for all $k \geq 0$. Then, for every state $s \in Q$, the *value* of the discounted reachability game at s is $(\langle\langle 1 \rangle\rangle \Diamond_a T)(s) = \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} E_s^{\pi_1, \pi_2} \{v_{\Diamond T}^a\}$. This defines a discounted stochastic game [31]. For $a = 1$, the value can be computed as a least fixpoint; for $a < 1$, as the unique fixpoint

$$\langle\langle 1 \rangle\rangle \Diamond_a T = \llbracket \lambda x. (T \vee a \cdot \text{pre}_1(x)) \rrbracket_{\cdot, [\alpha \mapsto a]}.$$

Picard iteration yields $\langle\langle 1 \rangle\rangle \Diamond_a T = \lim_{k \rightarrow \infty} f_k$ where $f_0 = \mathbf{0}$, and $f_{k+1} = (T \vee a \cdot \text{Qpre}_1(f_k))$ for all $k \geq 0$. This gives an approximation scheme from below to solve the discounted reachability game. The sequence converges geometrically in $a < 1$; more precisely, $(\langle\langle 1 \rangle\rangle \Diamond_a T)(s) - f_k(s) \leq a^k$ for all states $s \in Q$ and all $k \geq 0$. This permits the approximation of the value of the game for any desired precision. Furthermore, as the fixpoint is unique, an approximation scheme from above, which starts with $f_0 = \mathbf{1}$, also converges geometrically. For turn-based deterministic game structures, the value of the discounted reachability game $\langle\langle 1 \rangle\rangle \Diamond_a T$ at state s is a^k , where k is the length of the shortest path that player 1 can enforce to reach a T -state, if such a path exists (in the case of

one-player structures, k is the length of the shortest path from s to a T -state). For general game structures and $a = 1$, the value $\langle\langle 1 \rangle\rangle \Diamond_1 T$ at s is the maximal probability with which player 1 can achieve to reach a T -state [9]. A strategy π_1 for player 1 is *optimal* (resp., ϵ -*optimal* for $\epsilon > 0$) for the reachability condition $\Diamond_a T$ if for all states $s \in Q$, we have $\inf_{\pi_2 \in \Pi_2} E_s^{\pi_1, \pi_2} \{v_{\Diamond T}^a\} = (\langle\langle 1 \rangle\rangle \Diamond_a T)(s)$ (resp., $\inf_{\pi_2 \in \Pi_2} E_s^{\pi_1, \pi_2} \{v_{\Diamond T}^a\} \geq (\langle\langle 1 \rangle\rangle \Diamond_a T)(s) - \epsilon$). While undiscounted ($a = 1$) reachability games admit only ϵ -optimal strategies [16], discounted reachability games have optimal memoryless strategies for both players [16,31].

The dual of reachability is safety. A *discounted safety game* consists of a concurrent game structure $\mathcal{G}$ together with a winning condition $\Box_a T$, where $T \in \Theta$ and $a \in [0, 1]$. Starting from a state $s \in Q$, player 1 has the objective to stay within the set of T -states for as long as possible. The payoff function $v_{\Box T}^a: \Sigma \mapsto [0, 1]$ is defined by $v_{\Box T}^a(s_0, s_1, \dots) = 1 - a^k$ for $k = \min \{i \mid s_i \notin T\}$, and $v_{\Box T}^a(s_0, s_1, \dots) = 1$ if $s_k \models T$ for all $k \geq 0$. For every state $s \in Q$, the *value* of the discounted safety game at s is $(\langle\langle 1 \rangle\rangle \Box_a T)(s) = \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} E_s^{\pi_1, \pi_2} \{v_{\Box T}^a\}$. For $a = 1$, the value can be computed as a greatest fixpoint; for $a < 1$, as the unique fixpoint

$$\langle\langle 1 \rangle\rangle \Box_a T = \llbracket \lambda x. (T \wedge ((1 - \alpha) + \alpha \cdot \text{pre}_1(x))) \rrbracket_{\cdot, [\alpha \mapsto a]}.$$

For $a < 1$, the Picard iteration $\langle\langle 1 \rangle\rangle \Box_a T = \lim_{k \rightarrow \infty} f_k$ where $f_0 = \mathbf{0}$, and $f_{k+1} = (T \wedge a \cdot \text{Qpre}_1(f_k))$ for all $k \geq 0$, converges geometrically from below, and with $f_0 = \mathbf{1}$, it converges geometrically from above. For turn-based deterministic game structures and $a < 1$, the value $\langle\langle 1 \rangle\rangle \Box_a T$ at state s is $1 - a^k$, where k is the length of the longest path that player 1 can enforce to stay in T -states. For general game structures and $a = 1$, it is the maximal probability with which player 1 can achieve to stay in T -states forever [9].

In summary, the mathematical appeal of discounting reachability and safety, in addition to the practical appeal of emphasis on the near future, is threefold: (1) geometric convergence from both below and above (no theorems on the rate of convergence are known for $a = 1$); (2) the existence of optimal memoryless strategies (only ϵ -optimal strategies may exist for undiscounted reachability games); (3) the continuous approximation property (Theorem 1), which shows that for $a \rightarrow 1$, the values of discounted reachability and safety games converge to the values of the corresponding undiscounted games.

Trace semantics of fixpoint expressions. Reachability and safety properties are simple, and offer a natural discounted interpretation. For more general ω -regular properties, however, there are often multiple candidates for a discounted interpretation, as there are multiple algorithms for evaluating the property. Consider, for example, Büchi games. An undiscounted Büchi game consists of a concurrent game structure $\mathcal{G}$ together with a winning condition $\Box \Diamond T$, where $T \in \Theta$ specifies a set of Büchi states, which player 1 tries to visit infinitely often. The value of the game at a state s , denoted $(\langle\langle 1 \rangle\rangle \Box \Diamond T)(s)$, is the maximal probability with which player 1 can enforce that a T -state is visited infinitely often. The value of an undiscounted Büchi game can be characterized as [9]

$$\langle\langle 1 \rangle\rangle \Box \Diamond T = \nu y. \mu x. ((\neg T \wedge \text{pre}_1(x)) \vee (T \wedge \text{pre}_1(y))).$$

This fixpoint expression suggests several alternative ways of discounting the Büchi game. For example, one may require that the distances between the infinitely many visits to T -states are as small as possible, obtaining $\nu y. \lambda x. ((\neg T \wedge \alpha \cdot \text{pre}_1(x)) \vee (T \wedge \text{pre}_1(y)))$. Alternatively, one may require that the number of visits to T -states is as large as possible, but arbitrarily spaced, obtaining $\lambda y. \mu x. ((\neg T \wedge \text{pre}_1(x)) \vee (T \wedge ((1 - \beta) + \beta \cdot \text{pre}_1(y))))$. More generally, we can use both discount factors α and β , as in $\lambda y. \lambda x. ((\neg T \wedge \alpha \cdot \text{pre}_1(x)) \vee (T \wedge ((1 - \beta) + \beta \cdot \text{pre}_1(y))))$, and study the effect of various relationships, such as $\alpha < \beta$, $\alpha = \beta$, and $\alpha > \beta$. All these discounted interpretations of Büchi games have two key properties: (1) the value of the game can be computed by algorithms that converge geometrically; and (2) if all discount factors tend to one, then the value of the discounted game tends to the value of the undiscounted game. So instead of defining a discounted Büchi (or more general ω -regular) winning condition, chosen arbitrarily from the alternatives, we take a discounted μ -calculus formula itself as specification of the game and show that, under each interpretation, the formula naturally induces a discounted property of paths.

We first define the semantics of a formula on a path. Consider a concurrent game structure $\mathcal{G}$. Every path $\sigma = s_0, s_1, \dots$ of $\mathcal{G}$ induces an infinite-state² game structure in a natural way: the set of states is $\{(k, s_k) \mid k \geq 0\}$, and at each state (k, s_k) , both players have exactly one move available, whose combination takes the game deterministically to the successor state $(k+1, s_{k+1})$, for all $k \geq 0$. With abuse of notation, we write σ also for the game structure that is induced by the path σ . For this structure and $i \in \{1, 2\}$, $\text{Qpre}_i(\{(k+1, s_{k+1})\})$ is the function that maps (k, s_k) to 1 and all other states to 0. For a closed discounted μ -calculus formula ϕ and parameter valuation $\mathcal{P}$, we define the *trace semantics* of ϕ under $\mathcal{P}$ to be the payoff function $[\phi]_{\mathcal{P}}: \Sigma \mapsto [0, 1]$ that maps every path $\sigma \in \Sigma$ to the value $\llbracket \phi \rrbracket_{\sigma, \mathcal{P}}^{\sigma}(s_0)$, where s_0 is the first state of the path σ (the superscript σ indicates that the formula is evaluated over the game structure induced by σ).

The Cantor metric d_C is defined on the set Σ of paths by $d_C(\sigma_1, \sigma_2) = \frac{1}{2^k}$, where k is the length of the maximal prefix that is common to the two paths σ_1 and σ_2 . The following theorem shows that for discount factors strictly less than 1, the trace semantics of every discounted μ -calculus formula is a continuous function from this metric space to the interval $[0, 1]$. This is in contrast to undiscounted ω -regular properties, which can distinguish between paths that are arbitrarily close.

Theorem 2. *Let ϕ be a closed discounted μ -calculus formula, and let $\mathcal{P}$ be a contractive parameter valuation. For every $\epsilon > 0$, there is a $\delta > 0$ such that for all paths $\sigma_1, \sigma_2 \in \Sigma$ with $d_C(\sigma_1, \sigma_2) < \delta$, we have $|\llbracket \phi \rrbracket_{\sigma_1, \mathcal{P}} - \llbracket \phi \rrbracket_{\sigma_2, \mathcal{P}}| < \epsilon$.*

A formula ϕ of the discounted μ -calculus is *player-1 strongly guarded* [8] if (1) ϕ is closed and consists of a string of fixpoint quantifiers followed by a

² The infiniteness is harmless, because we do not compute in this structure.

quantifier-free part, (2) ϕ contains no occurrences of pre_2 , and (3) every conjunction in ϕ has at least one constant argument; that is, every conjunctive subformula of ϕ has the form $T \wedge \phi'$, where T is a boolean combination of propositions. In the classical μ -calculus, all ω -regular winning conditions of turn-based deterministic games can be expressed by strongly guarded (e.g., Rabin chain) formulas [13]. For player-1 strongly guarded formulas ϕ the following theorem gives the correspondence between the semantics of ϕ on structures and the semantics of ϕ on paths: the value of ϕ at a state s under parameter valuation $\mathcal{P}$ is the value of the game with start state s and payoff function $[\phi]_{\mathcal{P}}$.

Theorem 3. *Let $\mathcal{G}$ be a concurrent game structure, let ϕ be a player-1 strongly guarded formula of the discounted μ -calculus, and let $\mathcal{P}$ be a parameter valuation. For every state s of $\mathcal{G}$, we have $\llbracket \phi \rrbracket_{\mathcal{P}}^{\mathcal{G}}(s) = \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} E_s^{\pi_1, \pi_2} \{[\phi]_{\mathcal{P}}\}$.*

Rabin chain conditions. An undiscounted Rabin chain game [13,25] consists of a concurrent game structure $\mathcal{G}$ together with a winning condition $\bigvee_{i=0}^{n-1} (\Box \Diamond T_{2i} \wedge \neg \Box \Diamond T_{2i+1})$, where $n > 0$ and the T_j 's are propositions with $\emptyset \subseteq \llbracket T_{2n} \rrbracket \subseteq \llbracket T_{2n-1} \rrbracket \subseteq \dots \subseteq \llbracket T_0 \rrbracket = Q$. A more intuitive characterization of this winning condition can be obtained by defining, for all $0 \leq j \leq 2n-1$, the set $C_j \subseteq Q$ of states of *color* j by $C_j = \llbracket T_j \rrbracket \setminus \llbracket T_{j+1} \rrbracket$. For a path $\sigma \in \Sigma$, let $\text{MaxCol}(\sigma)$ be the maximal j such that a state in C_j occurs infinitely often in σ . The winning condition for player 1 is that $\text{MaxCol}(\sigma)$ is even. The ability to solve games with Rabin chain conditions suffices for solving games with arbitrary ω -regular winning conditions, because every ω -regular property can be translated into a deterministic Rabin chain automaton [25,32].

As in the Büchi case, there are many ways to discount a Rabin chain game, so we use the corresponding fixpoint expression to explore various tradeoffs. Accordingly, for discount factors $a_0, \dots, a_{2n-1} < 1$, we define the value of an $(a_0, \dots, a_{2n-1})$ -discounted Rabin chain game by

$$R(a_0, \dots, a_{2n-1}) = \llbracket \lambda x_{2n-1} \dots \lambda x_0. \bigvee_{0 \leq j < 2n} (C_j \wedge \text{Rpre}(x_j)) \rrbracket_{\{\alpha_j \mapsto a_j \mid 0 \leq j < 2n\}},$$

where $\text{Rpre}(x_j) = \alpha_j \cdot \text{pre}_1(x_j)$ if j is odd, and $\text{Rpre}(x_j) = (1 - \alpha_j) + \alpha_j \cdot \text{pre}_1(x_j)$ if j is even. Note that the fixpoint expression is player-1 strongly guarded. The value $R(a_0, \dots, a_{2n-1})$ of the discounted Rabin chain game can be approximated monotonically by Picard iteration from below and above. Moreover, if the j -th fixpoint is computed for k_j steps, we can bound the cumulative error of the process. Let ε_j be the error in the value of the j -th fixpoint; then $\varepsilon_0 \leq a_0^{k_0}$, and $\varepsilon_j \leq \frac{\varepsilon_{j-1}}{1-a_j} + a_j^{k_j}$ for all $1 \leq j \leq 2n-1$.

Theorem 4. *For a vector $(k_0, \dots, k_{2n-1})$ of integers, let $R_{k_0, \dots, k_{2n-1}}^+$ be the region obtained by approximating from below the j -th fixpoint of the discounted μ -calculus formula for $R(a_0, \dots, a_{2n-1})$ for k_j iterations, and let $R_{k_0, \dots, k_{2n-1}}^-$ be the region obtained by approximating from above. If*

$a_0, \dots, a_{2n-1} < 1$, then for each state s , we have $R(a_0, \dots, a_{2n-1})(s) - \varepsilon_{2n-1} \leq R_{k_0, \dots, k_{2n-1}}^\top(s) \leq R_{k_0, \dots, k_{2n-1}}(s) \leq R(a_0, \dots, a_{2n-1})(s) + \varepsilon_{2n-1}$. Moreover, if R is the value of the corresponding undiscounted Rabin chain game, then $\lim_{a_{2n-1} \rightarrow 1} \dots \lim_{a_0 \rightarrow 1} R(a_0, \dots, a_{2n-1}) = R$.

5 State Equivalences: Discounted Bisimilarity

Consider a concurrent game structure $\mathcal{G} = \langle Q, M, \Gamma_1, \Gamma_2, \delta \rangle$. A *distance function* $d: Q^2 \mapsto [0, 1]$ is a pseudo-metric on the states with the range $[0, 1]$. Distance functions provide a quantitative generalization for equivalence relations on states: distance 0 means “equivalent” in the boolean sense, and distance 1 means “different” in the boolean sense. For two distance functions d_1 and d_2 , we write $d_1 \leq d_2$ if $d_1(s, t) \leq d_2(s, t)$ for all states $s, t \in Q$. Given a discount factor $a \in [0, 1]$, we define the functor F_a mapping distance functions to distance functions: for every distance function d and all states $s, t \in Q$, we define $F_a(d)(s, t) = 1$ if there is a proposition $T \in \Theta$ such that $\llbracket T \rrbracket(s) \neq \llbracket T \rrbracket(t)$, and

$$F_a(d)(s, t) = a \cdot \max \left\{ \begin{array}{l} \sup_{\xi_1 \in \mathcal{D}_1(s)} \inf_{\hat{\xi}_1 \in \mathcal{D}_1(t)} \sup_{\xi_2 \in \mathcal{D}_2(t)} \inf_{\hat{\xi}_2 \in \mathcal{D}_2(s)} E_{s,t}^{s':\xi_1 \xi_2, t':\hat{\xi}_1, \hat{\xi}_2} \{d_a(s', t')\}, \\ \sup_{\hat{\xi}_1 \in \mathcal{D}_1(t)} \inf_{\xi_1 \in \mathcal{D}_1(s)} \sup_{\xi_2 \in \mathcal{D}_2(s)} \inf_{\hat{\xi}_2 \in \mathcal{D}_2(t)} E_{s,t}^{s':\xi_1 \xi_2, t':\hat{\xi}_1, \hat{\xi}_2} \{d_a(s', t')\} \end{array} \right\}$$

otherwise. In the above formula, $\mathcal{D}_i(u) = \mathcal{D}(\Gamma_i(u))$ is the set of probability distributions over the moves of player $i \in \{1, 2\}$ at the state $u \in Q$. By $E_{s,t}^{s':\xi_1 \xi_2, t':\hat{\xi}_1, \hat{\xi}_2} \{d(s', t')\}$ we denote the expected value $d(s', t')$ of the distance function d when the state s' results from playing the distributions of moves ξ_1 and ξ_2 from s , and t' results from playing $\hat{\xi}_1$ and $\hat{\xi}_2$ from t . Formally,

$$E_{s,t}^{s':\xi_1 \xi_2, t':\hat{\xi}_1, \hat{\xi}_2} \{d(s', t')\} = \sum_{s', t' \in Q} \sum_{\gamma_1 \in \Gamma_1(s)} \sum_{\gamma_2 \in \Gamma_2(s)} \sum_{\hat{\gamma}_1 \in \Gamma_1(t)} \sum_{\hat{\gamma}_2 \in \Gamma_2(t)} d(s', t') \cdot \delta(s' \mid s, \gamma_1, \gamma_2) \cdot \delta(t' \mid t, \hat{\gamma}_1, \hat{\gamma}_2) \cdot \xi_1(\gamma_1) \cdot \xi_2(\gamma_2) \cdot \hat{\xi}_1(\hat{\gamma}_1) \cdot \hat{\xi}_2(\hat{\gamma}_2).$$

The fixpoints of the functor F_a are called *a-discounted (game) bisimulations*. The least fixpoint of F_a is called *a-discounted (game) bisimilarity*, and denoted $B_a^\mathcal{G}$ (the superscript is omitted if the game structure $\mathcal{G}$ is clear from the context).³ If $a < 1$, then F_a has a unique fixpoint; in this case, there is a unique *a-discounted bisimulation*, namely, B_a . If $a = 1$, instead of 1-discounted, we say *undiscounted*.

On MDPs (one-player game structures), for $a < 1$, discounted game bisimulation coincides with the discounted bisimulation of [12], and undiscounted game bisimulation coincides with the probabilistic bisimulation of [30]. On transition systems (one-player deterministic game structures), undiscounted game bisimulation coincides with classical bisimulation [24]. However, undiscounted game bisimulation is not equivalent to the alternating bisimulation of [3], which has been defined for deterministic game structures. By the minimax theorem [35],

³ Bisimilarity is usually considered a greatest fixpoint, but in our setup, the distance function that considers all states to be equivalent in the boolean sense is the least distance function.

we can exchange the two middle sup and inf operators in the definition of F_a ; that is, the roles of players 1 and 2 can be exchanged. Hence, there is only one version of (un)discounted game bisimulation, while there are distinct player 1 and player 2 alternating bisimulations. Alternating bisimulation corresponds to the case where the sets $\mathcal{D}_i(u)$, for $i \in \{1, 2\}$ and $u \in Q$, consist only of deterministic distributions, where each player must choose a specific move (indeed, the minimax theorem does not hold if the players are forced to use deterministic distributions). In the case of turn-based deterministic game structures, the two definitions collapse, but for concurrent game structures the sup-inf interpretation of winning is strictly weaker than the deterministic interpretation [7].

The a -discounted bisimilarity B_a can be computed using Picard iteration: starting from $d_a^{(0)}$, with $d_a^{(0)}(s, t) = 0$ for all states $s, t \in Q$, let $d_a^{(k+1)} = F_a(d_a^{(k)})$ for all $k \geq 0$. If $a < 1$, then we may start from any distance function $d_a^{(0)}$ (because the fixpoint is unique) and the convergence is geometric with rate a . The theorem below relates discounted and undiscounted game bisimulation.

Theorem 5. *On every concurrent game structure, $\lim_{a \rightarrow 1} B_a = B_1$. Moreover, for two states s and t , we have $B_1(s, t) = 0$ iff $B_a(s, t) = 0$ for any and all discount factors $a > 0$, and $B_1(s, t) = 1$ iff $B_a(s, t) = 1$ for any and all $a > 0$.*

Our main theorem on discounted game bisimulation states that for all states, closeness in discounted game bisimilarity corresponds to closeness in the value of discounted μ -calculus formulas. In other words, a small perturbation of a system can only cause a small change of its properties.

Theorem 6. *Consider two states s and t of a concurrent game structure, and a discount factor $a < 1$. For all closed discounted μ -calculus formulas ϕ and a -bounded parameter valuations $\mathcal{P}$, we have $|\llbracket \phi \rrbracket_{\cdot, \mathcal{P}}(s) - \llbracket \phi \rrbracket_{\cdot, \mathcal{P}}(t)| \leq B_a(s, t)$. Also, $\sup_{\phi} |\llbracket \phi \rrbracket_{\cdot, \mathcal{P}}(s) - \llbracket \phi \rrbracket_{\cdot, \mathcal{P}}(t)| = B_a(s, t)$.*

Let ϵ be a nonnegative real. A game structure $\mathcal{G}' = \langle Q, M, \Gamma_1, \Gamma_2, \delta' \rangle$ is an ϵ -perturbation of the game structure $\mathcal{G} = \langle Q, M, \Gamma_1, \Gamma_2, \delta \rangle$ if for all states $s \in Q$, all sets $X \subseteq Q$, and all moves $\gamma_1 \in \Gamma_1(s)$ and $\gamma_2 \in \Gamma_2(s)$, we have $|\sum_{t \in X} \delta(t \mid s, \gamma_1, \gamma_2) - \sum_{t \in X} \delta'(t \mid s, \gamma_1, \gamma_2)| \leq \epsilon$. We write $B_a^{\mathcal{G} \uplus \mathcal{G}'}$ for the a -discounted bisimilarity on the disjoint union of the game structures $\mathcal{G}$ and $\mathcal{G}'$. The following theorem, which generalizes a result of [11] from one-player structures to games, shows that discounted bisimilarity is robust under perturbations.

Theorem 7. *Let $\mathcal{G}'$ be an ϵ -perturbation of a concurrent game structure $\mathcal{G}$, and let $a < 1$ be a discount factor. For every state s of $\mathcal{G}$ and corresponding state s' of $\mathcal{G}'$, we have $B_a^{\mathcal{G} \uplus \mathcal{G}'}(s, s') \leq K \cdot \epsilon$, where $K = \sup_{k \geq 0} \{k \cdot a^k\}$.*

References

1. R. Alur and T.A. Henzinger. Finitary fairness. *ACM TOPLAS*, 20:1171–1194, 1994.

2. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49:672–713, 2002.
3. R. Alur, T.A. Henzinger, O. Kupferman, and M.Y. Vardi. Alternating refinement relations. In *Concurrency Theory*, LNCS 1466, pp. 163–178. Springer, 1998.
4. M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.
5. J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Congr. Logic, Methodology, and Philosophy of Science 1960*, pp. 1–12. Stanford University Press, 1962.
6. L. de Alfaro. Stochastic transition systems. In *Concurrency Theory*, LNCS 1466, pp. 423–438. Springer, 1998.
7. L. de Alfaro, T.A. Henzinger, and O. Kupferman. Concurrent reachability games. In *Symp. Foundations of Computer Science*, pp. 564–575. IEEE, 1998.
8. L. de Alfaro, T.A. Henzinger, and R. Majumdar. From verification to control: Dynamic programs for ω -regular objectives. In *Symp. Logic in Computer Science*, pp. 279–290. IEEE, 2001.
9. L. de Alfaro and R. Majumdar. Quantitative solution of ω -regular games. In *Symp. Theory of Computing*, pp. 675–683. ACM, 2001.
10. C. Derman. *Finite-State Markovian Decision Processes*. Academic Press, 1970.
11. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labeled Markov systems. In *Concurrency Theory*, LNCS 1664, pp. 258–273. Springer, 1999.
12. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Symp. Logic in Computer Science*, pp. 413–422. IEEE, 2002.
13. E.A. Emerson and C.S. Jutla. Tree automata, μ -calculus and determinacy. In *Symp. Foundations of Computer Science*, pp. 368–377. IEEE, 1991.
14. E.A. Emerson, C.S. Jutla, and A.P. Sistla. On model checking for fragments of μ -calculus. In *Computer-aided Verification*, LNCS 697, pp. 385–396. Springer, 1993.
15. E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ -calculus. In *Symp. Logic in Computer Science*, pp. 267–278. IEEE, June 1986.
16. J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.
17. Y. Gurevich and L. Harrington. Trees, automata, and games. In *Symp. Theory of Computing*, pp. 60–65. ACM, 1982.
18. C.-C. Jou and S.A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In *Concurrency Theory*, LNCS 458, pp. 367–383. Springer, 1990.
19. D. Kozen. A probabilistic PDL. In *Symp. Theory of Computing*, pp. 291–297. ACM, 1983.
20. D. Kozen. Results on the propositional μ -calculus. *Theoretical Computer Science*, 27:333–354, 1983.
21. Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Symp. Principles of Distributed Computing*, pp. 377–408. ACM, 1990.
22. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1991.
23. A. McIver. Reasoning about efficiency within a probabilistic μ -calculus. *Electronic Notes in Theoretical Computer Science*, 22, 1999.
24. R. Milner. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, vol. B, pp. 1202–1242. Elsevier, 1990.

25. A.W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, LNCS 208, pp. 157–168. Springer, 1984.
26. G. Owen. *Game Theory*. Academic Press, 1995.
27. A. Pnueli. The temporal logic of programs. In *Symp. Foundations of Computer Science*, pp. 46–57. IEEE, 1977.
28. M.O. Rabin. *Automata on Infinite Objects and Church's Problem*. Conference Series in Mathematics, vol. 13. AMS, 1969.
29. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995. Tech. Rep. MIT/LCS/TR-676.
30. R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. In *Concurrency Theory*, LNCS 836, pp. 481–496. Springer, 1994.
31. L.S. Shapley. Stochastic games. *Proc. National Academy of Sciences*, 39:1095–1100, 1953.
32. W. Thomas. On the synthesis of strategies in infinite games. In *Theoretical Aspects of Computer Science*, LNCS 900, pp. 1–13. Springer, 1995.
33. M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state systems. In *Symp. Foundations of Computer Science*, pp. 327–338. IEEE, 1985.
34. M.Y. Vardi. A temporal fixpoint calculus. In *Symp. Principles of Programming Languages*, pp. 250–259. ACM, 1988.
35. J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1947.

Information Flow in Concurrent Games^{*}

Luca de Alfaro¹ and Marco Faella^{1,2}

¹ Department of Computer Engineering, UC Santa Cruz, USA

² Dipartimento di Informatica ed Applicazioni, Università degli Studi di Salerno,
Italy

Abstract. We consider games where the players have perfect information about the game's state and history, and we focus on the information exchange that takes place at each round as the players choose their moves. The ability of a player to gather information on the opponent's choice of move in a round determines her ability to counteract the move, and win the game. When the game is played between teams, rather than single players, the amount of intra-team communication determines the ability of the team members to coordinate their moves and win the game. We consider games with quantitative bounds on inter-team and intra-team information flow, and we provide algorithms and complexity bounds for their solution.

1 Introduction

We consider repeated games played for an infinite number of rounds on a finite state space [Sha53]. At each round of the game, each player selects a move; the selected moves jointly determine the next state of the game [Sha53,FV97,AHK97]. This process, repeated, gives rise to a play of the game, consisting in the infinite sequence of visited states. We consider safety games, where the goal consists in staying forever in a safe set of states, and reachability games, where the goal consists in reaching a desired subset of states [Tho95,Zie98]. The ability of a player to win such games depends on the information available to the player. In *partial information* games, players have incomplete information about the current state of the game and the past history; computing the sets of winning states for safety and reachability goals is EXPTIME complete [Rei84,KV97]. In this paper, we consider instead games where the players have perfect information about the game's current state and history, and we focus instead on the information exchange that takes place at each round, between players and within players, as the players choose their moves.

We first consider the distinction between *turn-based* and *concurrent* games. Usually, this distinction is defined structurally: a game is *concurrent* if at each state both players may have a choice of moves [FV97,AHK97], and is *turn-based* if at each state at most one player has the choice among multiple moves [BL69,

^{*} This research was supported in part by the NSF CAREER award CCR-0132780, the NSF grant CCR-0234690, the ONR grant N00014-02-1-0671, and the MIUR grant "Metodi Formali per la Sicurezza e il Tempo" (MEFISTO).

GH82,EJ91,Tho95]. We argue that the difference is best captured in terms of information: a game is concurrent if the two players must choose their moves independently, on the basis of the same information, and it is turn-based if one of the two players has full information about the opponent's choice of move when choosing her own move. Indeed, in a game where the players play simultaneously, if one player has full information about the other player's choice of move, the game is in effect turn-based, the player with full information playing second. While this may seem an odd way to play a game, it occurs in hardware design whenever a Moore machine, whose next outputs (the next move) can depend on the current inputs only, is composed with a Mealy machine, whose next outputs (move) can depend both on the current inputs, and on the *next* inputs from other machines. Effectively, the Moore machine chooses first, while the Mealy machine can look at the move chosen by the Moore machine before choosing its own move. Conversely, whenever in a turn-based game one of the players is prevented from observing the preceding opponent move (along with its effects), the game is effectively concurrent, even though the choice of moves is not simultaneous. Indeed, there would hardly be any concurrent game, if the distinction between concurrent and turn-based were based on truly simultaneous choice, rather than on independent choice under the same information.

Once the distinction between concurrent and turn-based games is phrased in terms of information, concurrent and turn-based games constitute the two extremes in a spectrum of games, which we call *semi-concurrent*, where one player is able to gather a bounded amount of information about the opponent's move before choosing her own. Games where players exchange information in a round have been considered in [dAHM00,dAHM01] to model the interaction of synchronous hardware; in those works the communication scheme is fixed, and is specified together with the game. We consider here games where the amount of information exchanged between players is bounded, but the information content, and the way it is gathered, is left to the discretion of the players. Semi-concurrent games have several applications. In the design of controllers for digital circuits, semi-concurrent games model the case where together with the controller, we can design combinatorial signals that provide information about the next state of the controlled system. Moreover, semi-concurrent games can be used to model games played with untrustworthy adversaries, who can exploit leaked information about our choice of move.

We provide algorithms for solving semi-concurrent games with respect to safety and reachability conditions. We consider both the case when the goal must be attained for all plays (*sure* winning), and the case when the goal must be attained with probability 1 (*almost* winning) [dAHK98], and we consider both the case when the player striving to achieve the goal can spy on the opponent, or is spied upon. We give tight bounds for the complexity of these algorithms, proving that for several combinations of goals, spying, and winning mode (sure or almost), deciding whether a player can win from a state is *NP*-complete. We also show that the larger the amount of information a player can gather about the opponent's choice of move, the more games the player can win; our results

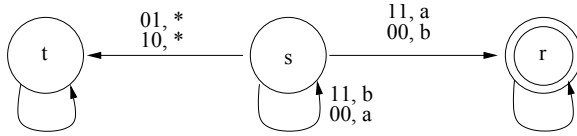


Fig. 1. A concurrent game. An edge label such as $11, a$ indicates that the edge is followed when player 1 chooses '11' and player 2 chooses 'a'. The game starts in s , and the goal is to reach r . States t and r are sink states, without outgoing transitions.

enable the determination of the minimum amount of information about the opponent's move that is required in order to win. Finally, investigate the need for randomization in winning reachability games. From [dAHK98] it is known that randomization is needed to win reachability games with probability 1 if the game is concurrent, but not if it is turn-based. We show that randomization in general is always needed to win semi-concurrent reachability games with probability 1, regardless of whether a player is spying or is being spied upon, as long as one of the players does not have perfect information about the other player's choice of move.

Concurrent games can also be seen as the extreme point of another spectrum, concerning the amount of communication *within* a player. While some players are single entities, others are internally composed of separate entities: such composite players are called *teams*, and the entities they comprise are called *team members*. We consider games where the move chosen by a team is a tuple, each team member choosing a component of the tuple. This problem was first studied in [PR79], where it was shown that team games where the players have incomplete information about the state of the game are in general undecidable. Later, [PR90] and [KV01] considered team games with linear or cyclic communication structure between team members, and showed that solving such games with respect to linear or branching time temporal logic conditions is decidable. These previous works considered games where each team member has a different, partial view of the state of the game. Here, we consider instead the situation in which the team members share complete information about the state of the game, but must coordinate their moves at each round.

A team can readily play any deterministic choice of move that can be played by a single-entity player: each team member simply chooses deterministically the desired move component. However, if the team members cannot communicate while choosing the move components, the team can only play *randomized* distributions of moves that result from the independent randomization of each member's choice. This limits the team's ability to win reachability games, as illustrated by the game of Figure 1. Player 1 can reach r from state s with probability 1 by choosing moves 00 and 11 with probability $1/2$ each, and by choosing moves 01 and 10 with probability 0. However, assume player 1 consists in a two-member team, where each member chooses one of the bits. If the two team members cannot communicate while choosing the bits, the team can only play

probability distributions $p(i, j)$ for $i, j \in \{0, 1\}$ of the form $p(i, j) = q_1(i)q_2(j)$, where q_1 and q_2 are the distributions chosen by the team members. It is easy to see that team 1 cannot reach r with probability 1 from s using these distributions. If an arbitrary amount of communication can take place in a round between team members, the team can replicate the behavior of a single-entity player: thus, concurrent games constitute the limit case of team games for arbitrary communication.

Here, we study team games where a bounded (possibly 0) amount of communication can take place among team members in a round. Team games model controller-design problems where the controller consists of distributed sub-controllers that can observe the current state of the controlled system, but that have limited communication ability to coordinate their next move. For instance, in synchronous digital circuits, it may not be feasible for the sub-controllers to communicate their next state if they communicate through links that are slow compared to the system clock. Moreover, team games model the real-world situation when members of the same team must coordinate their next move covertly using limited bandwidth.

Team safety games can be solved in the same manner as concurrent safety games, since no randomization is required in the winning strategies. We present algorithms for solving team reachability games with communicating and non-communicating team members, and we provide tight bounds for their complexity, showing in particular that solving non-communicating reachability games is an *NP*-complete problem. While in the case of semi-concurrent games, the more the information is communicated, the more the games that can be won, we show that for team games, a single bit of information communicated between team members is as efficient as complete coordination. On the other hand, we show that probability 1 reachability team games with are in general not determined: if one team cannot win the game with probability 1, this does not imply the existence of a single adversary strategy that prevents the team from winning.

2 Games

For a finite set A , a *probability distribution* on A is a function $p : A \mapsto [0, 1]$ such that $\sum_{a \in A} p(a) = 1$; we denote the set of probability distributions on A by $\mathcal{D}(A)$. A *game structure* is a tuple $G = (S, M, \Gamma_1, \Gamma_2, \tau)$, where:

- S is a finite set of *states*;
- M is a finite sets of *moves*;
- $\Gamma_1, \Gamma_2 : S \mapsto 2^M \setminus \{\emptyset\}$ are the *move assignments* of the players;
- $\tau : S \times M \times M \mapsto \mathcal{D}(S)$ is the *probabilistic transition function*.

At every state $s \in S$, player 1 chooses a move $a \in \Gamma_1(s)$, and player 2 chooses a move $b \in \Gamma_2(s)$; the game then proceeds to state $t \in S$ with probability $\tau(s, a, b)(t)$. For $s \in S$ and $a, b \in M$, we denote by $\delta(s, a, b) = \{t \in S \mid \tau(s, a, b)(t) > 0\}$ the set of possible successors of s when moves a and b are played. A *play* of G is an infinite sequence $s_0, \langle a_0, b_0 \rangle, s_1, \langle a_1, b_1 \rangle, \dots$ such that

for all $n \geq 0$, we have $a_n \in \Gamma_1(s_n)$, $b_n \in \Gamma_2(s_n)$, and $s_{n+1} \in \delta(s_n, a_n, b_n)$. We denote by $Plays_{s_0}$ the set of plays starting from $s_0 \in S$, and by $Plays = \bigcup_{s_0 \in S} Plays_{s_0}$ the set of all plays of G . A *history* σ is a finite play prefix $\sigma = s_0, \langle a_0, b_0 \rangle, s_1, \langle a_1, b_1 \rangle, \dots, s_n$ that terminates in a state; we denote by $last(\sigma)$ the last state s_n of σ . We denote by $Hist_{s_0}$ the set of histories of G starting from s_0 , and by $Hist = \bigcup_{s_0 \in S} Hist_{s_0}$ the set of all histories of G . We define the *size* of G to be equal to the number of entries of the transition function δ ; specifically, $|G| = \sum_{s \in S} \sum_{a \in \Gamma_1(s)} \sum_{b \in \Gamma_2(s)} |\delta(s, a, b)|$. Given $s \in S$, $Y \subseteq S$ and $B \subseteq M$, it is useful to define $Safe_1(s, Y, B)$ as the set of moves of player 1 that ensure staying in Y when the player 2 chooses moves from B . Formally, $Safe_1(s, Y, B) = \{a \in \Gamma_1(s) \mid \forall b \in B. \delta(s, a, b) \subseteq Y\}$. We define symmetrically $Safe_2(s, Y, A)$.

2.1 Strategies, Winning Conditions, and Games

Let Ω_s be the set of measurable subsets of $Plays_s$, defined as usual (see e.g. [Wil91]). A *family of strategies* $\langle \Upsilon_1, \Upsilon_2, \text{Pr} \rangle$ consists of two sets Υ_1 and Υ_2 of strategies for players 1 and 2, together with a mapping Pr that associates a probability measure $\text{Pr}_s^{\pi_1, \pi_2} : \Omega_s \mapsto [0, 1]$ with each initial state s and pair of strategies $\pi_1 \in \Upsilon_1$ and $\pi_2 \in \Upsilon_2$. Thus, $\text{Pr}_s^{\pi_1, \pi_2}(\mathcal{E})$ is the probability that a game starting from $s \in S$ follows a play in $\mathcal{E} \in \Omega_s$ when players 1 and 2 play according to strategies π_1 and π_2 , respectively. A probability measure $\text{Pr}_s^{\pi_1, \pi_2}$ over Ω_s gives rise to a set $Outcomes(s, \text{Pr}, \pi_1, \pi_2) \subseteq Plays_s$, of *outcome plays*, consisting of the plays whose finite prefixes can be followed with non-zero probability. Formally, for $\rho \in Plays_s$ and $n > 0$, let $\mathcal{E}(\rho, n) \in \Omega_s$ be the set of plays that agree with ρ up to round n : then, $\rho \in Outcomes(s, \text{Pr}, \pi_1, \pi_2)$ if $\text{Pr}_s^{\pi_1, \pi_2}(\mathcal{E}(\rho, n)) > 0$ for all $n > 0$.

We consider *safety games*, in which the winning condition $\Box R$ consists in remaining forever in a subset $R \subseteq S$ of states, and *reachability games*, in which the winning condition $\Diamond R$ consists in reaching a subset $R \subseteq S$ of states; we define $\llbracket \Box R \rrbracket = \{s_0, \langle a_0, b_0 \rangle, s_1, \langle a_1, b_1 \rangle, \dots \in Plays \mid \forall n \in \mathbb{N}. s_n \in R\}$ and $\llbracket \Diamond R \rrbracket = \{s_0, \langle a_0, b_0 \rangle, s_1, \langle a_1, b_1 \rangle, \dots \in Plays \mid \exists n \in \mathbb{N}. s_n \in R\}$. A *game* is thus a tuple $(G, \phi, i, \mathcal{M} \mid \Upsilon_1, \Upsilon_2, \text{Pr})$, composed of a game structure G , a winning condition $\phi \in \{\Box R, \Diamond R\}$, an integer $i \in \{1, 2\}$, a modality $\mathcal{M} \in \{sure, almost\}$, and a family of strategies. Given a family of strategies $\langle \Upsilon_1, \Upsilon_2, \text{Pr} \rangle$ and $\phi \in \{\Box R, \Diamond R\}$, we define the sets $win_1(G, \phi, sure \mid \Upsilon_1, \Upsilon_2, \text{Pr})$ of player-1 *sure-winning* states and the set $win_1(G, \phi, almost \mid \Upsilon_1, \Upsilon_2, \text{Pr})$ of player-1 *almost-winning* states as follows [dAHK98]:

Sure-winning. For all $s \in S$, we have $s \in win_1(G, \phi, sure \mid \Upsilon_1, \Upsilon_2, \text{Pr})$ if there is $\pi_1 \in \Upsilon_1$ such that for all $\pi_2 \in \Upsilon_2$ we have $Outcomes(s, \text{Pr}, \pi_1, \pi_2) \subseteq \llbracket \phi \rrbracket$.

Almost-winning. For all $s \in S$, we have $s \in win_1(G, \phi, almost \mid \Upsilon_1, \Upsilon_2, \text{Pr})$ if there is $\pi_1 \in \Upsilon_1$ such that for all $\pi_2 \in \Upsilon_2$ we have $\text{Pr}_s^{\pi_1, \pi_2}(\llbracket \phi \rrbracket) = 1$.

The sets of player-2 sure and almost-sure winning states are defined symmetrically. A *winning strategy* is a strategy that ensures victory to a player

with the prescribed mode (sure, or almost), for all winning states. Precisely, for $\mathcal{M} \in \{\text{sure}, \text{almost}\}$, a winning strategy for $(G, \phi, 1, \mathcal{M} \mid \mathcal{Y}_1, \mathcal{Y}_2, \text{Pr})$ is a strategy $\pi_1 \in \mathcal{Y}_1$ such that, for all $s \in \text{win}_1(G, \phi, \text{almost} \mid \mathcal{Y}_1, \mathcal{Y}_2, \text{Pr})$ and all $\pi_2 \in \mathcal{Y}_2$, we have that $\text{Outcomes}(s, \text{Pr}, \pi_1, \pi_2) \subseteq \phi$ if $\mathcal{M} = \text{sure}$, and $\text{Pr}_s^{\pi_1, \pi_2}(\llbracket \phi \rrbracket) = 1$ if $\mathcal{M} = \text{almost}$. A *spoiling strategy* is an adversary strategy that prevents a player from winning whenever victory cannot be assured. Precisely, for $\mathcal{M} \in \{\text{sure}, \text{almost}\}$, a spoiling strategy for $(G, \phi, 1, \mathcal{M} \mid \mathcal{Y}_1, \mathcal{Y}_2, \text{Pr})$ is a strategy $\pi_2 \in \mathcal{Y}_2$ such that, for all $s \notin \text{win}_1(G, \phi, \text{almost} \mid \mathcal{Y}_1, \mathcal{Y}_2, \text{Pr})$ and all $\pi_1 \in \mathcal{Y}_1$, we have that $\text{Outcomes}(s, \text{Pr}, \pi_1, \pi_2) \not\subseteq \phi$ if $\mathcal{M} = \text{sure}$, and $\text{Pr}_s^{\pi_1, \pi_2}(\llbracket \phi \rrbracket) < 1$ if $\mathcal{M} = \text{almost}$. Analogous definitions hold for the winning problems that refer to player 2.

A *game type* is a tuple $(\Delta, i, \mathcal{M} \mid \mathcal{Y}_1, \mathcal{Y}_2, \text{Pr})$ where $i \in \{1, 2\}$, $\mathcal{M} \in \{\text{sure}, \text{almost}\}$, and $\Delta \in \{\square, \diamond\}$. We say that a game type is *determined* iff, for all game structures G , player $i \in \{1, 2\}$, and $R \subseteq S$, both winning and spoiling strategies exist for $(G, \Delta R, i, \mathcal{M} \mid \mathcal{Y}_1, \mathcal{Y}_2, \text{Pr})$.

2.2 Concurrent Games

In concurrent games, the players choose their moves simultaneously and independently. A *concurrent strategy* for player $i \in \{1, 2\}$ is a mapping $\pi_i: \text{Hist} \mapsto \mathcal{D}(M)$ that associates with every history σ of the game a probability distribution $\pi_i(\sigma)$ used to select the next move; for all $a \in M$, we require that $\pi_i(\sigma)(a) > 0$, implies $a \in \Gamma_i(\text{last}(\sigma))$, ensuring that the strategy selects only moves that are available to the players. We denote by Π_i^c the set of all concurrent strategies for player $i \in \{1, 2\}$. Given a strategy $\pi \in \Pi_1^c \cup \Pi_2^c$, we say that π is *memoryless* if for all $\sigma \in \text{Hist}$ we have $\pi(\sigma) = \pi(\text{last}(\sigma))$, and we say that π is *deterministic* if for all $\sigma \in \text{Hist}$ and all $a \in M$ we have $\pi(\sigma)(a) \in \{0, 1\}$. An initial state s_0 and a pair of strategies $\pi_1 \in \Pi_1^c$ and $\pi_2 \in \Pi_2^c$ give rise to a probability $\text{Prb}_{s_0}^{\pi_1, \pi_2}$ on histories defined inductively by $\text{PrbC}_{s_0}^{\pi_1, \pi_2}(s_0) = 1$ and, for $n \geq 0$, by $\text{PrbC}_{s_0}^{\pi_1, \pi_2}(s_0, \dots, s_n, \langle a_n, b_n \rangle, s_{n+1}) = \text{PrbC}_{s_0}^{\pi_1, \pi_2}(s_0, \dots, s_n) \cdot \pi_1(s_0, \dots, s_n)(a_n) \cdot \pi_2(s_0, \dots, s_n)(b_n) \cdot \tau(s_n, a_n, b_n)(s_{n+1})$. These probabilities on histories give rise to a probability measure $\text{PrbC}_{s_0}^{\pi_1, \pi_2}$ on Ω_{s_0} [Wil91]. A *concurrent game* is a game in which the players use concurrent strategies. The winning states of concurrent safety and reachability games can be computed using μ -calculus; we briefly review the approach, as it will be the starting point of the algorithms we will present for other families of strategies.

Safety. The solution of concurrent safety games is entirely classical. The set of winning states can be computed using the *controllable predecessor* operator $CPre_1: 2^S \mapsto 2^S$, defined for all $X \subseteq S$ by

$$CPre_1(X) = \{s \in S \mid \exists a \in \Gamma_1(s). \forall b \in \Gamma_2(s). \delta(s, a, b) \subseteq X\}.$$

Intuitively, $CPre_1(X)$ consists of all the states from which player 1 can force the game to X in one round; the operator $CPre_2$ for player 2 can be defined symmetrically. For $i \in \{1, 2\}$ and $R \subseteq S$ we have then:

$$\text{win}_i(G, \Box R, \text{sure} \mid \Pi_1^c, \Pi_2^c, \text{PrbC}) = \text{win}_i(G, \Box R, \text{almost} \mid \Pi_1^c, \Pi_2^c, \text{PrbC}) = \nu X. (R \cap \text{CPre}_i(X)), \quad (1)$$

where ν denotes the greatest fixpoint operator. The fixpoint can be computed by Picard iteration by letting $X_0 = S$ and, for $n \geq 0$, $X_{n+1} = R \cap \text{CPre}_i(X_n)$; the solution is then given by the limit $\lim_{n \rightarrow \infty} X_n$, which can be computed in at most $|S|$ iterations.

Reachability. For mode *sure*, the solution of concurrent reachability games is also classical: for player $i \in \{1, 2\}$ and target set $R \subseteq S$, we have

$$\text{win}_i(G, \Diamond R, \text{sure} \mid \Pi_1^c, \Pi_2^c, \text{PrbC}) = \mu X. (R \cup \text{CPre}_i(X)) \quad (2)$$

where μ denotes the least fixpoint operator. The solution can again be computed iteratively, as the limit $\lim_{n \rightarrow \infty} X_n$ of the sequence $X_0, X_1, X_2, \dots$ defined by $X_0 = \emptyset$, and for $n \geq 0$, by $X_{n+1} = R \cup \text{CPre}_i(X_n)$. The solution for mode *almost* and player $i \in \{1, 2\}$ relies on the two-argument predecessor operator $\text{APre}_i : 2^S \times 2^S \mapsto 2^S$ [dAHK98, dAH00]. For $X, Y \subseteq S$, we have $s \in \text{APre}_i(Y, X)$ iff player i can force the game to stay in Y , while at the same time forcing a transition to X with positive probability:

$$\text{APre}_1(Y, X) = \{s \in S \mid \forall b \in \Gamma_2(s). \exists a \in \text{Safe}_1(s, Y, \Gamma_2(s)). \delta(s, a, b) \cap X \neq \emptyset\}.$$

The operator APre_2 for player 2 can be defined symmetrically. The set of states from which player $i \in \{1, 2\}$ wins with probability 1 with respect to the winning condition $\Diamond R$ can then be computed as a nested fixpoint [dAHK98, dAH00]:

$$\text{win}_i(G, \Diamond R, \text{sure} \mid \Pi_1^c, \Pi_2^c, \text{PrbC}) = \nu Y. \mu X. (R \cup \text{APre}_i(Y, X)). \quad (3)$$

To understand this algorithm, let Y^* set of winning states computed by (3). Since $Y^* = \mu X. (R \cup \text{APre}_i(Y^*, X))$, we can write $Y^* = \lim_{k \rightarrow \infty} X_k$, where $X_0 = R$, and for $k \geq 0$, where $X_{k+1} = (R \cup \text{APre}_i(Y^*, X_k))$. For $k \geq 0$, from $X_{k+1} \setminus X_k$ player i can ensure some probability of going to X_k , while never leaving Y^* . Hence, from any state in Y^* , player i can play a sequence of $|Y^*|$ rounds that ensure that (i) R is reached with positive probability, and (ii) Y^* is left only after R is reached. By repeating this $|Y^*|$ -round sequence indefinitely, player i is able to reach R with probability 1. The following theorem summarizes the results on concurrent games.

Theorem 1 [dAHK98] *For all game structures G , players $i \in \{1, 2\}$, and sets $R \subseteq S$, the following assertions hold.*

Safety. *For $\mathcal{M} \in \{\text{sure}, \text{almost}\}$, the set $\text{win}_i(G, \Box R, \mathcal{M} \mid \Pi_1^c, \Pi_2^c, \text{PrbC})$ can be computed in time $\mathcal{O}(|G|)$ by (1). The game type $(\Box, i, \mathcal{M} \mid \Pi_1^c, \Pi_2^c, \text{PrbC})$ is determined, and there always exist winning strategies that are both deterministic and memoryless.*

Sure reachability. *The set $\text{win}_i(G, \Diamond R, \text{sure} \mid \Pi_1^c, \Pi_2^c, \text{PrbC})$ can be computed in time $\mathcal{O}(|G|)$ by (2). The game type $(\Diamond, i, \text{sure} \mid \Pi_1^c, \Pi_2^c, \text{PrbC})$ is determined, and there always exist winning strategies that are both deterministic and memoryless.*

Almost sure reachability. *The set $\text{win}_i(G, \Diamond R, \text{almost} \mid \Pi_1^c, \Pi_2^c, \text{Prb}C)$ can be computed in time $\mathcal{O}(|G|^2)$ by (3). The game type $(\Diamond, i, \text{almost} \mid \Pi_1^c, \Pi_2^c, \text{Prb}C)$ is determined, there always exist winning strategies that are memoryless, but the existence of deterministic winning strategies is not guaranteed.*

3 Semi-concurrent Games

In *semi-concurrent* games, one of the players, when choosing her move, has access to a bounded amount of information about the opponent's choice of move. To model inter-player communication within a round, we introduce semi-concurrent strategies. Let $\Sigma_k = \{1, 2, \dots, k\}$. A *semi-concurrent strategy* of order $k > 0$ for player $i \in \{1, 2\}$ is a pair $\pi_i = \langle \pi_i^s, \pi_i^d \rangle$ consisting of a *spy strategy* $\pi_i^s : \text{Hist} \times M \mapsto \mathcal{D}(\Sigma_k)$ and of a *decision strategy* $\pi_i^d : \text{Hist} \times \Sigma_k \mapsto \mathcal{D}(M)$, such that for all $\sigma \in \text{Hist}$, all $1 \leq j \leq k$ and all $a \in M$, we have that $\pi_i^d(\sigma, j)(a) > 0$ implies $a \in \Gamma_i(\text{last}(\sigma))$. The spy strategy represents the method used by player i to gather information about the opponent's move: after the history σ , if the opponent chooses move b , one of the integers in Σ_k is received by player i , according to the distribution $\pi_i^s(\sigma, b)$. Once player i receives an integer n , it chooses the move $a \in M$ with probability $\pi_i^d(\sigma, n)(a)$. Note that semi-concurrent strategies of order 1 are essentially concurrent strategies, as the only symbol carries no information, and semi-concurrent strategies of order $|M|$ give rise to turn-based games, since one of the players can obtain full information about the move of the other. For $k > 0$, we let $\tilde{\Pi}_i^k$ be the set of all semi-concurrent strategies of order k for player $i \in \{1, 2\}$.

A *semi-concurrent game* is a game where one player uses semi-concurrent strategies, and the other uses concurrent strategies. We arbitrarily fix player 1 to be the player using semi-concurrent strategies. Hence, we consider the families of strategies $\langle \tilde{\Pi}_1^k, \Pi_2^c, \text{PrbS} \rangle$ for $k > 0$, where for $\pi_1 = \langle \pi_1^s, \pi_1^d \rangle \in \tilde{\Pi}_1^k$ and $\pi_2 \in \Pi_2^c$, $\text{PrbS}_{s_0}^{\pi_1, \pi_2}$ is defined inductively on histories by $\text{PrbS}_{s_0}^{\pi_1, \pi_2}(s_0) = 1$ and, for $n \geq 0$ and $\sigma \in \text{Hist}_{s_0}$, by

$$\begin{aligned} \text{PrbS}_{s_0}^{\pi_1, \pi_2}(\sigma, \langle a_n, b_n \rangle, s_{n+1}) &= \text{PrbS}_{s_0}^{\pi_1, \pi_2}(\sigma) \cdot \pi_2(\sigma)(b_n) \cdot \tau(\text{last}(\sigma), a_n, b_n)(s_{n+1}) \\ &\quad \cdot \sum_{j \in \Sigma_k} \pi_1^d(\sigma, j)(a_n) \cdot \pi_1^s(\sigma, b_n)(j). \end{aligned}$$

Again, these probabilities on histories give rise to a probability measure $\text{PrbS}_{s_0}^{\pi_1, \pi_2}$ on Ω_{s_0} .

In general, both the spy strategy π_i^s and the decision strategy π_i^d can be history-dependent and randomized. For $i \in \{1, 2\}$, we say that a decision strategy π_i^d of order k is *memoryless* if $\pi_i^d(\sigma, j) = \pi_i^d(\text{last}(\sigma), j)$ for all $\sigma \in \text{Hist}$ and all $j \in \{1, 2, \dots, k\}$, and we say that π_i^d is *deterministic* if $\pi_i^d(\sigma, j)(a) \in \{0, 1\}$ for all $\sigma \in \text{Hist}$, all $j \in \{1, 2, \dots, k\}$, and all $a \in M$. Analogously, for $i \in \{1, 2\}$ and $k > 0$, we say that a spy strategy π_i^s is *memoryless* if $\pi_i^s(\sigma, b) = \pi_i^s(\text{last}(\sigma), b)$ for all $\sigma \in \text{Hist}$ and all $b \in M$, and we say that π_i^s is *deterministic* if $\pi_i^s(\sigma, b)(j) \in$

$\{0, 1\}$, for all $\sigma \in \text{Hist}$, all $j \in \{1, 2, \dots, k\}$, and all $b \in M$. We say that a semi-concurrent strategy $\pi_1 = \langle \pi_1^s, \pi_1^d \rangle$ is *memoryless* (respectively *deterministic*) if both π_1^s and π_1^d are memoryless (resp. *deterministic*).

3.1 Semi-concurrent Safety Games

Since the information in each round flows from player 2 to player 1, the solution of semi-concurrent safety games is not symmetrical with respect to players 1 and 2. In order to win a safety game, player 1 must be able at each round to issue a move that keeps the game into the safe region, regardless of the opponent's move. If player 1 can use an order- k semi-concurrent strategy, the best approach consists, at each round, in partitioning the moves of player 2 in k groups, and in using the spy strategy to communicate the group of the move chosen by player 2. If player 1 has a move for each of the k groups that ensures the game stays in the safe region, player 1 can win the game. Hence, we define the *order- k semi-concurrent* predecessor operator $S\text{Pre}_1^k : 2^S \mapsto 2^S$ as follows. A *k -partition* of a set A consists in k subsets $A_1, \dots, A_k \subseteq A$ such that $A = \bigcup_{j=1}^k A_j$.

For all $X \subseteq S$ and $s \in S$, we have $s \in S\text{Pre}_1^k(X)$ iff there is a k -partition $B_1, \dots, B_k$ of $\Gamma_2(s)$ and $a_1, \dots, a_k \in \Gamma_1(s)$ (possibly not all distinct) such that, for all $b \in \Gamma_2(s)$, if $b \in B_j$ then $\delta(s, a_j, b) \subseteq X$.

Thus, when player 2 chooses move $b \in B_j$, player 1 can force the game to X by playing move a_j .

When player 2 tries to win a safety game using a concurrent strategy, the fact that player 1 uses a concurrent strategy, or a semi-concurrent strategy, is irrelevant: in fact, if player 2 had a move that guaranteed safety when not spied upon, the same move will guarantee safety also when spied upon by player 1. Thus, the game can be solved with the usual controllable predecessor operator $C\text{Pre}_2$. The following theorem summarizes the results about semi-concurrent safety games.

Theorem 2 *For all game structures G , sets $R \subseteq S$, $k > 1$, and $\mathcal{M} \in \{\text{sure}, \text{almost}\}$, the following assertions hold:*

1. *We have $\text{win}_1(G, \Box R, \mathcal{M} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S) = \nu X. (R \cap S\text{Pre}_1^k(X))$; the set can be computed in time $\mathcal{O}(|G|^k)$. There are always winning strategies that are both deterministic and memoryless.*
2. *We have $\text{win}_2(G, \Box R, \mathcal{M} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S) = \text{win}_2(G, \Box R, \mathcal{M} \mid \Pi_1^c, \Pi_2^c, \text{Prb})$, and as in the case of concurrent games, the above sets can be computed in time $\mathcal{O}(|G|)$ by (1). There are always winning strategies that are both deterministic and memoryless.*

As for determinacy, the following theorem holds.

Theorem 3 *For $i \in \{1, 2\}$, the game type $(\Box, i, \mathcal{M} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S)$ is determined.*

If we consider the order $k > 0$ to be part of the input, we obtain the following *NP*-completeness result. The result is proved by reducing Vertex Cover [GJ79] to the problem of computing $S\text{Pre}_1^k(\cdot)$.

Theorem 4 *Given input (k, G, R) , the membership problem in $\text{win}_1(G, \square R, \mathcal{M} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S)$ is *NP*-complete.*

3.2 Player One Reachability Games

In order to win a reachability game with mode *sure*, player 1 must guarantee that, at each round, deterministic progress is made toward the goal. Hence, the solution of player 1 reachability games for mode *sure* uses again the controllable predecessor operator $S\text{Pre}_1^k$.

When the desired winning mode is *almost*, rather than *sure*, the game is solved using a semi-concurrent version $SAPre_1^k$ of the operator $A\text{Pre}_1$ for concurrent games, for $k > 0$. Again, the best approach for player 1 consists in partitioning the adversary's moves into k subsets $B_1, \dots, B_k$, using the spy strategy to learn the subset in which the move played by player 2 lies. Thus, if the conditions of operator $A\text{Pre}_1$ hold for each subset $B_1, \dots, B_k$ of moves, then player 1 is able to ensure probabilistic progress toward the goal. The definition is as follows.

Given two sets $X, Y \subseteq S$ and a state $s \in S$, we say that $s \in SAPre_1^k(Y, X)$ if and only if there exist a k -partition $B_1, \dots, B_k$ of $\Gamma_2(s)$ such that, for all $b \in \Gamma_2(s)$, if $b \in B_j$ then there is $a \in \text{Safe}_1(s, Y, B_j)$ such that $\delta(s, a, b) \cap X \neq \emptyset$.

Theorem 5 *For all game structures G , $R \subseteq S$, and $k > 1$, the following assertions hold:*

1. *We have $\text{win}_1(G, \diamond R, \text{sure} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S) = \mu X. (R \cup S\text{Pre}_1^k(X))$; the fixpoint can be computed in time $\mathcal{O}(|G|^k)$. There always exist deterministic and memoryless winning strategies.*
2. *We have $\text{win}_1(G, \diamond R, \text{almost} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S) = \nu Y. \mu X. (R \cup SAPre_1^k(Y, X))$. Deciding whether a state belongs to the above fixpoint is *NP*-complete in $|G|$. There always exist memoryless winning strategies, but there may not be deterministic winning strategies.*

The theorem states, in particular, that computing the set of winning states of a player 1 semi-concurrent reachability game is an *NP*-complete problem even when $k = 2$, i.e., when the spy strategies can communicate at most 1 bit of information about player 2's choice of move. The *NP*-completeness result is proved by reducing 3-SAT to the problem of deciding $s \in SAPre_1^2(\cdot, \cdot)$. Then, it is shown that the result for $k = 2$ implies the result for all $k > 1$.

Theorem 6 *The following assertions hold:*

1. *The game type $(\diamond, 1, \text{sure} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S)$ is determined.*
2. *The game type $(\diamond, 1, \text{almost} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S)$ is not determined. On the other hand, if only memoryless spy strategies are considered, the latter game type is determined.*

The following theorem states that the more information is available to player 1, the more games player 1 can win.

Theorem 7 *For $\Delta \in \{\diamond, \square\}$ and $\mathcal{M} \in \{\text{sure}, \text{almost}\}$, if $k_1 > k_2 > 0$ then there is a game structure G and a subset of states $R \subseteq S$ such that*

$$\text{win}_1(G, \Delta R, \mathcal{M} \mid \vec{\Pi}_1^{k_2}, \Pi_2^c, \text{Prb}S) \subsetneq \text{win}_1(G, \Delta R, \mathcal{M} \mid \vec{\Pi}_1^{k_1}, \Pi_2^c, \text{Prb}S).$$

The theorem is proved by constructing a game where player 1 has moves $a_1, \dots, a_m$, and player 2 has moves $b_1, \dots, b_m$. In order to win, player 1 must match each move b_j , for $1 \leq j \leq m$, with move a_j . Obviously, player 1 can do this only in a semi-concurrent game of order $k \geq m$. Since semi-concurrent games of order 1 are concurrent games, the following corollary follows.

Corollary 1 *For $\Delta \in \{\diamond, \square\}$, $\mathcal{M} \in \{\text{sure}, \text{almost}\}$, and $k > 1$, there is a game structure G and a subset of states $R \subseteq S$ such that*

$$\text{win}_1(G, \Delta R, \mathcal{M} \mid \Pi_1^c, \Pi_2^c, \text{Prb}S) \subsetneq \text{win}_1(G, \Delta R, \mathcal{M} \mid \vec{\Pi}_1^{k_1}, \Pi_2^c, \text{Prb}S).$$

3.3 Player Two Reachability Games

We now consider the case when player 2 has to reach a region R , while player 1 is able to gather information about her choice of moves using a semi-concurrent strategy. Again, for winning mode *sure*, the solution of the game coincides with that of concurrent games. Informally, if player 2 must ensure that *all* outcome plays reach R (as opposed to a set of outcome plays with measure 1), player 1 does not need to get information about player 2's choice of moves: he can just guess it. For mode *almost*, semi-concurrent reachability games of order k is solved using a predecessor operator $VAPre_2^k : 2^S \times 2^S \mapsto 2^S$, that plays the same role as the operator $APre_2$ for concurrent games: for $X \subseteq Y \subseteq S$, the set $VAPre_2^k(Y, X) \subseteq S$ consists of the states from which player 2 can ensure a positive probability of going to X in one round, while staying in Y .

For $s \in S$ and $X, Y \subseteq S$, we have $s \in VAPre_2^k(Y, X)$ iff for all k -partitions $B_1, \dots, B_k$ of $\text{Safe}_2(s, Y, \Gamma_2(s))$, there is $j \in \{1, \dots, k\}$ such that:

$$\forall a \in \Gamma_1(s). \exists b \in B_j. \delta(s, a, b) \cap X \neq \emptyset.$$

The idea is as follows: the best strategy for player 2 at a state $s \in S$ consists in playing uniformly at random all moves in $\text{Safe}_2(s, Y, \Gamma_2(s))$; the above definition ensures that, if $s \in \text{VAPre}_2^k(Y, X)$, then a transition to X happens with positive probability, regardless of the partition of $\Gamma_2(s)$ chosen by player 1's spy strategy. We are now ready to state the following:

Theorem 8 *For all game structures G , $R \subseteq S$, and $k > 1$, the following holds:*

1. *We have $\text{win}_2(G, \Diamond R, \text{sure} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S) = \text{win}_1(G, \Diamond R, \text{sure} \mid \Pi_1^c, \Pi_2^c, \text{Prb})$; as in the case of concurrent games, the above sets can be computed in time $\mathcal{O}(|G|)$ by (2). There are always memoryless and deterministic winning strategies.*
2. *We have $\text{win}_2(G, \Diamond R, \text{almost} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S) = \nu Y. \mu X. (R \cup \text{VAPre}_2^k(Y, X))$; the fixpoint can be computed in time $\mathcal{O}(|G|^k)$. There are always winning strategies that are memoryless, but there may not be deterministic winning strategies.*

Theorem 9 *For $\mathcal{M} \in \{\text{sure}, \text{almost}\}$, the game type $(\Diamond, 1, \mathcal{M} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S)$ is determined.*

If we consider the order $k > 0$ to be part of the input, we obtain the following result (compare with Theorem 4). The result is proved by reducing Vertex Cover to non-membership in $\text{VAPre}_2^k(\cdot, \cdot)$.

Theorem 10 *Given input (k, G, R) , the membership problem in $\text{win}_2(G, \Box R, \text{almost} \mid \vec{\Pi}_1^k, \Pi_2^c, \text{Prb}S)$ is co-NP-complete.*

4 Team Games

In *team games*, one of the players does not consist of a single player, but rather of a team, composed by *members*. At each state, each team member can choose a move; the resulting team move is a tuple consisting of the choices of the members. We assume that each team member has complete information about the past history of the game, and we explicitly model the coordination used by the team members in choosing their moves. In particular, we consider both *non-communicating* and *communicating* strategies for team members. When using non-communicating strategies, the team members must select their moves simultaneously and independently not only from the opposing player, but also from each other. When using communicating strategies, the team members are allowed to communicate some information before choosing the moves.

Formally, a (m_1, m_2) -team game structure is a concurrent game structure $G = (S, M, \Gamma_1, \Gamma_2, \tau)$, where $M = \prod_{j=1}^{m_1} M_1^j \cup \prod_{j=1}^{m_2} M_2^j$, and for all $s \in S$, $\Gamma_1(s) = \prod_{j=1}^{m_1} \Gamma_1^j(s)$ and $\Gamma_2(s) = \prod_{j=1}^{m_2} \Gamma_2^j(s)$. Intuitively, the game is played by teams 1 and 2, and the team $i \in \{1, 2\}$ is composed by members $1, \dots, m_i$. At a state $s \in S$, the set $\Gamma_i^j(s)$ contains the moves that can be chosen by member $j \in \{1, \dots, m_i\}$. A move of team i is thus a tuple $\langle a_1, \dots, a_{m_i} \rangle$, consisting of the choices of the members. We let Π_1^c and Π_2^c be the sets of concurrent strategies for G , as defined in Section 2.

4.1 Team Games with Non-communicating Strategies

A *non-communicating team strategy* for team $i \in \{1, 2\}$ is a function $\bar{\pi}_i : \text{Hist} \mapsto \mathcal{D}(M)$ that prescribes for each game history a move distribution to be played by the team. Since we forbid communication between the members of the team, the distributions chosen by the team members must be mutually independent. Hence, we require that there are m_i functions $\pi_i^j : \text{Hist} \mapsto \mathcal{D}(M_i^j)$, for $1 \leq j \leq m_i$, such that $\bar{\pi}_i(\sigma)(\langle a_1, \dots, a_{m_i} \rangle) = \prod_{j=1}^{m_i} \pi_i^j(\sigma)(a_j)$ for all $\sigma \in \text{Hist}$ and all $\langle a_1, \dots, a_{m_i} \rangle \in \Gamma_i(s)$. We denote by $\bar{\Pi}_i$ the set of all team strategies for team $i \in \{1, 2\}$. In a (m_1, m_2) -team game we have $\bar{\Pi}_i \subseteq \Pi_i^c$, and the inclusion is strict whenever $m_i > 1$ and there is a state $s \in S$ with $|\Gamma_i(s)| > 1$. The probability measure $\text{PrbT}_s^{\pi_1, \pi_2}$ for $s \in S$ and $\pi_1 \in \bar{\Pi}_1$, $\pi_2 \in \bar{\Pi}_2$ can then be defined in a straightforward way, yielding the family of (m_1, m_2) -team strategies $\langle \bar{\Pi}_1, \bar{\Pi}_2, \text{PrbT} \rangle$.

A team game with non-communicating strategies (also called *non-communicating team game*) differs from a concurrent game because, at each state, each team must choose a probability distribution over moves that can be written as the product of the distributions chosen by the team members. Since deterministic distributions can always be written in product form (if team i wants to play tuple $\langle a_1, \dots, a_{m_i} \rangle$, each member $1 \leq j \leq m_i$ simply plays a_j), for the games where the existence of deterministic winning strategies is assured, the winning states of non-communicating team games coincide with the winning states of concurrent games.

Theorem 11 *For all $m_1, m_2 > 0$, all (m_1, m_2) -team game structures G , sets $R \subseteq S$, and teams $i \in \{1, 2\}$, the following assertions hold:*

1. *For $\mathcal{M} \in \{\text{sure}, \text{almost}\}$, we have $\text{win}_i(G, \Box R, \mathcal{M} \mid \bar{\Pi}_1, \bar{\Pi}_2, \text{PrbT}) = \text{win}_i(G, \Box R, \mathcal{M} \mid \Pi_1^c, \Pi_2^c, \text{Prb})$. There are always winning strategies that are memoryless and deterministic.*
2. *We have $\text{win}_i(G, \Diamond R, \text{sure} \mid \bar{\Pi}_1, \bar{\Pi}_2, \text{PrbT}) = \text{win}_i(G, \Diamond R, \text{sure} \mid \Pi_1^c, \Pi_2^c, \text{Prb})$. There are always winning strategies that are memoryless and deterministic.*

Corollary 2 *For $\mathcal{M} \in \{\text{sure}, \text{almost}\}$ and $i \in \{1, 2\}$, the game type $(\Box, i, \mathcal{M} \mid \bar{\Pi}_1, \bar{\Pi}_2, \text{PrbT})$ is determined. The game type $(\Diamond, i, \text{sure} \mid \bar{\Pi}_1, \bar{\Pi}_2, \text{PrbT})$ is also determined.*

Hence, the interesting problem in team games consists in solving reachability games with probability 1, where the winning strategies need randomization in the general case [dAHK98]. Such games can be solved using the predecessor operator TAPre_1 , defined as follows. In the following, we call *cube* any set $C \in \prod_{j=1}^{m_1} (2^{M_1^j} \setminus \{\emptyset\})$.

Given two sets $X, Y \subseteq S$ and a state $s \in S$, we say that $s \in \text{TAPre}_1(Y, X)$ if and only if there exists a cube $C \in \prod_{j=1}^{m_1} (2^{M_1^j} \setminus \{\emptyset\})$ such that:

$$\forall b \in \Gamma_2(s). (\forall a \in C. \delta(s, a, b) \subseteq Y \text{ and } \exists a \in C. \delta(s, a, b) \subseteq X).$$

Theorem 12 *For all $(2,1)$ -team game structures G and sets $R \subseteq S$, we have $\text{win}_1(G, \Diamond R, \text{almost} \mid \bar{\Pi}_1, \bar{\Pi}_2, \text{Prb}T) = \nu Y. \mu X. (R \cup TAPre_1(Y, X))$, and membership in this fixpoint is an NP-complete problem. Moreover, there are always winning strategies that are memoryless, but there may not be deterministic winning strategies.*

In order to prove the NP-hardness, a non-trivial reduction is developed, transforming the classical 3-CNF-SAT problem to membership in $TAPre_1(\cdot, \cdot)$.

By means of a counterexample, the following can be shown.

Theorem 13 *For $i \in \{1, 2\}$, the game type $(\Diamond, i, \text{almost} \mid \bar{\Pi}_1, \bar{\Pi}_2, \text{Prb}T)$ is not determined.*

4.2 Team Games with Communication

In this section we consider the case in which members of the same team are allowed to communicate some information in each turn of the game. To simplify the notation, rather than considering arbitrary flows of information between team members, we consider a team composed of only two members. This special case suffices to capture the interesting features of the general case.

Formally, given a $(2,1)$ -team game structure, a *communicating team strategy of order $k > 0$* for team 1 is a function $\hat{\pi}_1^k : \text{Hist} \mapsto \mathcal{D}(M)$ subject to the following requirements. There are three functions $\pi_1^t : \text{Hist}(G) \times \Sigma_k^* \mapsto \mathcal{D}(\Sigma_k)$, $\pi_1^1 : \text{Hist}(G) \times \Sigma_k^* \mapsto \mathcal{D}(M_1^1)$, and $\pi_1^2 : \text{Hist}(G) \times \Sigma_k^* \mapsto \mathcal{D}(M_1^2)$. The function π_1^t represents the generation of random symbols; these symbols are then communicated to the team members, which then use the functions π_1^1 and π_1^2 to choose their moves, on the basis of the game history and of the received symbols. Note that if we consider both the generation of symbols π_1^t and the choice π_1^i to be done by team member i , for $i \in \{1, 2\}$, then communication effectively takes place from team member i to team member $3 - i$. For $n \geq 0$, $\langle r_0, \dots, r_n \rangle \in \Sigma_k^*$ and $\langle s_0, \dots, s_{n+1} \rangle \in \text{Hist}$, we set

$$Pr_{s_0}^{\pi_1^t}(\langle r_0, \dots, r_n \rangle \mid \langle s_0, \dots, s_n \rangle) = \prod_{j=0}^n \pi_1^t(\langle s_0, \dots, s_j \rangle, \langle r_0, \dots, r_{j-1} \rangle)(r_j),$$

with the convention that $r_0, \dots, r_{-1} = \varepsilon$ (where ε denotes the empty string). Then, for all $n \geq 0$, all $\sigma = \langle s_0, \dots, s_n \rangle \in \text{Hist}_{s_0}$, all $a_1 \in M_1^1$, and all $a_2 \in M_1^2$ we define the overall team strategy $\hat{\pi}_1^k$ by

$$\hat{\pi}_1^k(\sigma)(\langle a_1, a_2 \rangle) = \sum_{\rho \in \Sigma_k^n} Pr_{s_0}^{\pi_1^t}(\rho \mid \sigma) \cdot \pi_1^1(\sigma, \rho)(a_1) \cdot \pi_1^2(\sigma, \rho)(a_2).$$

We denote by $\hat{\Pi}_i^k$ the set of communicating team strategies of order k for team i , and by PrbTC the probability measure on Ω induced by $\hat{\Pi}_1^k$ and Π_2^c , defined as for concurrent games. We prove that, for all $k > 1$, team 1 has a winning strategy if and only if player 1 has a winning strategy in the corresponding concurrent game.

Theorem 14 For all $(2,1)$ -team game structures G , sets $R \subseteq S$, and $k > 1$, we have $\text{win}_1(G, \Diamond R, \text{almost} \mid \widehat{\Pi}_1^k, \Pi_2^c, \text{PrbTC}) = \text{win}_1(G, \Diamond R, \text{almost} \mid \Pi_1^c, \Pi_2^c, \text{Prb})$.

This theorem implies that, from the point of view of winning reachability games with probability 1, being able to communicate 1 bit per round, or even just sharing 1 bit of random information, is as good as the ability to communicate an arbitrary amount of information.

We outline the idea of the proof for $k = 2$, the case for a one-bit channel. Consider the solution $Y^* = \text{win}_1(G, \Diamond R, \text{almost} \mid \Pi_1^c, \Pi_2^c, \text{Prb}) = \mu X.(R \cup A\text{Pre}_i(Y^*, X))$ of the concurrent reachability game. As remarked in Section 2.2, if the team members could coordinate perfectly, they could play a sequence of $|Y^*|$ moves that leads to R with positive probability, and that does not leave Y^* otherwise. The problem, here, is that the two team members cannot communicate perfectly. To communicate a sequence of $|Y^*|$ moves, they need $|Y^*| \cdot \lceil \log |M| \rceil$ bits. However, the two team members can play a *deterministic* strategy to stay in Y^* .

The winning strategy of the team thus consists in the alternation of two phases, a *planning* phase, and an *execution* phase. In the planning phase, which lasts $|Y^*| \cdot \lceil \log |M| \rceil$ rounds, the two team members play a deterministic strategy to stay in Y^* . In the meantime, the symbol generator generates $|Y^*| \cdot \lceil \log |M| \rceil$ random bits (notice that the bits are not visible to the adversary). In the subsequent execution phase, which lasts $|Y^*|$ rounds, the team members use the sequence of bits to coordinate their actions, and play at each $s \in Y^*$ all the moves in $\text{Safe}_1(s, Y^*, \Gamma_2(s))$ with strictly positive probability. It is easy to see that each cycle consisting in a planning and in an execution phase results in (i) either reaching R , or (ii) in remaining in Y^* , and outcome (i) occurs with positive probability. This leads to the result. The result can be easily extended to the case of more than two team members.

References

- [AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. 38th IEEE Symp. Found. of Comp. Sci.*, pages 100–109. IEEE Computer Society Press, 1997.
- [BL69] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.
- [dAH00] L. de Alfaro and T.A. Henzinger. Concurrent omega-regular games. In *Proc. 15th IEEE Symp. Logic in Comp. Sci.*, pages 141–154, 2000.
- [dAHK98] L. de Alfaro, T.A. Henzinger, and O. Kupferman. Concurrent reachability games. In *Proc. 39th IEEE Symp. Found. of Comp. Sci.*, pages 564–575. IEEE Computer Society Press, 1998.
- [dAHM00] L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. The control of synchronous systems. In *CONCUR 00: Concurrency Theory. 11th Int. Conf.*, volume 1877 of *Lect. Notes in Comp. Sci.*, pages 458–473. Springer-Verlag, 2000.

- [dAHM01] L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. The control of synchronous systems part II. In *CONCUR 01: Concurrency Theory. 12th Int. Conf.*, volume 2154 of *Lect. Notes in Comp. Sci.*, pages 566–581. Springer-Verlag, 2001.
- [EJ91] E.A. Emerson and C.S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proc. 32nd IEEE Symp. Found. of Comp. Sci.*, pages 368–377. IEEE Computer Society Press, 1991.
- [FV97] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.
- [GH82] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proc. 14th ACM Symp. Theory of Comp.*, pages 60–65. ACM Press, 1982.
- [GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Co., 1979.
- [KV97] O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *2nd International Conference on Temporal Logic*, pages 91–106, Manchester, July 1997.
- [KV01] O. Kupferman and M.Y. Vardi. Synthesizing distributed systems. In *Proc. 16th IEEE Symp. on Logic in Computer Science*, July 2001.
- [PR79] G.L. Peterson and J.H. Reif. Multiple-person alternation. In *Proc. 20th IEEE Symp. Found. of Comp. Sci.*, pages 348–363, 1979.
- [PR90] A. Pnueli and R. Rosner. Distributed-reactive systems are hard to synthesize. In *Proc. 31th IEEE Symp. Found. of Comp. Sci.*, pages 746–757, 1990.
- [Rei84] J.H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29:274–301, 1984.
- [Sha53] L.S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. USA*, 39:1095–1100, 1953.
- [Tho95] W. Thomas. On the synthesis of strategies in infinite games. In *Proc. of 12th Annual Symp. on Theor. Asp. of Comp. Sci.*, volume 900 of *Lect. Notes in Comp. Sci.*, pages 1–13. Springer-Verlag, 1995.
- [Wil91] D. Williams. *Probability With Martingales*. Cambridge University Press, 1991.
- [Zie98] Wiesaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200:135–183, June 1998.

Impact of Local Topological Information on Random Walks on Finite Graphs

Satoshi Ikeda¹, Izumi Kubo², Norihiro Okumoto³, and Masafumi Yamashita⁴

¹ Department of Computer Science,
Tokyo University of Agriculture and Technology,
Naka-cho 2-24-16, Koganei, Tokyo, 184-8588, Japan.
`bisu@cc.tuat.ac.jp`

² Department of Environmental Design, Faculty of Environmental Studies,
Hiroshima Institute of Technology,
2-1-1 Miyake, Saeki-ku Hiroshima 731-5193, Japan.
`kubo@cc.it-hiroshima.ac.jp`

³ Financial Information Systems Division, Hitachi,Ltd.
890 Kashimada, Saiwai-ku, Kawasaki, Kanagawa, 212-8567, Japan.
`okumoto@itg.hitachi.co.jp`

⁴ Department of Computer Science and Communication Engineering,
Kyushu University,
Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan.
`mak@csce.kyushu-u.ac.jp`

Abstract. It is just amazing that both of the mean hitting time and the cover time of a random walk on a finite graph, in which the vertex visited next is selected from the adjacent vertices at random with the same probability, are bounded by $O(n^3)$ for any undirected graph with order n , despite of the lack of global topological information. Thus a natural guess is that a better transition matrix is designable if more topological information is available. For any undirected connected graph $G = (V, E)$, let $P^{(\beta)} = (p_{uv}^{(\beta)})_{u,v \in V}$ be a transition matrix defined by

$$p_{uv}^{(\beta)} = \frac{\exp[-\beta U(u, v)]}{\sum_{w \in N(u)} \exp[-\beta U(u, w)]} \text{ for } u \in V, v \in N(u),$$

where β is a real number, $N(u)$ is the set of vertices adjacent to a vertex u , $\deg(u) = |N(u)|$, and $U(\cdot, \cdot)$ is a potential function defined as

$$U(u, v) = \log(\max\{\deg(u), \deg(v)\}) \text{ for } u \in V, v \in N(u).$$

In this paper, we show that for any undirected graph with order n , the cover time and the mean hitting time with respect to $P^{(1)}$ are bounded by $O(n^2 \log n)$ and $O(n^2)$, respectively. We further show that $P^{(1)}$ is best possible with respect to the mean hitting time, in the sense that the mean hitting time of a path graph of order n , with respect to *any* transition matrix, is $\Omega(n^2)$.

1 Introduction

Random walks on finite graphs are rich source of attractive researches both in applied mathematics and in computer science. Blom et al. [3, Chap. 12] surveyed works devoted to the cover time, which is the expected number of moves necessary for a random walk on a finite undirected connected graph $G = (V, E)$ to visit all vertices, where the vertex visited next is selected from the adjacent vertices at random with the same probability (see also [1,4,5,6,7,11,12,14]). The transition matrix $P = (p_{uv})_{u,v \in V} \in [0, 1]^{V \times V}$ on $V \times V$ is hence given by

$$p_{uv} = \begin{cases} 1/\deg(u) & \text{if } v \in N(u), \\ 0 & \text{otherwise,} \end{cases}$$

where $N(u)$ is the set of vertices adjacent to a vertex u , and $\deg(u) = |N(u)|$. For this transition rule, Aldous [1] showed, for any graph with order n and size m , an upper bound $2m(n-1) (= O(n^3))$ on the cover time, and on the other hand a lower bound $\Omega(n^3)$ is obtained for a lollipop graph L_n shown in Fig. 1. A lollipop graph L_n is a complete graph of order $\lceil n/2 \rceil$ with a tail (i.e., a path graph) of length $\lfloor n/2 \rfloor$. Let s and t be the two endpoints of the tail as shown in Fig. 1. Then the mean hitting time from s to t , i.e., the expected number of moves necessary for a random walk starting at s to reach t , is $\Omega(n^3)$ [12]. Thus both of the mean hitting time and the cover time are $\Theta(n^3)$ in this sense.¹

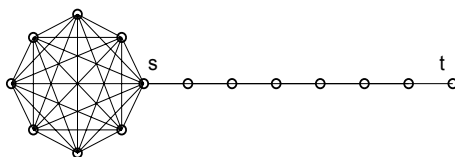


Fig. 1. A lollipop graph L_{15} .

Observing that P depends only on topological information $\deg(u)$ local to each vertex u , let us examine the following plausible claim:

The cover time and the mean hitting time are (properly) reducible by using more topological information to construct a transition matrix.

The claim is indeed correct in many cases. For instance, consider a complete graph K_n , assuming that the whole G is available to construct an ideal transition matrix for G . Then there is a transition matrix Q that achieves the cover time $n-1$, since it has a Hamiltonian circuit,² while the cover time for K_n achieved

¹ For more notes, an upper bound $2m(n-1) (= O(n^3))$ on the cover time of any regular graph is first shown by Aleliunas et al. [2]. Both of the cover time and the mean hitting time are shown to be at most $4n^3/27$ [4,7].

² Let $\{0, 1, \dots, n-1\}$ be the vertex set of K_n and define $Q = (q_{ij})$ by $q_{ij} = 1$ if $j \equiv i+1 \pmod{n}$, and $q_{ij} = 0$ otherwise. Then the cover time for K_n with respect to Q is obviously $n-1$.

by P is $O(n \log n)$. However, there are of course some other cases in which even the complete information G does not help in reducing the cover time.

This paper shows that the bounds $\Theta(n^3)$ on the mean hitting time and the cover time are reducible to $\Theta(n^2)$ and $\Theta(n^2 \log n)$ respectively, in the sense mentioned in the first paragraph, if the topological information on the adjacent vertices is available. For any $\beta \in \mathbf{R}$, let $P^{(\beta)} = (p_{uv}^{(\beta)})_{u,v \in V}$ be a transition matrix defined by

$$p_{uv}^{(\beta)} = \begin{cases} \frac{\exp[-\beta U(u, v)]}{\sum_{w \in N(u)} \exp[-\beta U(u, w)]} & \text{if } v \in N(u), \\ 1 - \sum_{w \in N(u)} p_{uw}^{(\beta)} & \text{if } u = v, \\ 0 & \text{otherwise,} \end{cases}$$

which is known as the Gibbs distribution with respect to a local potential U in statistical mechanics. In this paper, we adopt

$$U(u, v) = \log(\max\{\deg(u), \deg(v)\}).$$

Observe that $P^{(\beta)}$ depends only on the topological information on $N(u)$, and that $P^{(0)} = P$.

We summarize our results.

1. For any transition matrix, the maximum mean hitting time (and hence the cover time) of a path graph with order n is $\Omega(n^2)$.
2. For $P^{(\beta)}$, the maximum mean hitting time is $O(n^{1+\beta})$ if $\beta \geq 1$, and $O(n^{3-\beta})$ if $\beta < 1$. The maximum mean hitting time of any graph with order n is hence always $O(n^2)$ for $P^{(1)}$, which means that $P^{(1)}$ is best possible with respect to the mean hitting time.
3. For $P^{(\beta)}$, the cover time is $O(n^{\beta+1} \log n)$ if $\beta \geq 1$, $O(n^{3-\beta} \log n)$ if $0 < \beta \leq 1$, and $O(n^{3-\beta})$ if $\beta \leq 0$. The cover time of any graph with order n is hence always $O(n^2 \log n)$ for $P^{(1)}$. There is still a possible gap from the lower bound $\Omega(n^2)$ in Item 1.
4. For $P^{(\beta)}$ with $\beta \geq 1$, the cover time of a “glitter star” S_n given in Fig. 3 is $\Omega(n^{1+\beta} \log n)$, i.e., for $P^{(\beta)}$ with $\beta \geq 1$, the cover time is $\Theta(n^{1+\beta} \log n)$.

For the sake of generality, we analyze random walks in a more general setting.

2 Preliminaries

Suppose that $G = (V, E)$ is a finite, undirected, simple connected graph with the order $n = |V|$ and the size $m = |E|$. For $u \in V$, by $N(u) = \{v : \{u, v\} \in E\}$ we denote the set of vertices adjacent to u . Note that $v \in N(u)$ iff $u \in N(v)$. The number of adjacent vertices, denoted by $\deg(u) = |N(u)|$, is called the degree of $u \in V$.

Let $\Omega = V^{\mathbf{N} \cup \{0\}}$ be the set of all infinite sequences of vertices, where $\mathbf{N}$ is the set of natural numbers. For $\omega = (\omega_0, \omega_1, \dots) \in \Omega$, the $(i+1)$ -st element w_i

is denoted by $X_i(\omega)$ for $i \geq 0$. By $\mathcal{M}(\Omega)$ we denote the space of the Markov measures on Ω . Put $\mu \in \mathcal{M}(\Omega)$ with an initial distribution (vector) $\mathbf{q} = (q_v) \in [0, 1]^{V \times V}$ and a transition matrix $Q = (q_{uv}) \in (0, 1]^{V \times V}$. That is, for $\omega = (\omega_0, \omega_1, \dots) \in \Omega$, $t \in \mathbf{N} \cup \{0\}$,

$$X_t : \Omega \rightarrow V, \quad X_t(\omega) = \omega_t,$$

$$\sum_{v \in V} q_v = 1, \quad \mu(X_0(\omega) = v) = q_v \quad \text{for any } v \in V,$$

$$\sum_v q_{uv} = 1 \quad \text{for any } u \in V,$$

and for $u, v, x_0, x_1, \dots, x_i \in V$ and $i \in \mathbf{N} \cup \{0\}$

$$\begin{aligned} \mu(X_{i+1}(\omega) = v | X_0(\omega) = x_0, X_1(\omega) = x_1, \dots, X_i(\omega) = x_i = u) \\ = \mu(X_{i+1}(\omega) = v | X_i(\omega) = u) = q_{uv}. \end{aligned}$$

As we are analyzing random walks on graph $G = (V, E)$, we assume without loss of generality that $q_{uv} > 0$ if $v \in N(u)$, and $q_{uv} = 0$ if $v \notin N(u) \cup \{u\}$. The space of Markov measures that meet the above requirement is denoted by $\mathcal{M}^+(\Omega)$, i.e.,

$$\mathcal{M}^+(\Omega) = \left\{ \mu \in \mathcal{M}(\Omega) : q_{uv} > 0 \text{ if } v \in N(u) \text{ and } q_{uv} = 0 \text{ if } v \notin N(u) \cup \{u\} \right\}.$$

To define the edge cost, let us now introduce a cost matrix $K = (k_{uv}) \in [0, \infty)^{V \times V}$. For $\omega \in \Omega$, put

$$n(\omega) = \inf \left\{ i \in \mathbf{N} : \{X_0(\omega), X_1(\omega), \dots, X_i(\omega)\} = V \right\}.$$

If ω is an infinite legal token circulation on G , $n(\omega)$ denotes the minimum number of token moves necessary to visit all the vertices in V . We are interested in the circulation cost $k(\omega)$ incurred before the token visits all the vertices, i.e.,

$$k(\omega) = \sum_{i=0}^{n(\omega)-1} k_{X_i(\omega)X_{i+1}(\omega)}.$$

For any connected graph $G = (V, E)$, $\mu \in \mathcal{M}^+(\Omega)$, cost matrix K and $u, v \in V$, we define the *weighted mean hitting time* $H_\mu^{G,K}(u, v)$ from u to v with respect to μ and K by

$$H_\mu^{G,K}(u, v) = E_\mu \left[\sum_{i=0}^{t(\omega, v)-1} k_{X_i(\omega)X_{i+1}(\omega)} \middle| X_0(\omega) = u \right],$$

where

$$t(\omega, v) = \inf \left\{ i \geq 1 : X_i(\omega) = v \right\}.$$

In particular, $\max_{u,v \in V} H_\mu^{G,K}(u, v)$ is called the *maximum weighted mean hitting time* of G with respect to μ and K . By the reason of asymmetry either of the graph G or of the cost matrix K , $H_\mu^{G,K}(u, v) \neq H_\mu^{G,K}(v, u)$ may hold. Finally, we define the *weighted cover time* $C_\mu(G, K)$ of G with respect to μ and K by

$$C_\mu(G, K) = \max_{u \in V} C_\mu(G, K, u), \quad C_\mu(G, K, u) = \mathbf{E}_\mu \left[k(\omega) | X_0(\omega) = u \right].$$

Let $Q = (q_{uv})_{u,v \in V}$ be a transition matrix for a Markov measure $\mu \in \mathcal{M}^+(\Omega)$ and $\pi = (\pi_v)_{v \in V}$ be its stationary distribution vector. Since

$$H_\mu^{G,K}(u, v) = \sum_{w \in V} q_{uw} (k_{uw} + H_\mu^{G,K}(w, v)) - q_{uv} H_\mu^{G,K}(v, v), \quad (2.1)$$

we get

$$\sum_{u \in V} \pi_u H_\mu^{G,K}(u, v) = \sum_{w \in V} \pi_w H_\mu^{G,K}(w, v) + \sum_{u \in V} \sum_{w \in V} \pi_u q_{uw} k_{uw} - \pi_v H_\mu^{G,K}(v, v)$$

by the equality $\sum_{u \in V} \pi_u q_{uv} = \pi_v$, which implies that

$$\pi_v H_\mu^{G,K}(v, v) = \bar{k} \quad \text{for all } v \in V, \quad (2.2)$$

where

$$\bar{k} = \bar{k}(\mu, K, G) = \sum_{u \in V} \sum_{v \in V} \pi_u q_{uv} k_{uv}. \quad (2.3)$$

We call $\bar{k} = \bar{k}(\mu, K, G)$ the *weighted average cost* with respect to μ and K , which is the mean value of $k_{X_0(\omega), X_1(\omega)}$ with respect to the stationary measure of $\{X_t\}$. By (2.1) and (2.2), we get

$$H_\mu^{G,K}(u, v) \leq \bar{k} (q_{vu} \pi_v)^{-1} \quad \text{for any } u \in V, v \in N(u). \quad (2.4)$$

Let K^0 be a cost matrix such that $k_{uv} = 1$ if $\{u, v\} \in E$ or $u = v$. Then $\bar{k} = 1$ for K^0 . Let $\pi^{(\beta)} = (\pi_v^{(\beta)}) \in [0, 1]^V$ be the stationary distribution of transition matrix $P^{(\beta)} = (p_{uv}^{(\beta)})_{u,v \in V}$, that is, $P^{(\beta)} \pi^{(\beta)} = \pi^{(\beta)}$, $\sum_{v \in V} \pi_v^{(\beta)} = 1$, and $\pi_v^{(\beta)} \geq 0$ for all $v \in V$. Through out this paper, $\nu^{(\beta)}$ denotes the Markov measure on Ω with $\pi^{(\beta)}$ as the initial distribution and $P^{(\beta)}$ as the transition matrix.

3 Lower Bounds for Path Graph

This section proves a lower bound $\Omega(n^2)$ on the maximum mean hitting time and the cover time of a path graph of order n for any transition matrix.

Theorem 1. *Let $P_n = (V, E)$ be a path graph with order n . Then for any $\mu \in \mathcal{M}^+(\Omega)$ and cost matrix $K = (k_{uv})_{u,v \in V}$,*

$$\frac{1}{2} (\bar{k} n^2 - \sum_{w \in V} q_{vw} k_{vw}) \leq C_\mu(P_n, K) \leq 2 \max_{u,v \in V} H_\mu^{P_n, K}(u, v),$$

where $\bar{k} = \bar{k}(\mu, K, P_n)$ is defined by (2.3).

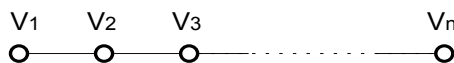


Fig. 2. A path graph with order n .

Proof. Suppose that $P_n = (V, E)$ is a path graph given in Fig. 2.

Let $Q = (q_{uv})_{u,v \in V}$ be any transition matrix for $\mu \in \mathcal{M}^+(\Omega)$ and $\pi = (\pi_v)_{v \in V}$ be its stationary distribution. Then by (2.2) we have

$$\pi_v H_\mu^{P_n, K}(v, v) = \bar{k} \quad \text{for any } v \in V. \quad (3.1)$$

By the definitions of $H_\mu^{P_n, K}$ and $C_\mu(P_n, K)$,

$$\max_{s, t \in V} H_\mu^{P_n, K}(s, t) = \max \{ H_\mu^{P_n, K}(v_1, v_n), H_\mu^{P_n, K}(v_n, v_1) \},$$

$$\max_{s, t \in V} H_\mu^{P_n, K}(s, t) \leq C_\mu(P_n, K),$$

and

$$C_\mu(P_n, K) \leq H_\mu^{P_n, K}(v_1, v_n) + H_\mu^{P_n, K}(v_n, v_1)$$

for any $\mu \in \mathcal{M}^+(\Omega)$ and cost matrix K . Hence

$$\frac{1}{2} \left\{ H_\mu^{P_n, K}(v_1, v_n) + H_\mu^{P_n, K}(v_n, v_1) \right\} \leq C_\mu(P_n, K) \leq 2 \max_{u, v \in V} H_\mu^{P_n, K}(u, v). \quad (3.2)$$

By putting $u = v$ in (2.1),

$$H_\mu^{P_n, K}(v, v) = \sum_{w \in N(v)} q_{vw} H_\mu^{P_n, K}(w, v) + \sum_{w \in V} q_{vw} k_{vw} \quad \text{for any } v \in V.$$

Thus we have

$$\begin{aligned} \bar{k} \sum_{v \in V} \pi_v^{-1} &= \sum_{v \in V} H_\mu^{P_n, K}(v, v) = \sum_{v \in V} \left(\sum_{w \in N(v)} q_{vw} H_\mu^{P_n, K}(w, v) + \sum_{w \in V} q_{vw} k_{vw} \right) \\ &\leq \sum_{v \in V} \sum_{w \in N(v)} H_\mu^{P_n, K}(w, v) + \sum_{v \in V} \sum_{w \in V} q_{vw} k_{vw}, \end{aligned}$$

by (3.1). By Markov property,

$$\sum_{v \in V} \sum_{w \in N(v)} H_\mu^{P_n, K}(w, v) = H_\mu^{P_n, K}(v_1, v_n) + H_\mu^{P_n, K}(v_n, v_1),$$

which implies that

$$H_\mu^{P_n, K}(v_1, v_n) + H_\mu^{P_n, K}(v_n, v_1) \geq \bar{k} \sum_{v \in V} \pi_v^{-1} - \sum_{w \in V} q_{vw} k_{vw} \geq \bar{k} n^2 - \sum_{w \in V} q_{vw} k_{vw}.$$

Together with (3.2), we have

$$\frac{1}{2}(\bar{k}n^2 - \sum_{w \in V} q_{vw}k_{vw}) \leq C_\mu(P_n, K) \leq 2 \max_{u,v \in V} H_\mu^{P_n, K}(u, v).$$

□

By Theorem 1, we have the following lower bounds.

Corollary 1. *For any $\mu \in \mathcal{M}^+(\Omega)$,*

1. $\max_{u,v \in V} H_\mu^{P_n, K^0}(u, v) = \Omega(n^2)$, and
2. $C_\mu(P_n, K^0) = \Omega(n^2)$.

4 Upper Bounds

In Section 3, we showed that both of the mean hitting time and the cover time are bounded from below by $\Omega(n^2)$ for any $\mu \in \mathcal{M}^+(\Omega)$. As mentioned, Aldous [1] showed that for $\nu^{(0)} \in \mathcal{M}^+(\Omega)$, both of them are bounded from above by $O(n^3)$. In this section, we show that for $\nu^{(1)} \in \mathcal{M}^+(\Omega)$, they are bounded by $O(n^2)$ and $O(n^2 \log n)$, respectively. This implies that $\nu^{(1)}$ is best possible with respect to the hitting time. Let us start this section with associating the weighted cover time with the weighted mean hitting time. The following theorem generalizes [11] and [12].

Theorem 2. *Let $G = (V, E)$ be a graph. Then for any $\mu \in \mathcal{M}^+(\Omega)$ and cost matrix K ,*

$$H_{n-1} \min_{u \neq v \in V} H_\mu^{G, K}(u, v) \leq C_\mu(G, K) \leq H_{n-1} \max_{u \neq v \in V} H_\mu^{G, K}(u, v), \quad (4.1)$$

where H_n denotes the n -th harmonic number, i.e., $H_n = \sum_{i=1}^n i^{-1}$.

Proof. Let S_V be the set of all permutations of V and ν be the uniform measure on S_V . For $\pi = (v_1, v_2, \dots, v_n) \in S_V$, we put $\sigma_j(\pi) = v_j$.

For a fixed $u \in V$, let ν_u be the conditional measure of ν conditioned by the set $\{\pi : \sigma_1(\pi) = u\}$ and μ_u be the conditional measure of μ conditioned by the set $\{\omega : X_0(\omega) = u\}$. Let P_u be the product measure of μ_u and ν_u . Define $\tau(\omega, v)$, $T_j(\omega, \pi)$ and $\ell_j(\omega, \pi)$ by

$$\tau(\omega, v) = \inf\{t \geq 0 : X_t(\omega) = v\}, \quad T_j(\omega, \pi) = \max_{i \leq j} \tau(\omega, \sigma_i(\pi)) \quad \text{and} \quad \ell_j(\omega, \pi) = X_{T_j(\omega, \pi)}(\omega),$$

respectively. Then obviously, $T_{j-1}(\omega, \pi) < T_j(\omega, \pi)$ holds, iff $\ell_{j-1}(\omega, \pi) \neq \ell_j(\omega, \pi)$. Therefore we have that

$$\begin{aligned}
P_u(\ell_{j-1}(\omega, \pi) \neq \ell_j(\omega, \pi)) &= P_u(T_{j-1}(\omega, \pi) < T_j(\omega, \pi)) \\
&= P_u(\tau(\omega, \sigma_i(\pi)) < \tau(\omega, \sigma_j(\pi)), \ 2 \leq i < j) \\
&= \int_{\Omega} \nu_u(\{\pi : \tau(\omega, \sigma_i(\pi)) < \tau(\omega, \sigma_j(\pi)), \ 2 \leq i < j\}) d\mu_u(\omega) \\
&= \int_{\Omega} \frac{(n-1)!}{(j-1)!(n-j)!} \times \frac{(j-2)!(n-j)!}{(n-1)!} d\mu_u(\omega) = \frac{1}{j-1}
\end{aligned}$$

by Fubini's theorem.

Since $T_n(\omega, \pi) = n(\omega)$ holds for any $\pi \in S_n$, we see that

$$\begin{aligned}
&C_\mu(G, K, v) \\
&= E_{P_u} \left[\sum_{s=0}^{T_n(\omega, \pi)-1} k_{X_s(\omega)X_{s+1}(\omega)} \right] \\
&= \sum_{j=2}^n E_{P_u} \left[\sum_{s=0}^{T_j(\omega, \pi)-1} k_{X_s(\omega)X_{s+1}(\omega)} - \sum_{s=0}^{T_{j-1}(\omega, \pi)-1} k_{X_s(\omega)X_{s+1}(\omega)} \right] \\
&= \sum_{j=2}^n E_{P_u} \left[\sum_{s=T_{j-1}(\omega, \pi)}^{T_j(\omega, \pi)-1} k_{X_s(\omega)X_{s+1}(\omega)} : \ell_{j-1}(\omega, \pi) \neq \ell_j(\omega, \pi) \right] \\
&= \sum_{j=2}^n \sum_{\xi \neq \eta \in V} E_{P_u} \left[\sum_{s=T_{j-1}(\omega, \pi)}^{T_j(\omega, \pi)-1} k_{X_s(\omega)X_{s+1}(\omega)} : \ell_{j-1}(\omega, \pi) = \xi, \ell_j(\omega, \pi) = \eta \right] \\
&= \sum_{j=2}^n \sum_{\xi \neq \eta \in V} H_\mu^{G, K}(\xi, \eta) P_u(\ell_{j-1}(\omega, \pi) = \xi, \ell_j(\omega, \pi) = \eta) \\
&\leq \max\{H_\mu^{G, K}(\xi, \eta) : \xi \neq \eta \in V\} \sum_{j=2}^n \sum_{\xi \neq \eta} P_u(\ell_{j-1}(\omega, \pi) = \xi, \ell_j(\omega, \pi) = \eta) \\
&= \max\{H_\mu^{G, K}(\xi, \eta) : \xi \neq \eta \in V\} \sum_{j=2}^n P_u(\ell_{j-1}(\omega, \pi) \neq \ell_j(\omega, \pi))
\end{aligned}$$

Thus we showed the right-hand side inequality of (4.1). The left-hand side inequality can be shown similarly. $\square$

Theorem 2 can be generalized further to obtain the following theorem. For a given $G = (V, E)$, cost matrix K , $\mu \in M^+(\Omega)$ and $V' \subseteq V$, we define the weighted cover time $C_\mu(G, V', K)$ with respect to V' by

$$C_\mu(G, V', K) = \max_{u \in V'} C_\mu(G, V', K, u),$$

where for $u \in V'$,

$$C_\mu(G, V', K, u) = E_\mu[k_{V'}(\omega) | X_0(\omega) = u].$$

Here $k_{V'}(\omega)$ is defined by

$$k_{V'}(\omega) = \sum_{i=0}^{n_{V'}(\omega)-1} k_{x_i(\omega)x_{i+1}(\omega)},$$

where

$$n_{V'}(\omega) = \inf \left\{ i \in \mathbf{N} \mid \{X_0(\omega), X_1(\omega), \dots, X_{i-1}(\omega)\} = V' \right\}.$$

Theorem 3. *Let $G = (V, E)$ be a graph and $V' \subseteq V$. Then for any $\mu \in \mathcal{M}^+(\Omega)$ and cost matrix K ,*

$$H_{n'-1} \min_{u \neq v \in V'} H_{\mu}^{G,K}(u, v) \leq C_{\mu}(G, V', K) \leq H_{n'-1} \max_{u \neq v \in V'} H_{\mu}^{G,K}(u, v),$$

where $n' = |V'|$.

Let $\nu \in \mathcal{M}^+(\Omega)$ be a Markov measure with respect to a transition matrix $Q = (q_{uv})_{u,v \in V}$. By the definition of $\mathcal{M}^+(\Omega)$, $q_{vv} > 0$ may hold for some $v \in V$. We define $\hat{Q} = (\hat{q}_{uv})_{u,v \in V}$ by

$$\hat{q}_{uv} = \begin{cases} 0 & \text{if } u = v, \\ q_{uv}(1 + q_{vv})(1 - q_{vv})^{-1} & \text{otherwise.} \end{cases}$$

Let $\hat{\nu}$ be a Markov measure with respect to $\hat{Q}$. Then

$$H_{\hat{\nu}}^{G,K}(u, v) \leq H_{\nu}^{G,K}(u, v) \quad \text{for any } u \neq v \in V$$

holds[13]. We thus have the following lemma.

Lemma 1. *For a given undirected graph $G = (V, E)$ and cost matrix K , let $\nu, \mu \in \mathcal{M}^+(\Omega)$ be two Markov measures with respect to transition matrices $A = (a_{uv})_{u,v \in V}$ and $B = (b_{uv})_{u,v \in V}$, respectively. If there is a set of real numbers $\{c(u) \in (0, 1] : u \in V\}$ such that $a_{uv}c(u) = b_{uv}$ for any $u \in V, v \in N(u)$, then*

$$H_{\nu}^{G,K}(u, v) \leq H_{\mu}^{G,K}(u, v) \quad \text{for any } u \neq v.$$

Hence

$$\left\{ \min_{w \in V} c(w) \right\} H_{\mu}^{G,K^0}(u, v) \leq H_{\nu}^{G,K^0}(u, v) \leq \left\{ \max_{w \in V} c(w) \right\} H_{\mu}^{G,K^0}(u, v)$$

hold for any $u \neq v \in V$.

We are now ready to introduce our main results.

Theorem 4. *Let $G = (V, E)$ be a graph. Then the following two statements hold for any cost matrix K :*

$$\begin{aligned}
\text{(a)} \quad \max_{u,v \in V} H_{\nu^{(\beta)}}^{G,K}(u,v) &\leq \begin{cases} 2\bar{k}_\beta n^\beta (3n-4) & \text{for } \beta \geq 1, \\ 2\bar{k}_\beta n^{2-\beta} (3n-4) & \text{for } 0 < \beta \leq 1, \\ \bar{k}_\beta n^{2-\beta} (n-1) & \text{for } \beta \leq 0. \end{cases} \\
\text{(b)} \quad C_{\nu^{(\beta)}}(G,K) &\leq \begin{cases} 2\bar{k}_\beta n^\beta (3n-4) H_{n-1} & \text{for } \beta \geq 1, \\ 2\bar{k}_\beta n^{2-\beta} (3n-4) H_{n-1} & \text{for } 0 < \beta \leq 1, \\ \bar{k}_\beta n^{2-\beta} (2n-3) & \text{for } \beta \leq 0. \end{cases}
\end{aligned}$$

Here $\bar{k}_\beta \equiv k(\hat{\nu}^{(\beta)}, K, G) = \frac{1}{n} \sum_{u \in V} \sum_{v \in V} \hat{p}_{uv}^{(\beta)} k_{uv}$ and $\hat{\nu}^{(\beta)}$ is a Markov measure with respect to a symmetrical transition matrix $\hat{P}^{(\beta)} = (\hat{p}_{uv}^{(\beta)})_{u,v \in V}$ defined by

$$\hat{p}_{uv}^{(\beta)} = \begin{cases} \exp[-\beta U(u,v)] & \text{if } v \in N(u), \\ 1 - \sum_{w \in N(u)} \hat{p}_{uw}^{(\beta)} & \text{if } u = v, \\ 0 & \text{otherwise,} \end{cases}$$

for $\beta \geq 1$ and

$$\hat{p}_{uv}^{(\beta)} = \begin{cases} n^{\beta-1} \exp[-\beta U(u,v)] & \text{if } v \in N(u), \\ 1 - \sum_{w \in N(u)} \hat{p}_{uw}^{(\beta)} & \text{if } u = v, \\ 0 & \text{otherwise,} \end{cases}$$

for $\beta \leq 1$.

Proof. By definition, $\hat{\nu}^{(\beta)}$'s stationary distribution $\hat{\pi}^{(\beta)} = (\hat{\pi}_v^{(\beta)})_{v \in V}$ is uniform, that is, $\hat{\pi}_v^{(\beta)} = 1/n$ for all $v \in V$ and $\beta \in \mathbf{R}$.

Assume first $\beta \geq 1$. By (2.4) and Lemma 1,

$$H_{\nu^{(\beta)}}^{G,K}(u,v) \leq H_{\hat{\nu}^{(\beta)}}^{G,K}(u,v) \leq \bar{k}_\beta n \max\{\deg(u), \deg(v)\}^\beta,$$

which implies that

$$H_{\nu^{(\beta)}}^{G,K}(u,v) \leq \bar{k} n^\beta \max\{\deg(u), \deg(v)\}. \quad (4.2)$$

We next assume $\beta \leq 1$. Again by (2.4) and Lemma 1,

$$H_{\nu^{(\beta)}}^{G,K}(u,v) \leq H_{\hat{\nu}^{(\beta)}}^{G,K}(u,v) \leq \bar{k} n^{2-\beta} \max\{\deg(u), \deg(v)\}^\beta \quad \text{for } v \in N(u).$$

Since

$$n^{2-\beta} \max\{\deg(u), \deg(v)\}^\beta \leq \begin{cases} n^{2-\beta} \max\{\deg(u), \deg(v)\} & \text{for } 0 < \beta \leq 1, \\ n^{2-\beta} & \text{for } \beta \leq 0, \end{cases}$$

together with (4.2), we get for $v \in N(u)$

$$H_{\nu^{(\beta)}}^{G,K}(u,v) \leq \begin{cases} \bar{k}_\beta n^\beta \max\{\deg(u), \deg(v)\} & \text{for } \beta \geq 1, \\ \bar{k}_\beta n^{2-\beta} \max\{\deg(u), \deg(v)\} & \text{for } 0 < \beta \leq 1, \\ \bar{k}_\beta n^{2-\beta} & \text{for } \beta \leq 0. \end{cases} \quad (4.3)$$

Now, we evaluate the weighted mean hitting time. For given $u, v \in V$ with $u \neq v$, we choose a shortest path $u = v_0, v_1, \dots, v_l = v$ satisfying

$$\left(N(v_i) \cup \{v_i\} \right) \cap \left(N(v_j) \cup \{v_j\} \right) = \emptyset \quad (4.4)$$

for $1 \leq i < i+2 < j \leq l$. The existence of such a path can be shown as follows. Suppose that for i and j with $j > i+2$, there exists a w in $N(v_i) \cap N(v_j)$, then we can take a shortcut

$$u = v_0, v_1, \dots, v_i, w, v_j, \dots, v_l = v,$$

whose length is less than l . Applying this procedure finitely many times, we get a path satisfying (4.4). For the path satisfying condition (4.4), we have

$$\sum_{i=0}^l \deg(v_i) \leq 3n - 4. \quad (4.5)$$

Since $H_\mu^{G,K}(x, y) \leq H_\mu^{G,K}(x, z) + H_\mu^{G,K}(z, y)$ for any $x, y, z \in V$ and $\mu \in \mathcal{M}^+(\Omega)$,

$$H_{\nu^{(\beta)}}^{G,K}(u, v) \leq \sum_{i=0}^{l-1} H_{\nu^{(\beta)}}^{G,K}(v_i, v_{i+1}) \quad (4.6)$$

holds. Together with (4.3), (4.5) and (4.6), we get

$$H_{\nu^{(\beta)}}^{G,K}(u, v) \leq \begin{cases} 2\bar{k}_\beta n^\beta (3n - 4) & \text{for } \beta \geq 1, \\ 2\bar{k}_\beta n^{2-\beta} (3n - 4) & \text{for } 0 < \beta \leq 1, \\ \bar{k}_\beta n^{2-\beta} (n - 1) & \text{for } \beta \leq 0, \end{cases}$$

which imply (a).

As for (b), the inequality for $\beta > 0$ holds by (a) and Theorem 2. For $\beta \leq 0$, since there is a path of length $2n - 3$ that visits all vertices,

$$H_{\nu^{(\beta)}}^{G,K}(u, v) \leq \bar{k}_\beta n^{2-\beta} (2n - 3)$$

by (4.3). Again by Theorem 2, we have inequality (b). $\square$

Recall that with respect to $P^{(0)}$, both of the mean hitting time and the cover time are $O(n^3)$. Theorem 4 generalizes this fact: With respect to $P^{(\beta)}$, both of them are $O(n^{3-\beta})$ if $\beta \leq 0$. A more important conclusion is that both of the mean hitting time and the cover time achieve the minimum values when $\beta = 1$. Since $\bar{k} = 1$ for K^0 , we have the following corollary.

Corollary 2. *Let $G = (V, E)$ be a graph. Then the following two statements hold for any $\beta \in \mathbb{R}$:*

$$(a) \quad \max_{u, v \in V} H_{\nu^{(\beta)}}^{G, K^0}(u, v) = \begin{cases} O(n^{1+\beta}) & \text{for } \beta \geq 1, \\ O(n^{3-\beta}) & \text{for } \beta \leq 1. \end{cases}$$

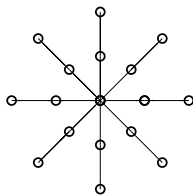


Fig. 3. A glitter star S_{17} .

$$(b) \quad C_{\nu^{(\beta)}}(G, K^0) = \begin{cases} O(n^{\beta+1} \log n) & \text{for } \beta \geq 1, \\ O(n^{3-\beta} \log n) & \text{for } 0 < \beta \leq 1, \\ O(n^{3-\beta}) & \text{for } \beta \leq 0. \end{cases}$$

We finally show that the cover time of a glitter star S_n introduced in Section 1 is $\Omega(n^{\beta+1} \log n)$, when $\beta \geq 1$.

Theorem 5. For any $n = 2m + 1$, $m \in \mathbf{N}$ with $m \geq 3$, and $\beta \geq 1$,

$$C_{\nu^{(\beta)}}(S_n, K^0) = \Theta(n^{1+\beta} \log n).$$

Proof. By Corollary 2, it is sufficient to show

$$C_{\nu^{(\beta)}}(S_n, K^0) = \Omega(n^{\beta+1} \log n).$$

Let V and V_O be the set of vertices of S_n and the set of pendant vertices of S_n . Hence $|V_O| = m$. Since $V_O \subseteq V$,

$$C_{\nu^{(\beta)}}(S_n, K^0) \geq C_{\nu^{(\beta)}}(S_n, V_O, K^0). \quad (4.7)$$

On the other hand, for any $u, v \in V_O$ with $u \neq v$, we can easily calculate that

$$H_{\nu^{(\beta)}}^{S_n, K^0}(u, v) = 2(n+1)(n^\beta + 1) + \frac{2n}{n-1}.$$

Hence

$$C_{\nu^{(\beta)}}(S_n, V_O, K^0) \geq \left\{ 2(n+1)(n^\beta + 1) + \frac{2n}{n-1} \right\} H_{m-1}$$

by Theorem 3. Together with (4.7), we have

$$C_{\nu^{(\beta)}}(S_n, K^0) = \Omega(n^{1+\beta} \log n).$$

□

By Theorem 1 and Corollary 2, $\nu^{(1)}$ is best possible with respect to the mean hitting time. As for the cover time, by Theorem 5, there is still a gap from the lower bound given by Theorem 1, as long as we adopt $\beta \geq 1$.

5 Conclusion

Random walks on finite graphs are rich source of attractive researches both in applied mathematics and in computer science. Despite of the lack of global topological information, both of the maximum mean hitting time and the cover time of a (conventional) random walk with respect to transition matrix $P = P^{(0)}$ can be bounded by $O(n^3)$. Hence a natural guess is that a better transition matrix is designable if more topological information is available. This paper showed that the guess is correct by investigating the maximum mean hitting time and the cover time with respect to $P^{(1)}$; the maximum mean hitting time of any graph with respect to $P^{(1)}$ is bounded by $O(n^2)$. Since the maximum mean hitting time of a path graph is shown to be $\Omega(n^2)$ for any transition matrix, $P^{(1)}$ is the best transition matrix as order, with respect to the mean hitting time. We also showed that the cover time of any graph with respect to $P^{(1)}$ is bounded by $\Theta(n^2 \log n)$.

There are many problems left unsolved. There is still a possible gap from a known lower bound $\Omega(n^2)$ on the cover time for a path graph. Looking for a matching bound seems to be challenging. We only investigated “universal” bounds on the mean hitting time and the cover time. Perhaps, there are some β values good for some classes of graphs.

The authors would like to thank an anonymous referee who contributes to Theorem 5 by pointing out a glitter star that achieves the matching lower bound, which graph makes the proof simpler than our original one.

References

1. D.J. Aldous, “On the time taken by random walks on finite groups to visit every state”, *Z. Wahrsch. verw. Gebiete* 62 361–393, 1983.
2. R. Aleliunas, R.M Karp, R.J. Lipton, L. Lovász, and C. Rackoff, “Random walks, universal traversal sequences, and the complexity of maze problems”, *Proc. 20th Ann. Symposium on Foundations of Computer Science*, 218–223, 1979.
3. G. Blom, L. Holst, and D. Sandell, “Problems and Snapshots from the World of Probability”, Springer-Verlag, New York, NY, 1994.
4. G. Brightwell and P. Winkler, “Maximum hitting time for random walks on graphs”, *J. Random Structures and Algorithms*, 3, 263–276, 1990.
5. A.Z. Broder and Karlin, “Bounds on covering times”, *In 29th Annual Symposium on Foundations of Computer science*, 479–487, 1988.
6. D. Coppersmith, P. Tetali, and P. Winkler, “Collisions among random walks on a graph”, *SIAM Journal on Discrete Mathematics*, 6, 3, 363–374, 1993.
7. U. Feige, “A tight upper bound on the cover time for random walks on graphs”, *J. Random Structures and Algorithms*, 6, 4, 433–438, 1995.
8. S. Ikeda, I. Kubo, N. Okumoto and M. Yamashita, “Fair circulation of a token,” *IEEE Trans. Parallel and Distributed Systems*, Vol.13, No.4, 367–372, 2002.
9. L. Isaacson and W. Madsen, “Markov chains: Theory and Application”, Wiley series in probability and mathematical statistics, New York, 1976.
10. A. Israeli and M. Jalfon, “Token management schemes and random walks yield self stabilizing mutual exclusion”, *Proc. of the 9th ACM Symposium on Principles of Distributed Computing*, 119–131, 1990.

11. P. Matthews, "Covering Problems for Markov Chain", *The Annals of Probability* Vol.16, No.3, 1215–1228, 1988.
12. R. Motowani and P. Raghavan, "Randomized Algorithms", Cambridge University Press, New York, 1995.
13. N. Okumoto, "A study on random walks of tokens on graphs", M.E.Thesis, Hiroshima Univ., Higashi-Hiroshima, Japan, 1996.
14. J.L. Palacios, "On a result of Aleiliunas et al. concerning random walk on graphs," *Probability in the Engineering and Informational Sciences*, 4, 489–492, 1990.

Analysis of a Simple Evolutionary Algorithm for Minimization in Euclidean Spaces

Jens Jägersküpper*

FB Informatik, LS 2, Univ. Dortmund, 44221 Dortmund, Germany
jj@ls2.cs.uni-dortmund.de

Abstract. Although evolutionary algorithms (EAs) are widely used in practical optimization, their theoretical analysis is still in its infancy. Up to now results on the (expected) runtime are limited to discrete search spaces, yet EAs are mostly applied to continuous optimization problems. So far results on the runtime of EAs for continuous search spaces rely on validation by experiments/simulations since merely a simplifying model of the respective stochastic process is investigated.

Here a first algorithmic analysis of the expected runtime of a simple, but fundamental EA for the search space $\mathbb{R}^n$ is presented. Namely, the so-called (1+1) Evolution Strategy ((1+1)ES) is investigated on unimodal functions that are monotone with respect to the distance between search point and optimum. A lower bound on the expected runtime is proven under the only assumption that isotropic distributions are used to generate the random mutation vectors. Consequently, this bound holds for any mutation adaptation mechanism. Finally, we prove that the commonly used “Gauss mutations” in combination with the so-called 1/5-rule for the mutation adaptation do achieve asymptotically optimal expected runtime.

Keywords: Evolutionary Algorithms, Black-Box Optimization, Continuous Search Space, Expected Runtime, Mutation Adaptation

1 Introduction

The optimization, here the minimization, of functions $f: S \rightarrow \mathbb{R}$ for some given search space S is one of the fundamental algorithmic problems. Discrete search spaces, e. g. $\{0, 1\}^n$, lead to combinatorial optimization problems like TSP, knapsack, or maximum matching. Mathematical optimization deals with continuous search spaces, usually $\mathbb{R}^n$. Here, problems are commonly defined by classes of functions, like polynomials of degree d , k -times differentiable functions, etc. Many problem-specific algorithms have been designed for each of these two scenarios. Since such algorithms are analyzed (in general), they can be compared and there is a theory on algorithms.

* supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the collaborative research center “Computational Intelligence” (SFB 531)

If not enough resources are on hand to design a problem-specific algorithm, however, robust algorithms like randomized search heuristics are often a good alternative. Especially, if the knowledge about the function f to be optimized is not sufficient, classical mathematical optimization algorithms like the steepest descent method or the conjugate gradient method cannot be applied. In the extreme, for instance if f is only given implicitly, knowledge about f can solely be gathered by consecutively evaluating f at selected points. This situation is commonly named “black-box optimization.” In this scenario, runtime is measured by the number of f -evaluations. Obviously, if we know nothing about f , a (reasonable) theoretical analysis of the runtime of some search heuristic like an evolutionary algorithm is impossible. Thus, to get insight into why such algorithms do often work quite well in practice, assumptions about the properties of f must be made, with respect to which the analysis is carried out.

This approach has been taken since the early 1990s for the discrete search space $\{0, 1\}^n$. Probably the first function that was analyzed is $\text{ONEMAX}(\mathbf{b}) := b_1 + \dots + b_n$, $\mathbf{b} = (b_1, \dots, b_n) \in \{0, 1\}^n$ (the name reflects that maximization was considered rather than minimization). The algorithm investigated was the so-called (1+1) Evolutionary Algorithm ((1+1) EA), which is in fact the discrete counterpart of the (1+1) ES investigated here. Both algorithms use a population consisting of only one search point, called an individual in the field of Evolutionary Computation. Thus, recombination is precluded, and mutation is the only “evolutionary force.” Within each beat of the evolution loop, the mutation of the current individual temporarily generates a second individual, and selection determines which one of both founds the next generation. An $O(n \log n)$ bound on the expected runtime of the (1+1) EA for ONEMAX (if mutation consists in flipping each bit of the individual independently with probability $1/n$) is proved in [9]. Retrospectively, this bound is easy to obtain; yet more sophisticated papers on the (1+1) EA have been published: In [2] linear functions are analyzed, in [14] quadratic polynomials, and in [13] monotone polynomials. Furthermore, [4] investigates the (1+1) EA for the maximum matching problem. Even the effect of recombination has been analyzed for the search space $\{0, 1\}^n$ [7,8], and the number of papers on algorithmic analyses is increasing.

The situation for continuous search spaces is different: The vast majority of results on EAs are empirical, i. e., based on experiments and simulations. In the few papers that focus on theoretical analyses, however, either (global) convergence is investigated or local changes from one generation to the next. In the former case, one must recall that EAs for continuous search spaces merely approximate an optimum rather than optimize the respective function. Convergence deals with the question of whether the algorithm reaches the ε -neighborhood of some (global) optimum in a finite number of steps or not (e. g. [11]). However, the order of the number of steps necessary remains open—in particular with respect to the dimension of the search space. On the other hand, results dealing with local changes in one step, for instance convergence rates, (generally) do not enable statements on the long-time behavior of EAs. Normally, the effect of mutation/recombination depends on the location of the respective individual(s) in the search space. Consequently, the changes from one generation to

the next one generally do not resemble the changes from the next generation to the second next. This is the reason why EAs for continuous search spaces apply so-called adaptation mechanisms, particularly mutation adaptation. The idea behind such adaptation mechanisms is to enable EAs to optimize as many types of functions as possible. Another idea behind mutation adaptation is that the mutative changes must in some way scale with the approximation quality. The rule of thumb reads: the closer the search approaches an optimum, the smaller the mutative changes. Unfortunately, (mutation) adaptation complicates the stochastic process an EA induces — and the analysis of the expected runtime.

The Scenario

As mentioned above, we will concentrate on the (1+1) ES which uses solely mutation because of a single-individual population. Let $\mathbf{c} \in \mathbb{R}^n$ denote this current individual. For a given initialization of $\mathbf{c}$, i. e., for a given starting point, the rough structure of the (1+1) ES is given by the following evolution loop:

1. Randomly choose the mutation vector $\mathbf{m} \in \mathbb{R}^n$.
2. Generate the mutant $\mathbf{x} \in \mathbb{R}^n$ by $\mathbf{x} := \mathbf{c} + \mathbf{m}$.
3. Using $f(\mathbf{c})$ and $f(\mathbf{x})$, the selection rule determines whether this mutant becomes the current individual ($\mathbf{c} := \mathbf{x}$) or is discarded ($\mathbf{c}$ unchanged).
4. If the stopping criterion is met then output $\mathbf{c}$ else goto 1.

A single execution of the loop is called a *step* of the (1+1) ES, and if “ $\mathbf{c} := \mathbf{x}$ ” is executed in a step, the mutation/mutant is said to be *accepted*, otherwise *rejected*. For a concrete instantiation of the (1+1) ES, the distribution of $\mathbf{m}$, the selection rule, and the stopping criterion must be specified. Although the stopping criterion is important in practice, we investigate the (1+1) ES as an infinite process. Let $T_f \in \mathbb{N}$ denote the number of steps the (1+1) ES needs to reach some fixed approximation quality when optimizing f . Then we are interested in $\mathbb{E}[T_f]$ and in $\mathbb{P}\{T_f \leq \tau\}$ for a given number of steps τ . By defining an appropriate randomized selection rule, simulated annealing can be realized for instance. However, we will investigate the commonly and originally used elitist selection where the mutant $\mathbf{x}$ becomes/replaces the current individual $\mathbf{c}$ if and only if $f(\mathbf{x}) \leq f(\mathbf{c})$. As this selection rule precludes worsenings, the (1+1) ES becomes a randomized hill-climber. If mutation adaptation is applied, obviously, the distribution of the mutation vector $\mathbf{m}$ is not fixed, but varies during the optimization process. Here we concentrate on mutation vectors that are isotropically distributed.

Definition 1. For $\mathbf{m} \in \mathbb{R}^n$, let $|\mathbf{m}|$ denote its length, i. e., its L_2 -norm, and $\widehat{\mathbf{m}} := \mathbf{m}/|\mathbf{m}|$ the normalized vector. The distribution of the random vector $\mathbf{m}$ is isotropic if $|\mathbf{m}|$ is independent of $\widehat{\mathbf{m}}$ and $\widehat{\mathbf{m}}$ is uniformly distributed upon the unit hyper-sphere $\{\mathbf{u} \in \mathbb{R}^n \mid |\mathbf{u}| = 1\}$.

Under these two assumptions (elitist selection and isotropically distributed mutation vectors) the lower bound on the runtime will be proved. That is, in each

step the mutation adaptation is free to choose an arbitrary isotropic distribution for $\mathbf{m}$. Consequently, the lower bound particularly holds for so-called “Gauss mutations” which are very common in practice (cf. Lemma 5 for the isotropy).

Definition 2. Let $\widetilde{\mathbf{m}} \in \mathbb{R}^n$ be $(N_1(0, 1), \dots, N_n(0, 1))$ -distributed (each component is independently standard normal distributed). A mutation is called Gauss mutation if the mutation vector’s distribution equals the one of $s \cdot \widetilde{\mathbf{m}}$, $0 < s \in \mathbb{R}$.

In particular, the upper bound on the runtime of the (1+1) ES will be proved with respect to Gauss mutations.

This scenario, (1+1) ES using elitist selection and Gauss mutations, has been introduced by Rechenberg, whose 1973 book *Evolutionstrategie* [10] is one starting point of evolutionary optimization. Rechenberg applied the (1+1) ES to optimize the shape of some workpiece. Furthermore, he presents some rough calculations on what length of the mutation vector maximizes the expected spacial gain in one step. These calculations are carried out with respect to two different kinds of functions. On the one hand, the so-called corridor model is considered, and on the other hand, the SPHERE function, where $\text{SPHERE}(\mathbf{x}) := x_1^2 + \dots + x_n^2$ for $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$. The calculations for the one-step behavior of the (1+1) ES on SPHERE have been improved by Beyer and can be found in his 2001 book *The Theory of Evolution Strategies* [1]. As a conclusion, Rechenberg states that the length of a Gauss mutation vector should be adapted such that the *success probability of a step*, the probability that the mutation in this step is accepted, is about $1/5$. This led to the notion of the *1/5-rule* for mutation adaptation: The (expected) length of the Gauss mutation vectors are scaled by adapting the factor s in Definition 2 as follows. For a certain number of steps (originally $\Theta(n)$ many), the relative frequency of successful steps is observed without changing s . Subsequent to each observation phase, the relative share of successful steps in the respective phase is evaluated; if it is smaller than $1/5$, s is divided by some fixed constant greater than 1, and otherwise, s is multiplied by some fixed (possibly different) constant greater than 1. The upper bound on the runtime will be proved with respect to this $1/5$ -rule.

Finally, the class of functions we consider contains all unimodal $f: \mathbb{R}^n \rightarrow \mathbb{R}$, $n \in \mathbb{N}$, such that for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and the respective optimum/minimum $\mathbf{o}_f \in \mathbb{R}^n$: $|\mathbf{x} - \mathbf{o}_f| < |\mathbf{y} - \mathbf{o}_f| \Rightarrow f(\mathbf{x}) < f(\mathbf{y})$. In other words, if an individual is closer to the optimum than some other, also its function value is better/smaller. We assume w.l.o.g. that the optimum $\mathbf{o}_f$ coincides with the origin, and thus, w.l.o.g. $|\mathbf{x}| < |\mathbf{y}| \Rightarrow f(\mathbf{x}) < f(\mathbf{y})$ for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Obviously, the L_2 -norm itself and for instance SPHERE (as well as all their translations) bear this property.

Results

As mentioned above, the one-step behavior of (1+1) ES on SPHERE has been investigated by Rechenberg and in great detail by Beyer. Unfortunately, at certain points within these calculations the limit $n \rightarrow \infty$ is taken without controlling the error terms; this is problematic in an algorithmic analysis, which exactly focuses on how the runtime depends on n . Thus, in Section 2 the n -dependence of the

one-step behavior of the (1+1) ES is investigated. The impact of the 1/5-rule on the convergence of the (1+1) ES is investigated in [12] and [5] for instance; yet the order of the number of steps is not tackled. Applying methods and concepts known from the field of randomized algorithms, the main results mentioned in the abstract are shown in Section 3. Finally, we close with some concluding remarks. Note that more detailed proofs can be found in [6].

Notions and Notations

As mentioned in Definition 1, $|\mathbf{x}|$ denotes the L_2 -norm of the vector $\mathbf{x} \in \mathbb{R}^n$, i. e., its length in Euclidean space, and $x_i \in \mathbb{R}$ its i th component. Furthermore, for instance, “ n -sphere” abbreviates “ n -dimensional sphere”.

Definition 3. A probability $p(n)$ is exponentially small in n if for a positive constant ε , $p(n) = \exp(-\Omega(n^\varepsilon))$. An event $A(n)$ happens with overwhelming probability (w. o. p.) with respect to n if $\mathbb{P}\{\neg A(n)\}$ is exponentially small in n .

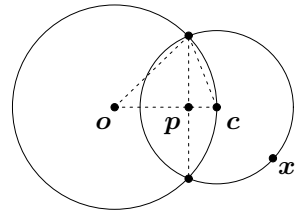
2 One-Step Behavior

As we are interested in how fast the “evolving” individual of the (1+1) ES approaches the optimum in the search space, the spatial gain towards the optimum in one step is the intermediate objective. Since the 1/5-rule for mutation adaptation is investigated, it is particularly interesting what length of the mutation vector results in the mutant being accepted with probability 1/5. Due to the independence of the random length of an isotropic mutation vector and its random direction (cf. Definition 1), we may assume that the length $\ell > 0$ of the mutation vector $\mathbf{m}$ is chosen according to $|\mathbf{m}|$ ’s distribution first; then the mutant is uniformly distributed upon the n -sphere with radius ℓ centered at the current search point $\mathbf{c}$.

The situation is depicted by the figure on the right. The left sphere $F := \{\mathbf{c}' \in \mathbb{R}^n \mid |\mathbf{c}'| = |\mathbf{c}|\}$ will be called the *fitness sphere* since the properties of f imply that all points inside (resp. outside) the fitness sphere are better (resp. worse) than the current search point $\mathbf{c}$. The potential mutants define the *mutation sphere* $M := \{\mathbf{x} \in \mathbb{R}^n \mid |\mathbf{x} - \mathbf{c}| = \ell\}$.

Let $I := F \cap M \subset \mathbb{R}^n$ denote the intersection of the two spheres. Obviously, if $\ell > 2|\mathbf{c}|$, I is empty, and if $\ell = 2|\mathbf{c}|$, I is a singleton, such that we concentrate on $\ell < 2|\mathbf{c}|$. It is easy to see that I forms an $(n-1)$ -sphere, and that the hyperplane $P \supset I$ is orthogonal to the line passing through $\mathbf{c}$ and $\mathbf{o}$. (Let $\mathbf{p} \in P$ denote the point where this line passes through P .) Hence, the mutation sphere’s part lying inside the fitness sphere forms a hyper-spherical cap $C \subset M - I$, the missing boundary of which is I . Basic geometry shows that the distance between $\mathbf{c}$ and P equals $g := |\mathbf{c}| - |\mathbf{p}| = \ell^2/(2|\mathbf{c}|)$ if $\ell \leq \sqrt{2}|\mathbf{c}|$.

Since the mutant $\mathbf{x}$ is uniformly distributed upon the mutation sphere M , for any (Lebesgue measurable) $S \subseteq M$, $\mathbb{P}\{\mathbf{x} \in S \mid |\mathbf{m}| = \ell\}$ equals the ratio



of the $(n-1)$ -volume of S to the one of M , inducing a probability measure. Consequently, I is of zero measure, and since $\mathbf{x}$ is better than $\mathbf{c}$ if $\mathbf{x} \in C$, and worse if $\mathbf{x} \in M - (C \cup I)$, the probability that the mutant is accepted equals the ratio of the hypersurface area of C to the one of M . Now, the interesting question is how this ratio depends on $|\mathbf{c}|$, ℓ , and, of course, n , the number of dimensions.

As the height of the mutation sphere's cap that is cut off by the fitness sphere equals $h := \ell - g = \ell - \ell^2/(2|\mathbf{c}|)$, the *relative height* of C , the ratio h/ℓ , equals $1 - \ell/(2|\mathbf{c}|)$. It can be shown (cf. [3, Appendix B] for instance) that

$$\frac{\text{hypersurface area of } C}{\text{hypersurface area of } M} = \frac{\Psi_{n-2}(\arccos(1 - h/\ell))}{\Psi_{n-2}(\pi)}$$

in n -space, $n \geq 3$, where $\Psi_k(\gamma) := \int_0^\gamma (\sin \beta)^k d\beta$. Note that $1 - h/\ell = (\ell - h)/\ell = g/\ell$. This formula may be directly used to estimate a step's success probability, yet it can also be utilized more generally: The ratio $\Psi_{n-2}(\arccos(g/\ell))/\Psi_{n-2}(\pi)$ not only equals the probability that the mutation hits C , but also the one of “the spatial gain of an isotropic mutation vector $\mathbf{m}$ parallel to some fixed direction (for instance $\overline{\mathbf{c}\mathbf{o}}$) is greater than g ,” under the condition $|\mathbf{m}| = \ell$. Therefore, let G denote the random variable given by the spatial gain of an isotropic mutation $\mathbf{m}$ parallel to a fixed direction under the condition $|\mathbf{m}| = \ell$. Then

$$\mathbf{P}\{G \leq g\} = 1 - \mathbf{P}\{G > g\} = 1 - \frac{\Psi_{n-2}(\arccos(g/\ell))}{\Psi_{n-2}(\pi)},$$

and hence, $F_n(x) := 1 - \Psi_{n-2}(\arccos(x/\ell))/\Psi_{n-2}(\pi)$ for $x \in [-\ell, \ell]$ is G 's probability distribution over $[-\ell, \ell]$ in n -space. Since Ψ_k is continuous, the probability density of G at $g \in [-\ell, \ell]$ equals $\frac{dF_n(x)}{dx}(g) = F'_n(g)$,

$$\begin{aligned} F'_n(x) &= \Psi_{n-2}(\pi)^{-1} \cdot (-1) \cdot \frac{d}{dx} \Psi_{n-2}(\arccos(x/\ell)) \\ &= \Psi_{n-2}(\pi)^{-1} \cdot (-1) \cdot \frac{d}{dx} \int_0^{\arccos(x/\ell)} (\sin \beta)^{n-2} d\beta \\ &= \Psi_{n-2}(\pi)^{-1} \cdot \ell^{-1} \cdot (1 - (g/\ell)^2)^{(n-3)/2} \end{aligned}$$

for $n \geq 4$. To make things clear, this is the density of the spatial gain of an isotropically distributed mutation vector $\mathbf{m}$ parallel to an arbitrarily fixed direction—independently of the function optimized—if $|\mathbf{m}|$ takes the value ℓ , not the one towards the optimum after selection.

With the help of this density function, we obtain an alternative formula for the success probability of a step, in which $\mathbf{c}$ is mutated using an isotropically distributed mutation vector $\mathbf{m}$ with $|\mathbf{m}| = \ell$ (y substitutes g/ℓ):

$$\begin{aligned} \mathbf{P}\{\mathbf{x} \text{ is accepted} \mid |\mathbf{m}| = \ell\} &= \mathbf{P}\{\mathbf{x} \in C \mid |\mathbf{m}| = \ell\} = \mathbf{P}\left\{G \geq \frac{\ell^2}{2|\mathbf{c}|}\right\} \\ &= \int_{\ell^2/(2|\mathbf{c}|)}^{\ell} F'_n(g) dg = \frac{\ell}{\Psi_{n-2}(\pi) \cdot \ell} \int_{\ell/(2|\mathbf{c}|)}^1 (1 - y^2)^{(n-3)/2} dy \end{aligned}$$

With respect to the $1/5$ -rule, which will be investigated for the upper bound on the expected runtime, we can now answer what length of the mutation vector results in a step of the $(1+1)$ ES having success probability $1/5$. Note that, obviously, this probability approaches $1/2$ as $\ell/|\mathbf{c}| \rightarrow 0$.

Lemma 1. *In the scenario considered, the mutant $\mathbf{c} + \mathbf{m} \in \mathbb{R}^n$ is accepted with a constant probability greater than 0 and smaller than $1/2$ if and only if $|\mathbf{m}|$ takes a value $\ell = \Theta(|\mathbf{c}|/\sqrt{n})$ in the respective step.*

Proof. The distance between $\mathbf{c}$ and P , the hyperplane containing the intersection of mutation sphere and fitness sphere, equals $\ell^2/(2|\mathbf{c}|) = \ell \cdot (\lambda/\sqrt{n})$ with $\lambda = \Theta(1)$, i.e., the relative height of the cap C equals $1 - \lambda/\sqrt{n}$. Using the formula derived above, we must show that $\int_{\lambda/\sqrt{n}}^1 (1 - y^2)^{(n-3)/2} dy$ as well as $\int_0^{\lambda/\sqrt{n}} (1 - y^2)^{(n-3)/2} dy$ are in $\Omega(\Psi_{n-2}(\pi))$, respectively. See [6]. $\square$

In other words, if the $1/5$ -rule was able to ensure a success probability of exactly $1/5$ in a step, the length of the mutation vector would be $\Theta(|\mathbf{c}|/\sqrt{n})$ in this step. Thus, the expected spatial gain towards the optimum in this situation is of particular interest and is estimated in the following.

Lemma 2. *If (in the scenario considered) $|\mathbf{m}| = \Theta(|\mathbf{c}|/\sqrt{n})$ in a step then the spatial gain towards the optimum is $\Omega(|\mathbf{m}|/\sqrt{n}) = \Omega(|\mathbf{c}|/n)$ with probability $\Omega(1)$ in this step, and thus, also the expected decrease in distance to the optimum in this step is $\Omega(|\mathbf{c}|/n)$.*

Proof. As in Lemma 1, the assumptions imply that C has height $\ell \cdot (1 - \lambda/\sqrt{n})$ for $\lambda = \Theta(1)$. One result of that Lemma is that the mutation hits C with probability $\Omega(1)$. Let $A \subset C$ denote the cap with height $\ell \cdot (1 - 2\lambda/\sqrt{n})$ such that its pole coincides with the one of C . Then each point in A is at least $\ell \cdot (1 - \lambda/\sqrt{n}) - \ell \cdot (1 - 2\lambda/\sqrt{n}) = \ell \cdot \lambda/\sqrt{n}$ distance units closer to the optimum than a point belonging to the boundary of C . Since the boundary of C equals the intersection of mutation sphere and fitness sphere, the distance to the optimum is decreased by at least $\ell \cdot \lambda/\sqrt{n} = \Theta(|\mathbf{c}|/n)$ distance units if the mutation hits A . This still happens with probability $\Omega(1)$ because the relative height of A equals $1 - \Theta(1/\sqrt{n})$ like the one of C . Since the properties of f in combination with the selection rule preclude a negative spatial gain, the expected decrease in distance to the optimum is $\Omega(|\mathbf{c}|/n)$. $\square$

Consequently, if the $1/5$ -rule is capable of adjusting the mutation vector's length such that the success probability is close to $1/5$, the distance to the optimum is expected to decrease by an $\Omega(1/n)$ -fraction. Note that, e.g., an $1/8$ -rule or an $1/3$ -rule would lead to the same asymptotic expected gain. Naturally, one might ask if an expected spatial gain $\omega(|\mathbf{c}|/n)$ is possible. We prove that in our scenario the expected spatial gain towards the optimum is $O(|\mathbf{c}|/n)$ for any adaptation of the length of an isotropic mutation vector. Hence, the $1/5$ -rule indeed tries to adjust the mutation vector's length to have optimal order $\Theta(|\mathbf{c}|/\sqrt{n})$ such that the expected spatial gain towards the optimum has maximum order $\Theta(|\mathbf{c}|/n)$.

Obviously, the spatial gain of a step equals 0 if the mutation is rejected, and is upper bounded by the mutation's spatial gain parallel to $\overline{\mathbf{c}}\mathbf{o}$, otherwise. A mutation is accepted (resp. rejected) if the spatial gain parallel to $\overline{\mathbf{c}}\mathbf{o}$ is greater (resp. smaller) than $\ell^2/(2|\mathbf{c}|)$. Using the probability density function obtained above, the expected spatial gain of a step, call it $\mathbb{E}[\text{gain}]$, is bounded above by

$$\begin{aligned} \int_{\ell^2/(2|\mathbf{c}|)}^{\ell} gF'_{n-2}(g) dg &= \frac{\ell}{\Psi_{n-2}(\pi)} \int_{\ell/(2|\mathbf{c}|)}^1 y \cdot (1-y^2)^{(n-3)/2} dy \\ &= \frac{\ell}{\Psi_{n-2}(\pi) \cdot (n-1)} \cdot \left(1 - \left(\frac{\ell}{2|\mathbf{c}|}\right)^2\right)^{(n-1)/2} \\ &< \frac{\ell}{\sqrt{2\pi}\sqrt{n-1}} \cdot \left(1 - \left(\frac{\ell}{2|\mathbf{c}|}\right)^2\right)^{(n-1)/2} \end{aligned}$$

because for $n \geq 4$, $\int y \cdot (1-y^2)^{(n-3)/2} dy = (1-y^2)^{(n-1)/2} / (-(n-1))$ and $\Psi_{n-2}(\pi) > \sqrt{2\pi}/\sqrt{n-1}$ (cf. [6]). Consequently, $\mathbb{E}[\text{gain}] = O(|\mathbf{m}|/\sqrt{n})$ independently of the *scaled distance from the optimum* $|\mathbf{c}|/|\mathbf{m}|$ (remember that $|\mathbf{m}| > 2|\mathbf{c}|$ results in the mutant being rejected since it lies outside the fitness sphere). Furthermore, the inequality enables the proof that $\mathbb{E}[\text{gain}] = O(|\mathbf{c}|/n)$ for any adaptation of the mutation vector's length.

Lemma 3. *In the scenario considered, the expected spatial gain towards the optimum in a step is $O(|\mathbf{c}|/n)$ —for any isotropic mutation.*

Proof. To prove this claim, we must show that $\mathbb{E}[\text{gain}]/|\mathbf{c}| = O(1/n)$ even if the mutation vector's length ℓ is chosen such that the expected spatial gain is maximized. Let $d := |\mathbf{c}|/\ell$ denote the scaled distance from the optimum. Applying the upper bound on the expected spatial gain from above yields

$$\mathbb{E}[\text{gain}]/|\mathbf{c}| < (2\pi(n-1))^{-1/2} \cdot \underbrace{(1/d) \cdot (1 - (2d)^{-2})^{(n-1)/2}}_{=: w_n(d)}.$$

Hence, an upper bound on $\mathbb{E}[\text{gain}]/|\mathbf{c}|$ can be derived by maximizing the function w_n . In fact, $w_n(d) = O(1/\sqrt{n})$ for $d > 0$ (cf. [6]), and thus,

$$\mathbb{E}[\text{gain}]/|\mathbf{c}| < w_n(d)/\sqrt{2\pi(n-1)} = O(1/\sqrt{n})/\sqrt{2\pi(n-1)} = O(1/n) \quad \square$$

3 Multi-step Behavior and Expected Runtime

Obviously, the multi-step behavior of the (1+1)ES crucially depends on the mutation adaptation used. For a lower bound on the expected runtime, however, optimal mutation adaptation may be assumed. Surprisingly, we need not prove explicitly what mutation adaptation is optimal. Furthermore, it is not evident what “runtime” means since f is merely approximated rather than optimized.

Due to the symmetry and scalability properties of f , linearity of expectation enables further statements if one knows (for an arbitrary starting point) the

expected number of steps to halve the distance from the optimum using optimal mutation adaptation. Namely, the expected runtime to reduce the distance from the optimum to a $1/k$ -fraction is lower bounded by $\lfloor \log_2 k \rfloor$ times the lower bound on the expected runtime to halve it. We apply the following modification of Wald's equation to prove the lower bound on the expected number of steps the (1+1) ES needs to halve the distance from the optimum (cf. [6] for the proof of this lemma).

Lemma 4. *Let $X_1, X_2, \dots$ denote random variables with bounded range and T the random variable defined by $T = \min\{t \mid X_1 + \dots + X_t \geq g\}$ for a given $g > 0$. If $\mathbb{E}[T]$ exists and $\mathbb{E}[X_i \mid T \geq i] \leq u$ for $i \in \mathbb{N}$ then $\mathbb{E}[T] \geq g/u$.*

Theorem 1. *In the scenario considered, for any adaptation of isotropic mutations the expected number of steps to halve the distance to the optimum is $\Omega(n)$.*

Proof. For $i \geq 1$, let X_i denote the random variable that corresponds to the spatial gain towards the optimum in the i th step. Furthermore, let $\mathbf{a} \in \mathbb{R}^n - \{\mathbf{o}\}$ denote the starting point and T the (random) number of steps until $|\mathbf{c}| \leq |\mathbf{a}|/2$ for the first time. As mentioned previously, worsenings are precluded such that $X_i \geq 0$ and in particular $|\mathbf{c}| \leq |\mathbf{a}|$ in each step. Consequently, $X_i \leq |\mathbf{c}| \leq |\mathbf{a}|$, and according to Lemma 3, $\mathbb{E}[X_i \mid T \geq i] = O(|\mathbf{c}|/n) = O(|\mathbf{a}|/n)$. Choosing $g := |\mathbf{a}|/2$ in Lemma 4, $\mathbb{E}[T] \geq (|\mathbf{a}|/2)/O(|\mathbf{a}|/n) = \Omega(n)$ if $\mathbb{E}[T]$ exists. If $\mathbb{E}[T]$ is not defined (due to improper adaptation), one may informally argue that “ $\mathbb{E}[T] = \infty = \Omega(n)$ ” since T is positive. $\square$

This lower bound on the expected runtime holds independently of the mutation adaptation applied since theoretically optimal adaptation is (implicitly) assumed. For the upper bound, we concretize the lower-bound scenario by choosing Gauss mutations and the 1/5-rule for mutation adaptation. The following properties of Gauss-mutations are useful (and proved in [6]).

Lemma 5. *A Gauss-mutation $\mathbf{m} \in \mathbb{R}^n$ is isotropically distributed, and moreover, $\ell_E := \mathbb{E}[|\mathbf{m}|]$ exists and $\mathbb{P}\{|\mathbf{m}| - \ell_E \geq \delta \cdot \ell_E\} \leq \delta^{-2}/(2n - 1)$.*

Let $\mathbf{m}_1, \dots, \mathbf{m}_n$ denote independent copies of $\mathbf{m}$. For any constant $\lambda < 1$ two positive constants a_λ, b_λ exist such that $\#\{i \mid a_\lambda \ell_E \leq |\mathbf{m}_i| \leq b_\lambda \ell_E\} \geq \lambda n$ w. o. p.

Furthermore, we investigate this instantiation of the 1/5-rule: The scaling factor s (cf. Definition 2) is adapted after every n th step: if less than $n/5$ of the respective last n steps were successful, s is halved, otherwise doubled. The asymptotic calculations we present, however, are valid for any 1/5-rule keeping s unchanged for $\Theta(n)$ steps, respectively, and using any two constants, each greater than 1, for the scaling of s .

The run of the (1+1) ES is partitioned into phases each of which lasts n steps such that $\mathbb{E}[|\mathbf{m}|]$ is constant in each phase. Let s_i denote the scaling factor used throughout the i th phase and ℓ_i the corresponding $\mathbb{E}[|\mathbf{m}|]$. A phase after which s is doubled is symbolized by “ $\times$ ” and one after which s is halved by “ $\div$ ”. Furthermore, let d_i denote the distance from the optimum at the beginning of the i th phase; hence, $d_i - d_{i+1}$ equals the spatial gain in/of the i th phase.

Lemma 6. *In the scenario considered for the 1/5-rule for Gauss mutations:*

1. *if $\ell_i = \Theta(d_i/\sqrt{n})$ then $d_{i+1} = d_i - \Omega(d_i)$ w. o. p.,*
2. *if s is doubled after the i th phase then $\ell_i = O(d_i/\sqrt{n})$ w. o. p.,*
3. *if s is halved after the i th phase then $\ell_{i+1} = \Omega(d_{i+1}/\sqrt{n})$ w. o. p.*

Proof. Assume that the total spatial gain of the i th phase is not $\Omega(d_i)$. Then the distance from the optimum is $\Theta(d_i)$ in each step of the phase (remember that the distance is non-increasing). Lemma 5 yields that w. o. p. $|\mathbf{m}| = \Theta(d_i/\sqrt{n})$ in $0.9n$ steps. According to Lemma 2, in each such step the spatial gain is $\Omega(d_i/n)$ with probability $\Omega(1)$. Hence, we expect $\Omega(n)$ steps each of which reduces the distance by $\Omega(d_i/n)$. By Chernoff bounds, the number of such steps is $\Omega(n)$ w. o. p. Consequently, our initial assumption contradictorily implies that the total spatial gain of the i th phase is $\Omega(d_i)$ w. o. p.

For the second claim, assume ℓ_i is not $O(d_i/\sqrt{n})$. Since the distance from the optimum is non-increasing, ℓ_i is not $O(|\mathbf{c}|/\sqrt{n})$ in each step of the i th phase. Lemma 5 yields that $|\mathbf{m}|$ is not $O(|\mathbf{c}|/\sqrt{n})$ in $0.9n$ steps w. o. p. According to Lemma 1, the success probability of each such step is $o(1)$. Hence, the expected number of unsuccessful steps is lower bounded by $0.9n - o(n)$. By Chernoff bounds, w. o. p. more than $0.8n$ steps are not successful. Thus, the assumption “ ℓ_i is not $O(d_i/\sqrt{n})$ ” contradictorily implies that s is halved w. o. p.

Assume ℓ_{i+1} is not $\Omega(d_{i+1}/\sqrt{n})$ for the third claim. Since $s_i = 2s_{i+1}$ also $\ell_i = 2\ell_{i+1}$. As the distance is non increasing, the assumption implies that ℓ_i is not $\Omega(|\mathbf{c}|/\sqrt{n})$ for each step of the $\div$ -phase. Following the proof of the second claim with symmetric arguments, w. o. p. more than $0.8n$ steps are successful—contradictorily implying that the i th phase is a $\times$ -phase w. o. p. $\square$

Now we can deal with sequences of phases in a run of the (1+1) ES.

Lemma 7. *If (in the scenario considered) the 1/5-rule for Gauss mutations causes a sequence $\div \times^k$ of phases, $k = \text{poly}(n)$, then w. o. p. the distance from the optimum is k times reduced by a constant fraction in the respective phases.*

Proof. Let the $\div$ -phase be the i th one. By Lemma 6, $\ell_{i+1} = \Omega(d_{i+1}/\sqrt{n})$ w. o. p. Since the adaptation yields $\ell_{i+w} \geq \ell_{i+1}$, $1 \leq w \leq k$, and the distance is non-increasing, w. o. p. $\ell_{i+w} = \Omega(d_{i+w}/\sqrt{n})$ for $1 \leq w \leq k$. Lemma 6 also yields that w. o. p. $\ell_{i+w} = O(d_{i+w}/\sqrt{n})$ for $1 \leq w \leq k$. Consequently, w. o. p. $\ell_{i+w} = \Theta(d_{i+w}/\sqrt{n})$ for $1 \leq w \leq k$, and finally, again according to Lemma 6, in each of the k $\times$ -phases the distance is reduced by a constant fraction w. o. p. $\square$

Lemma 8. *If (in the scenario considered) the 1/5-rule for Gauss mutations causes a sequence $\times \div^k$ of phases, $k = \text{poly}(n)$, then w. o. p. the distance from the optimum is k times reduced by a constant fraction in the respective phases.*

Proof. Let the $\times$ -phase be the i th one. For $k = 1$, assume that the total spatial gain of the i th and the $(i+1)$ th phase is not $\Omega(d_i)$. According to Lemma 6, w. o. p. $\ell_i = O(d_i/\sqrt{n})$ and w. o. p. $\ell_{i+2} = \Omega(d_{i+2}/\sqrt{n})$. Hence, $\ell_i = \Theta(d_i/\sqrt{n})$

as well as $\ell_{i+1} = \Theta(d_{i+1}/\sqrt{n})$, and Lemma 6 contradictorily implies that in each of the two phases the distance is reduced by a constant fraction w. o. p. Consequently, w. o. p. these two phases yield $d_{i+2} = d_i - \Omega(d_i)$.

For $k \geq 2$, the adaptation yields $s_{i+w} = s_i 2^{2-w} = 4 s_i / 2^w$ for $1 \leq w \leq k$, and according to Lemma 6, for $2 \leq w \leq k$ w. o. p. $\ell_{i+w} = \Omega(d_{i+w}/\sqrt{n})$. If $d_{i+w} \leq d_i / 2^w$ then by a simple accounting argument after the $(i+w)$ th phase $d_{i+w+1} \leq d_{i+w} \leq d_i / 2^w \leq d_i / \lambda^{w+1}$ for a constant $\lambda \geq \sqrt{2}$ and we are done. Thus, assume $d_{i+w} > d_i / 2^w$. As $\ell_{i+w} = 4 \ell_i / 2^w$, in this case “w. o. p. $\ell_i = O(d_i / \sqrt{n})$ ” implies that w. o. p. $\ell_{i+w} = O(d_{i+w} / \sqrt{n})$. Since also $\ell_{i+w} = \Omega(d_{i+w} / \sqrt{n})$, Lemma 6 yields that the $(i+w)$ th phase reduces the distance by a constant fraction w. o. p.

Altogether, the first two phases yield w. o. p. $d_{i+2} = d_i - \Omega(d_i)$, and for $2 \leq w \leq k$, either the distance from the optimum is reduced by a constant fraction in the $(i+w)$ th phase w. o. p., or after this phase $d_{i+w+1} \leq d_i / \lambda^{w+1}$ for a constant $\lambda \geq \sqrt{2}$ even if there was no spatial gain in the $(j+w)$ th phase. $\square$

Finally, the three preceding lemmas together with Theorem 1 yield the bound on the expected runtime, the expected number of steps the (1+1)ES needs for a predefined reduction of the distance from the optimum $\mathbf{o}$ in the search space.

Theorem 2. *If (in the scenario considered) for the suboptimal initial search point $\mathbf{a} \in \mathbb{R}^n - \{\mathbf{o}\}$ and the initial scaling factor s_1 , $|\mathbf{a} - \mathbf{o}| / s_1 = \Theta(n)$ then the expected number of steps to obtain a search point $\mathbf{c}$ such that $|\mathbf{c} - \mathbf{o}| \leq 2^{-t} |\mathbf{a} - \mathbf{o}|$ for $t \in \text{poly}(n)$ is $\Theta(t \cdot n)$.*

Proof. Assume w. l. o. g. that the optimum $\mathbf{o}$ coincides with the origin. The lower bound $\Omega(t \cdot n)$ follows immediately from Theorem 1.

If the sequence of phases starts with $\times \div$ or with $\div \times$, the two preceding lemmas yield that the number phases until $\mathbb{E}[|\mathbf{c}|] < 2^{-(t+1)} |\mathbf{a}|$ is $O(t)$. If the sequence starts with $\times^k$ or with $\div^k$ for $k \geq 2$, we must show that in these phases the distance is w. o. p. reduced k times by a constant fraction. The assumptions on the starting values ensure that in the first phase $\ell_1 = \mathbb{E}[|\widetilde{\mathbf{m}}|] \cdot s_1 = \Theta(\sqrt{n}) \cdot s_1 = \Theta(d_1 / \sqrt{n})$ (cf. Definition 2 for $\widetilde{\mathbf{m}}$ and [6] for $\mathbb{E}[|\widetilde{\mathbf{m}}|] = \Theta(\sqrt{n})$). Therefore, the same argumentation as for $\div \times^k$ resp. $\times \div^k$ can be applied (without the preceding $\div$ -phase resp. $\times$ -phase).

Hence, the number of phases such that $\mathbb{E}[|\mathbf{c}|] < |\mathbf{a}| \cdot 2^{-t} / 2$ is bounded by $O(t)$. By Markov’s inequality, $\mathbb{P}\{|\mathbf{c}| \leq |\mathbf{a}| \cdot 2^{-t}\} \geq 1/2$ after these $O(t)$ phases. If this is not the case, after all $|\mathbf{c}| \leq |\mathbf{a}|$ such that again with probability at least $1/2$, $|\mathbf{c}| \leq |\mathbf{a}| \cdot 2^{-t}$ after another $O(t)$ phases. Repeating this argument, the expected number of phases is upper bounded by $\sum_{i \geq 1} 2^{-i} \cdot i \cdot O(t) = 2 \cdot O(t)$, and the expected number of steps is $O(t \cdot n)$. $\square$

For other starting conditions, the (expected) number of steps necessary to ensure the theorem’s assumptions must be estimated before the theorem can be applied—for instance by estimating the number of steps until the scaling factor is halved and doubled at least once, respectively. This is a rather simple task when using the results presented.

4 Conclusion

For the first time, the (expected) runtime of a simple, but fundamental evolutionary algorithm for optimization in $\mathbb{R}^n$ is rigorously analyzed—not a simplifying model of it. In particular, this analysis shows that, in the scenario considered, the well-known 1/5-rule for mutation adaptation indeed results in asymptotically optimal expected runtime. As the analysis covers a wide range of realizations of the 1/5-rule, it additionally yields an interesting byproduct: Fine tuning the parameters of the 1/5-rule actually does not affect the order of the expected runtime; we could even replace 1/5 by 1/8 or by 1/3, for instance. This may be interpreted as an indicator for the robustness often ascribed to evolutionary algorithms; yet it is proved for the scenario considered only.

Acknowledgments. Thanks for productive discussions and for pointing out flaws especially go to Ingo Wegener, Carsten Witt, and Stefan Droste.

References

1. Beyer, H.-G. (2001). *The Theory of Evolution Strategies*. Springer, Berlin.
2. Droste, S., Jansen, T., Wegener, I. (2002). On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276, pp. 51–82.
3. Ericson, T., Zinoviev, V. (2001). *Codes on Euclidian Spheres*. Elsevier, Amsterdam.
4. Giel, O., Wegener, I. (2003). Evolutionary algorithms and the maximum matching problem. *Proceedings of the 20th International Symposium on Theoretical Computer Science (STACS 2003)*, LNCS 2607, pp. 415–426.
5. Greenwood, G. W., Zhu, Q. J. (2001). Convergence in evolutionary programs with self-adaptation. *Evolutionary Computation*, 9(2), pp. 147–157.
6. Jägersküpper, J. (2002). Analysis of a simple evolutionary algorithm for the minimization in euclidian spaces. Tech. Rep. CI-140/02, Univ. Dortmund, SFB 531, <http://sfhci.uni-dortmund.de/home/English/Publications/Reference/>.
7. Jansen, T., Wegener, I. (2001). Real royal road functions—where crossover provably is essential. *Proceedings of the 3rd Genetic and Evolutionary Computation Conference (GECCO 2001)*, Morgan Kaufmann, San Francisco, pp. 375–382.
8. Jansen, T., Wegener, I. (2002). The analysis of evolutionary algorithms—A proof that crossover really can help. *Algorithmica*, 34, pp. 47–66.
9. Mühlenbein, H. (1992). How genetic algorithms really work: Mutation and hill-climbing. *Proceedings of the 2nd Parallel Problem Solving from Nature (PPSN II)*, North-Holland, Amsterdam, pp. 15–25.
10. Rechenberg, I. (1973). *Evolutionsstrategie*. Frommann-Holzboog, Stuttgart, Germany.
11. Rudolph, G. (1997). *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovač, Hamburg.
12. Rudolph, G. (2001). Self-adaptive mutations may lead to premature convergence. *IEEE Transactions on Evolutionary Computation*, 5(4), pp. 410–414.
13. Wegener, I. (2001). Theoretical aspects of evolutionary algorithms. *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP 2001)*, LNCS 2076, pp. 64–78.
14. Wegener, I., Witt, C. (2003). On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions. *Journal of Discrete Algorithms*, to appear.

Optimal Coding and Sampling of Triangulations

Dominique Poulalhon and Gilles Schaeffer

LIX – CNRS, École polytechnique, 91128 Palaiseau Cedex, France,
{Dominique.Poulalhon,Gilles.Schaeffer}@lix.polytechnique.fr,
<http://lix.polytechnique.fr/Labo/{Dominique.Poulalhon,Gilles.Schaeffer}>

Abstract. We present a bijection between the set of plane triangulations (aka. maximal planar graphs) and a simply defined subset of plane trees with two leaves per inner node. The construction takes advantage of the minimal realizer (or Schnyder tree decomposition) of a plane triangulation.

This yields a simple interpretation of the formula for the number of plane triangulations with n vertices. Moreover the construction is simple enough to induce a linear random sampling algorithm, and an explicit information theory optimal encoding.

1 Introduction

This paper addresses three problems on finite *triangulations*, or *maximal planar graphs*: coding, counting, and sampling. The results are obtained as consequences of a new bijection, between triangulations endowed with their minimal realizer and trees in the simple class of plane trees with two leaves per inner node.

Coding. The coding problem was first raised in algorithmic geometry: find an encoding of triangulated geometries which is as compact as possible. As demonstrated by previous work, a very effective “structure driven” approach consists in distinguishing the encoding of the combinatorial structure, – that is, the triangulation – from the geometry – that is, vertex coordinates (see [26] for a survey and [16] for an opposite “coordinate driven” approach). Three main properties of the combinatorial code are then desirable: *compactity*, that is minimization of the bit length of code words, *linear complexity* of the complete coding and decoding procedure, and *locality*, that is the possibility to navigate efficiently (and to code the coordinates by small increments).

For the fundamental class $\mathcal{T}_n$ of triangulations of a sphere with $2n$ triangles, several codes of linear complexity were proposed, with various bit length $\alpha n(1 + o(1))$: from $\alpha = 4$ in [6,11,18], to $\alpha = 3.67$ in [21,29], and recently $\alpha = 3.37$ bits in [7]. The information theory bound on α is $\alpha_0 = \frac{1}{n} \log |T_n| \sim \frac{256}{27} \approx 3.245$ (see below). In some sense the compactness problem was given an optimal solution for general recursive classes of planar maps by Lu *et al.* [19,22]. For a fixed class, say triangulations, this algorithm does not use the knowledge of α_0 , as expected for a generic algorithm, and instead relies on a cycle separator algorithm

and, at bottom levels of recursion, on an exponential optimal coding algorithm. This leads to an algorithm difficult to implement with low complexity constants. Moreover the implicit nature of the representation makes it unlikely that locality constraints can be dealt with in this framework: known methods to achieve locality require the code to be based on a spanning tree of the graph.

Counting. The exact enumeration problem for triangulations was solved by Tutte in the sixties [31]. The number of rooted triangulations with $2n$ triangles, $3n$ edges and $n + 2$ vertices is

$$T_n = \frac{2(4n-3)!}{n!(3n-1)!}. \quad (1)$$

(This formula gives the previous constant $\alpha_0 = \frac{256}{27}$.) More generally Tutte was interested in *planar maps*: embedded planar multigraphs considered up to homeomorphisms of the sphere. He obtained several elegant formulas akin to (1) for the number of planar maps with n edges and for several subclasses (bipartite maps, 2-connected maps, 4-regular maps). It later turned out that constraints of this kind lead systematically to explicit enumeration results for subclasses of maps (in the form of algebraic generating functions, see [5] and references therein). A natural question in this context is to find simple combinatorial proofs explaining these results, as opposed to the technical computational proofs *à la Tutte*. This was done in a very general setting for maps without restrictions on multiple edges and loops [9,27]. Two main ingredients are at the heart of this approach: dual breath-first search to go from maps to trees, and a closure operation for the inverse mapping. When loops are forbidden, the first ingredient is no longer suited, but it was shown that it can be replaced by bipolar orientations [24,28]. When multiple edges are forbidden as well, the situation appears completely different and neither of the previous methods directly apply.

It should be stressed that planar graphs have in general non-unique embeddings: a given planar graph may underlie many planar maps. This explains that, as opposed to the situation for maps, no exact formula is known for the number of planar graphs with n vertices (even the asymptotic growth factor is not known, see [7,23]). However according to Whitney's theorem, 3-connected planar graphs have an essentially unique embedding. In particular the class of triangulations is equivalent to the class of maximal planar graphs (a graph is maximal planar if no edge can be added without losing planarity).

Sampling. A perfect (resp. approximate) random sampling algorithm outputs a random triangulation chosen in $\mathcal{T}_n$ under the uniform distribution (resp. under an approximation thereof): the probability to output a specific rooted triangulation T with $2n$ vertices is (resp. is close to) $1/T_n$. Safe for an exponentially small fraction of them, triangulations have a trivial automorphism group [25], so that as far as polynomial parameters are concerned, the uniform distribution on rooted or unrooted triangulations are indistinguishable.

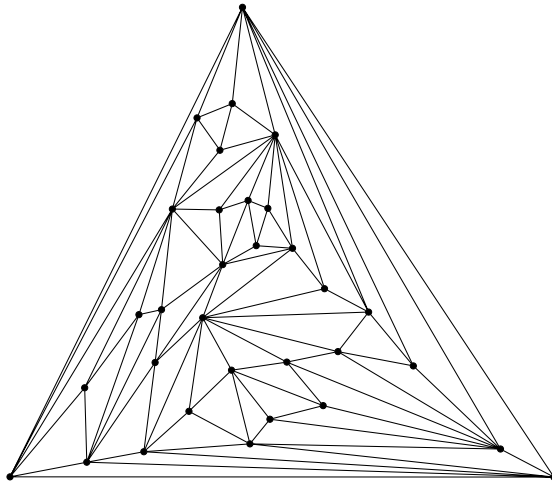


Fig. 1. A random triangulation with 60 triangles.

This question was first considered by physicists willing to test experimentally properties of two dimensional quantum gravity: it turns out that the proper discretization of a typical quantum universe is precisely obtained by sampling from the uniform distribution on rooted triangulations [4]. Several approximate sampling algorithms were thus developed by physicists for planar maps, including for triangulations [3]. Most of them are based on Markov chains, the mixing times of which are not known (see however [17] for a related study). A recursive perfect sampler was also developed for cubic maps, but has at least quadratic complexity [1]. More efficient and perfect samplers were recently developed for a dozen of classes of planar maps [5,29]. These algorithms are linear for triangular maps (with multiple edges allowed) but have average complexity $O(n^{5/3})$ for the class of triangulations.

Most random sampling algorithms are usually either based on Markov chains, or on enumerative properties. On the one hand, an algorithm of the first type perform a random walk on the set of configurations until it has (approximately) forgotten its start point. This is a very versatile method that requires little knowledge of the structures. It can even allow for perfect sampling in some restricted cases [32]. However in most cases it yields only approximate samplers of at least quadratic complexities. On the other hand, algorithms of the second type take advantage of exact counting results to construct directly a configuration from the uniform distribution [15]. As a result these perfect samplers often operate in linear time with little more than the amount of random bits required by information theory bounds to generate a configuration [2,13]. It is very desirable to obtain such an algorithm when the combinatorial class to be sampled displays simple enumerative properties, like Formula (1) for triangulations.

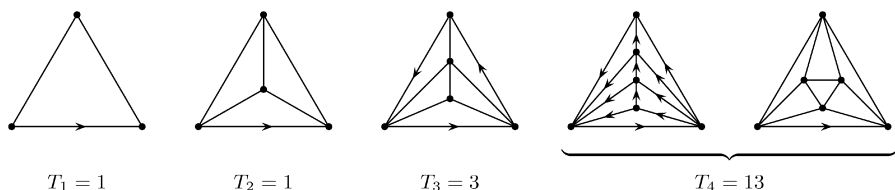


Fig. 2. The smallest triangulations with their inequivalent rootings.

New results. The central result of this paper is a one-to-one correspondence between the triangulations of $\mathcal{T}_n$ and the *balanced trees* of a new simple family $\mathcal{B}_n$ of plane trees. We give a linear *closure algorithm* that constructs a triangulation out of a balanced tree, and conversely, a linear *opening algorithm* that recovers a balanced tree as a special depth-first search spanning tree of a triangulation endowed with its minimal realizer. Realizers, or Schnyder tree decompositions, were introduced by Schnyder [30] to compute graph embeddings and have proved a fundamental tool in the study of planar graphs [8,10,14,20]. The role played in this paper by minimal realizers of triangulations is akin to the role of breadth-first search spanning trees in planar maps [9,27,29], and of minimal bipolar orientations in 2-connected maps [24,28], however the closure algorithm is very different from the closure used in the latter works. Our bijection allows us to address the three previously discussed problems.

From the coding point of view, our encoding in terms of trees preserves the entropy and satisfies linearity: each triangulation is encoded by one of the $\binom{4n}{n}$ bit strings of length $4n$ with sum of bits equal to n . The techniques of [18] to ensure locality apply to this $4n$ bit encoding. Optimal compacity can then be reached still within linear time, using for instance [7, Lemma 7].

From the exact enumerative point of view, the outcome of this work is a bijective derivation of Formula (1), giving it a simple interpretation in terms of trees. As far as we know, this is the first such bijective construction for a natural family of 3-connected planar graphs.

As far as random sampling is concerned, we obtain a linear time algorithm to sample random triangulations according to the (perfect) uniform distribution. In practice the speed we reach is about 100,000 vertices per second on a standard PC and triangulations with millions of vertices can be generated.

2 A One-to-One Correspondence

Let us first recall some definitions, illustrated by Figure 2.

Definition 1. A planar map is an embedding of a connected planar graph in the oriented sphere. It is rooted if one of its edges is distinguished and oriented; this determines a root edge, a root vertex (its origin) and a root face (to its right), which is usually chosen as infinite face for drawing in the plane.

A triangular map is a rooted planar map with all faces of degree 3. It is a triangulation if moreover it has no loop or multiple edge. A triangular map of size n has $2n$ triangular faces, $3n$ edges and $n + 2$ vertices; the three vertices incident to the root face are called external, as opposed to the $n - 1$ internal other ones. The set of triangulations of size n is denoted by $\mathcal{T}_n$.

2.1 From Trees to Triangulations

In view of Formula (1), it seems natural to ask for a bijection between triangulations and some kind of quaternary trees: indeed the number of such trees with n nodes is well known to be $\frac{(4n)!}{n!(3n+1)!}$. It proves however more interesting to consider the following less classical family of plane trees, illustrated by Fig. 3:

Definition 2. Let $\mathcal{B}_n$ be the set of plane trees with n nodes each carrying two leaves and rooted on one of these leaves.

It will prove useful to make a distinction between *nodes* (vertices of degree at least 2) and *leaves* (vertices of degree 1), and between *inner edges* (connecting two nodes) and *external edges* (connecting a node to a leaf).

The partial closure. We introduce here a partial closure operation that merges leaves to nodes in order to create triangular faces.

Let B be a tree of $\mathcal{B}_n$. The border of the infinite face consists of inner and external edges. An *admissible triple* is a sequence (e_1, e_2, e_3) of two successive inner edges followed by an external one in counterclockwise direction around the infinite face. An admissible triple is thus formed of three edges $e_1 = (v, v')$, $e_2 = (v', v'')$ and $e_3 = (v'', \ell)$ such that v, v' and v'' are nodes and ℓ is a leaf. The *local closure* of such an admissible triple (e_1, e_2, e_3) consists in merging the leaf ℓ with the node v so as to create a bounded face of degree 3. The external edge $e_3 = (v'', \ell)$ then becomes an internal edge (v'', v) . For instance the first three edges after the root around the infinite face of the tree of Figure 4(a) form an admissible triple, and the local closure of this triple produces the planar map of Figure 4(b). In turn, the first three edges of this map form a new admissible triple, and its local closure yields the map of Figure 4(c).

The *partial closure* $\tilde{B}$ of a tree B is the result of the greedy recursive application of local closure to all admissible triples available. The partial closure of the tree of Figure 4(a) is shown on Figure 4(d). At a given step of the construction,

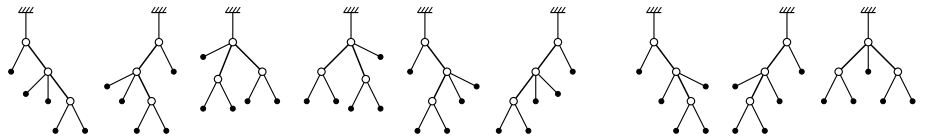


Fig. 3. The 9 elements of the set $\mathcal{B}_3$.

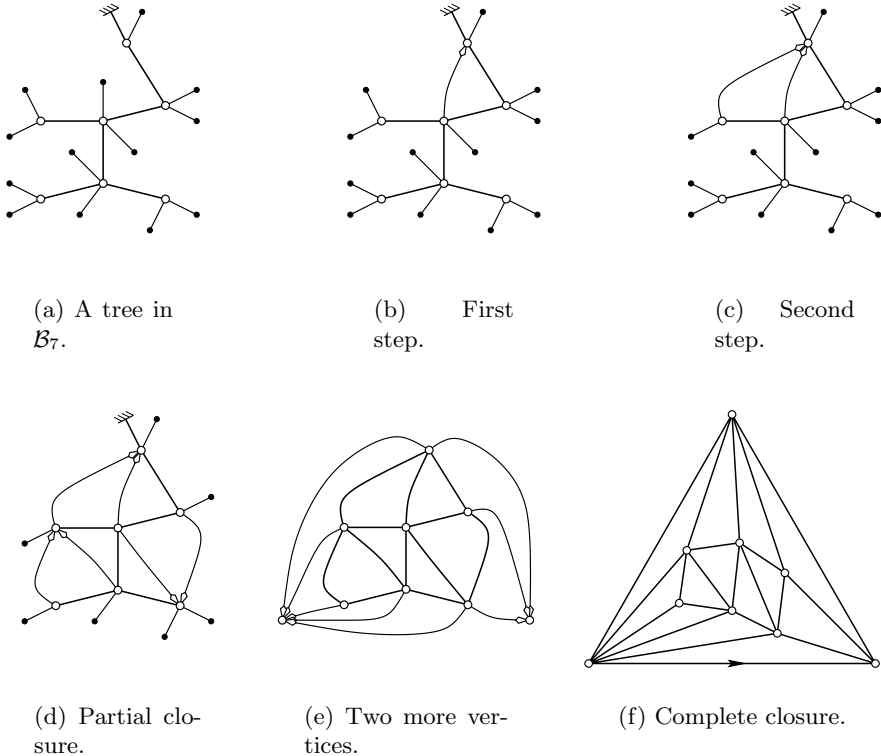
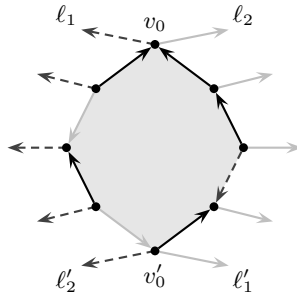


Fig. 4. Complete closure construction on an element of $\mathcal{B}_7$.

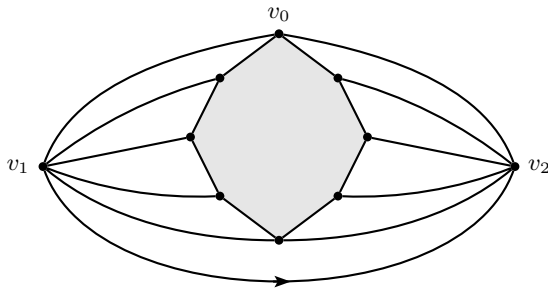
there are usually several admissible triples, but their local closures are independent so that the order in which they are performed is irrelevant and the final map $\tilde{B}$ is uniquely defined.

In the tree B there are two more external edges than sides of inner edges in the infinite face, and this property is preserved by local closures. When the partial closure ends, there is no more admissible triple but some leaves remain unmatched. Hence in the infinite face of $\tilde{B}$ no two inner edges can be consecutive: each inner edge lies between two external edges, as illustrated by Figures 4(d) and 5(a) (ignore orientations and colors for the time being). More precisely the external edges and sides of inner edges alternate except at two special nodes: these two nodes v_0 and v'_0 each carry two external edges with leaves ℓ_1, ℓ_2 and ℓ'_1, ℓ'_2 such that ℓ_1 (resp. ℓ'_1) follows ℓ_2 (resp. ℓ'_2) in the infinite face.

Observe that the partial closure of a tree is defined regardless of which of its leaves is the root. A tree B of $\mathcal{B}_n$ is said to be *balanced* if its root leaf is one of the two leaves ℓ_1 or ℓ'_1 of its partial closure $\tilde{B}$. Let $\mathcal{B}_n^*$ be the subset of balanced trees of $\mathcal{B}_n$. The fourth, sixth and eighth trees in Figure 3 are balanced. The following immediate property shall be useful later on.



(a) After partial closure.



(b) After complete closure.

Fig. 5. Generic situation.

Property 1. Let B be a balanced tree; then each local closure is performed between a leaf ℓ and a vertex v that is before ℓ in the left-to-right preorder on B .

The complete closure. Let B be a balanced tree of $\mathcal{B}_n^*$, and call v_0 and v'_0 the two special nodes of $\tilde{B}$ that carry the leaves ℓ_1, ℓ_2, ℓ'_1 and ℓ'_2 . The *complete closure* of B is obtained from its partial closure as follows (see Figures 4 and 5(b)):

1. merge ℓ_1, ℓ'_2 and all leaves in between at a new vertex v_1 ;
2. merge ℓ'_1, ℓ_2 and all leaves in between at a new vertex v_2 ;
3. add a root edge (v_1, v_2) .

The result of this complete closure is clearly a triangular map, which we denote $\bar{B}$. Apart from the orientation of the root, the complete closure is more generally well defined for any tree of $\mathcal{B}_n$ and does not depend on which of the $2n$ leaves is the root of the tree. Since $2n$ rooted trees correspond to a given unrooted one B (or n in the exceptional case where B has a global symmetry),



Fig. 6. Local property of a realizer.

in general $2n$ trees have the same image. This image is a triangular map with a marked (non oriented) edge (v_1, v_2) . However the use of the subset of balanced trees allows us to state a plain one-to-one correspondence rather than a “ $2n$ -to-2” one. We shall prove the following theorem in Section 3:

Theorem 1. *The complete closure is a one-to-one correspondence between the set $\mathcal{B}_n^*$ of balanced plane trees with n nodes with two leaves per node, and the set $\mathcal{T}_n$ of rooted triangulations of size n .*

Although the constructions are formally unrelated, the terminology we used here is reminiscent from [9,24,27], where bijections were proposed between some trees and planar maps with multiple edges.

2.2 From Triangulations to Trees

Minimal realizer. We shall use the following notion, due to Schnyder [30].

Definition 3. *Let T be a triangulation, with root edge (v_1, v_2) , and with v_0 its third external vertex. A realizer of T is a coloration of its internal edges in three colors c_0 , c_1 and c_2 satisfying the following conditions:*

- *for each $i \in \{0, 1, 2\}$, edges of color c_i form a spanning oriented tree of $T \setminus \{v_{i+1}, v_{i+2}\}$ rooted on v_i ; this induces an orientation of edges of color c_i toward v_i , such that each vertex has exactly one outgoing edge of color c_i ;*
- *around each internal vertex, outgoing edges of each color always appear in the cyclic order shown on Figure 6, and entering edges of color c_i appear between outgoing edges of the two other colors.*

From now on, this second condition is referred to as Schnyder condition.

Realizers of triangulations satisfy a number of nice properties [12,14,30], among which we shall use the following ones:

Proposition 1. – *Every triangulation has a realizer.*

- *The set of realizers of a triangulation can be endowed with an order for which there is a unique minimal (resp. maximal) element.*
- *The minimal realizer of a triangulation T is the unique realizer of T that contains no direct cycle.*
- *The minimal realizer of a triangulation can be computed in linear time.*

Depth-first search opening. Let T be a triangulation, endowed with its minimal realizer. Let (v_1, v_2) be its root edge, v_0 be the other external vertex. We construct a spanning tree of $T \setminus \{v_1, v_2\}$ using a right-to-left depth-first search traversal of T , modified to accept edges only if they are oriented toward the root:

1. delete (v_1, v_2) , and detach (v_0, v_1) and (v_0, v_2) from v_1 and v_2 to form two new leaves ℓ_1, ℓ_2 attached to v_0 ;
2. set $v \leftarrow v_0$ and $e \leftarrow (v_0, \ell_2)$, and mark v and e ;
3. as long as $e \neq (v_0, \ell_1)$, repeat:
 - a) $e' \leftarrow (v, u)$, the edge after e around v in clockwise direction;
 - b) *special orientation test*: if e' is oriented $v \rightarrow u$ and is not marked, mark e' and detach it from u to produce a leaf attached to v ;
 - c) otherwise, if u is marked and e' is not, set $e \leftarrow e'$;
 - d) otherwise, mark both u and e' if necessary and set $e \leftarrow e'$ and $v \leftarrow u$.

Step 3c prevents the opening algorithm from forming a cycle of marked edges and ensures that it eventually terminates. Let $S(T)$ be the visited tree, containing all marked edges. Without Step 3b, the opening algorithm would be a standard right-to-left depth-first search, and $S(T)$ would be the corresponding spanning tree. We shall prove the following proposition:

Proposition 2. *For any triangulation T , the tree $S(T)$ is a spanning tree of $T \setminus \{v_1, v_2\}$. Moreover it is the unique balanced tree with complete closure T .*

Because of the minimal orientation of T (without counterclockwise circuit), we shall see that the condition of Step 3c is in fact never satisfied. This line of the algorithm could thus as well be ignored: it was included only to make clear the fact that the algorithm terminates.

3 Proofs

3.1 The Closure Produces a Triangulation

The closure construction adds edges to a planar map and only creates triangular faces. It is thus clear that the resulting map is a triangular map with external vertices v_0, v_1 and v_2 , and with exactly two more vertices than B has nodes. Let us show that $\bar{B}$ is indeed a triangulation, *i.e.* has no multiple edge.

Let B be a balanced tree of $\mathcal{B}_n^*$. By definition the root leaf ℓ_1 of B is immediately followed around v_0 in clockwise direction by a second leaf ℓ_2 . Set ℓ_1 in color c_1 , ℓ_2 in color c_2 , and other edges incident to v_0 in color c_0 . Upon orienting all inner edges of B toward v_0 and all external edges toward their leaf, all vertices but v_0 have three outgoing edges. Since the tree B is acyclic, its orientation induces a unique coloration of edges satisfying the Schnyder condition (Figure 6) at all vertices but v_0 .

Lemma 1. *The orientation and coloration of edges still satisfy the Schnyder condition on each node but v_0 after the partial closure of B .*

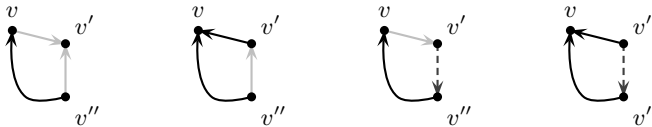


Fig. 7. The different cases of closure of a leaf.

Proof. This lemma is checked iteratively, by observing that each face created during the partial closure falls into one of the four types indicated on Figure 7 (up to cyclic permutation of colors). Indeed, consider an admissible triple (e_1, e_2, e_3) . Assuming that the external edge e_3 to be closed is of color c_0 , only two colors are possible for e_2 in view of the Schnyder condition at v'' . In each case again, only two colors are possible for e_1 . Finally in all four cases, the merging of ℓ into v does not contradict the Schnyder condition at v .

Property 2. If a (triangular) face of $\tilde{B}$ is oriented so that its sides form a circuit, then this circuit is necessarily oriented in the clockwise direction. More generally, each circuit in $\tilde{B}$ is created by the closure of a (last) leaf, the orientation of which imposes on the circuit to be clockwise.

Lemma 2. *After the complete closure, the Schnyder condition is satisfied at each internal vertex, and, apart from the three edges of the root face, each external vertex v_i is incident only to entering edges of color c_i .*

Proof. As illustrated by Figure 5(a), the Schnyder condition on nodes along the border of the partial closure implies that all external edges between ℓ_1 and ℓ'_2 (resp. ℓ_2 and ℓ'_1) are of color c_1 (resp. c_2). This is readily checked iteratively by a case analysis akin to the previous one.

The following proposition can be seen as an independant result on realizers.

Proposition 3. *A triangular map endowed with a colored 3-orientation satisfying the Schnyder condition on its inner vertices and the special condition on its three external vertices is in fact a triangulation endowed with a realizer.*

Proof. Let us first consider the color c_0 of the external vertex v_0 . By Schnyder condition each inner vertex has exactly one outgoing edge of color c_0 . In particular any cycle of edges of color c_0 is in fact a circuit. Moreover from each inner vertex originates a unique oriented path of color c_0 , ending either in v_0 or on a circuit of color c_0 .

Now consider two paths with distinct colors, say c_0 and c_1 . In view of the Schnyder coloring, a crossing between these two paths is necessarily of the type:



Hence two such paths can only cross once. Here crossing is taken in the (weak) sense having a vertex in common, even if this is just the origin of the path.

As a consequence monochrome circuits are vertex disjoint, and thus ordered by inclusion with respect to the external face. Consider a vertex v on an innermost circuit C . The Schnyder condition at v provides an edge e going out of v into the inner region delimited by C . Since this region contains no cycle the oriented path extending e has to cross C a second time, in contradiction with the previous discussion. This excludes monochrome circuits and proves that, for each $i = 0, 1, 2$, edges of color c_i form an oriented tree rooted at v_i . In particular multiple edges are excluded, and the coloring satisfies the definition of a realizer.

Combining Proposition 3 and Property 2, we obtain the following corollary that concludes the first part of the proof.

Corollary 1. *Upon keeping colors, the closure maps a balanced trees B of $\mathcal{B}_n^*$ on a triangulation with $n + 2$ vertices endowed with its minimal realizer.*

3.2 The Depth-First Search Opening Is Inverse to Closure

The following Lemmas 3-6 imply Proposition 2, and, together with Corollary 1, conclude the proof of Theorem 1.

Lemma 3. *The depth-first search opening visits all vertices of $T \setminus \{v_1, v_2\}$.*

Proof. Assume that the inner vertex v is not visited by the opening algorithm, that is to say, v does not belong to $S(T)$. By definition of realizers, there is a unique oriented path P of color c_0 starting in v and ending in v_0 . Let t be the last vertex on P that does not belong to $S(T)$, and $u \in S(T)$ the next vertex on P . Then the edge between t and u is oriented toward u and should have been included in the tree when u was visited.

Lemma 4. *The conditions of Step 3c are never satisfied.*

Proof. Consider the first time the conditions of Step 3c are satisfied. Up to that point an oriented tree S was constructed that contains v and u but not the edge (v, u) . Since the unmarked edge (v, u) was not considered by Step 3b, it is oriented from u toward v .

Let E be the set of edges that were already cut by Step 3b. Then S is the initial part of the right-to-left depth-first search tree of $T \setminus E$. In particular since the edge (u, v) is probed from v , the vertex u is an ancestor of v in the tree. But then the tree contains an oriented path from v to u , that forms a counterclockwise circuit with (u, v) . This contradicts the minimality of the orientation.

Lemma 5. *Edges that are cut by the opening algorithm lie on the left-hand side of the tree, as in Property 1. Hence the complete closure of $S(T)$ is T .*

Proof. As already observed, as the algorithm proceeds the tree which is constructed can be thought of as the right-to-left depth-first search tree of a submap of T . In particular when the algorithm probes an edge $e' = (v, u)$, this edge lies on the left-hand side of the tree, as in Property 1.

To check that the complete closure of $S(T)$ is T , it is sufficient to check that a cut edge would be properly replaced by local application of the closure algorithm. Since cut edges are bordered on one side by the infinite face and the final tree is a spanning tree, then the other face is bounded, that is triangular. Hence when $e' = (v, u)$ is cut, the vertex u lie two corners away from v along the infinite face in clockwise direction, as specified for admissible triples.

Lemma 6. *At most one spanning tree of $T \setminus \{v_1, v_2\}$ satisfies Property 1.*

Proof. Assume there are two such trees S and S' . Consider a left-to-right depth-first search traversal of both trees in parallel. Let $e = (v, u)$ be the first met edge that belongs to one of them – say S – and not to the other one. As the tree S' is also a spanning tree, there exists in S' a path from u to v_0 , the first edge of which, (u, t) , is oriented from u towards t . This orientation forbids to this edge to belong to the tree S ; it corresponds thus in that tree to the closure of a leaf of u . But since the edge (v, u) has been visited before (u, t) in the depth-first search traversal, this contradicts Property 1; there is therefore only one covering tree of T that satisfies this property.

4 Applications

4.1 An Explicit Optimal Code for Triangulations

As a first byproduct of Theorem 1, we obtain a code of triangulations in $\mathcal{T}_n$ by balanced trees in $\mathcal{B}_n$. Since a triangulation can be endowed with its minimal realizer in linear time (Proposition 1), the tree code can be obtained in linear time. Another fundamental feature of our code is that the tree code is a spanning tree of the original triangulation, making locality amenable to the techniques of [18]. Elements of $\mathcal{B}_n$ can themselves be coded by bit strings of length $4n - 2$ and weight $n - 1$ using a trivial variant of the usual prefix code for trees.

Theorem 2. *A tree B of $\mathcal{B}_n$ can be linearly represented by the word $s(B)$ that is obtained by writing 1 for “down” steps along inner edges, and 0 for leaves and for “up” steps along inner edges, during left-to-right depth-first search traversal.*

Hence a code for triangulations which is a subset of the set S of bit strings with length $4n - 2$ and weight $n - 1$. According to [7, Lem. 7] it can be given in linear time a representation as a bit string of length $\log |S| + o(n) \sim \log \binom{4n}{n} \sim \frac{256}{27} n$.

4.2 A Bijective Proof of Formula (1)

Proposition 4. *The set $\mathcal{B}_n$ has cardinality $\frac{2}{4n-2} \cdot \binom{4n-2}{n-1}$.*

Proof. As for classical prefix code of trees, the code words corresponding to trees of $\mathcal{B}_n$ can be easily characterized: they are the bit strings of length $4n - 2$ with weight $n - 1$ such that any proper prefix satisfies $3|u|_1 - |u|_0 > -2$. Now the number of such bit strings is readily obtained by the cycle lemma: in each cyclic class of words with length $4n - 2$ and weight $n - 1$, exactly 2 elements among $4n - 2$ are codes words (or 1 among $2n - 1$ for symmetric classes).

Now as seen in Section 2.1, any tree in $\mathcal{B}_n$ has two particular leaves among its $2n$ ones, and it is balanced if and only if one of these two is its root. Hence the ratio of balanced trees in $\mathcal{B}_n$ is $\frac{2}{2n}$. From Theorem 1 we obtain:

Corollary 2. *The number of triangulations with $2n$ triangles, $3n$ edges and $n+2$ vertices is $\frac{2}{2n} \cdot \frac{2}{4n-2} \cdot \binom{4n-2}{n-1}$, which is exactly Formula (1).*

4.3 Linear Time Perfect Random Sampling of Triangulations

The closure construction provides a sampling algorithm with linear complexity:

1. generate a random bit string of length $4n - 2$ and weight $n - 1$;
2. choose randomly one of its two cyclic shift that code an element of $\mathcal{B}_n$;
3. decode this word to construct the corresponding tree;
4. construct its partial closure by turning around the tree; using a stack, this can be done in at most two complete turns, hence in linear time;
5. complete the closure and choose a random orientation for the edge (v_1, v_2) .

Theorem 3. *This algorithm produces in linear time a random triangulation uniformly chosen in $\mathcal{T}_n$.*

Observe that Steps 1–3 correspond to a special case of the classical algorithm described *e.g.* in [2] for sampling trees.

Acknowledgments. We thank the authors of [7] for providing a draft of this work and for interesting discussions. In particular special thanks are due to Nicolas Bonichon for his invaluable knowledge of minimal realizers, and to Cyril Gavaille for pointing out Lemma 7 in [7].

References

1. M.E. Agishtein and A.A. Migdal. Geometry of a two-dimensional quantum gravity: numerical study. *Nucl. Phys. B*, 350:690–728, 1991.
2. L. Alonso, J.-L. Remy, and R. Schott. A linear-time algorithm for the generation of trees. *Algorithmica*, pages 162–183, 1997.

3. J. Ambjørn, P. Białas, Z. Burda, J. Jurkiewicz, and B. Petersson. Effective sampling of random surfaces by baby universe surgery. *Phys. Lett. B*, 325:337–346, 1994.
4. J. Ambjørn, B. Durhuus, and T. Jonsson. *Quantum geometry*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, Cambridge, 1997.
5. C. Banderier, P. Flajolet, G. Schaeffer, and M. Soria. Planar maps and Airy phenomena. In *ICALP*, pages 388–402, 2000.
6. N. Bonichon. A bijection between realizers of maximal plane graphs and pairs of non-crossing Dyck paths. In *FPSAC*, 2002.
7. N. Bonichon, C. Gavaille, and N. Hanusse. An information-theoretic upper bound of planar graphs using triangulations. In *STACS*, 2003.
8. N. Bonichon, B. Le Saëc, and M. Mosbah. Wagner’s theorem on realizers. In *ICALP*, pages 1043–1053, 2002.
9. M. Bousquet-Melou and G. Schaeffer. The degree distribution in bipartite planar maps: applications to the Ising model. 2002, [arXiv:math.CO/0211070](#).
10. E. Brehm. 3-orientations and Schnyder 3-tree-decompositions. Master’s thesis, FB Mathematik und Informatik, Freie Universität Berlin, 2000.
11. R. C.-N. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I Lu. Compact encodings of planar graphs via canonical orderings. In *ICALP*, pages 118–129, 1998.
12. H. de Fraysseix and P. Ossona de Mendez. Regular orientations, arboricity and augmentation. In *Graph Drawing*, 1995.
13. P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer. Random sampling from Boltzmann principles. In *ICALP*, pages 501–513, 2002.
14. S. Felsner. Convex drawings of planar graphs and the order dimension of 3-polytopes. *Order*, 18:19–37, 2001.
15. P. Flajolet, P. Zimmermann, and B. Van Cutsem. A calculus for random generation of labelled combinatorial structures. *Theoret. Comput. Sci.*, 132(2):1–35, 1994.
16. P.-M. Gandoin and O. Devillers. Progressive lossless compression of arbitrary simplicial complexes. *ACM Transactions on Graphics*, 21(3):372–379, 2002.
17. Z. Gao and J. Wang. Enumeration of rooted planar triangulations with respect to diagonal flips. *J. Combin. Theory Ser. A*, 88(2):276–296, 1999.
18. X. He, M.-Y. Kao, and H.-I Lu. Linear-time succinct encodings of planar graphs via canonical orderings. *SIAM J. on Discrete Mathematics*, 12(3):317–325, 1999.
19. X. He, M.-Y. Kao, and H.-I Lu. A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM J. Comput.*, 30(3):838–846, 2000.
20. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996. (also *FOCS’92*).
21. D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. In *CCCG*, 1999.
22. H.-I Lu. Linear-time compression of bounded-genus graphs into information-theoretically optimal number of bits. In *SODA*, pages 223–224, 2002.
23. D. Osthus, H.J. Prömel, and A. Taraz. On random planar graphs, the number of planar graphs and their triangulations. *J. Comb. Theory, Ser. B*, 2003. to appear.
24. D. Poulalhon and G. Schaeffer. A bijection for triangulations of a polygon with interior points and multiple edges. *Theoret. Comput. Sci.*, 2003. to appear.
25. L. B. Richmond and N. C. Wormald. Almost all maps are asymmetric. *J. Combin. Theory Ser. B*, 63(1):1–7, 1995.
26. J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.
27. G. Schaeffer. Bijective census and random generation of Eulerian planar maps with prescribed vertex degrees. *Electron. J. Combin.*, 4(1):# 20, 14 pp., 1997.

28. G. Schaeffer. *Conjugaison d'arbres et cartes combinatoires aléatoires*. PhD thesis, Université Bordeaux I, 1998.
29. G. Schaeffer. Random sampling of large planar maps and convex polyhedra. In *STOC*, pages 760–769, 1999.
30. W. Schnyder. Embedding planar graphs on the grid. In *SODA*, pages 138–148, 1990.
31. W. T. Tutte. A census of planar triangulations. *Canad. J. Math.*, 14:21–38, 1962.
32. D. B. Wilson. Annotated bibliography of perfectly random sampling with Markov chains. <http://dimacs.rutgers.edu/~dbwilson/exact>.

Generating Labeled Planar Graphs Uniformly at Random

Manuel Bodirsky¹, Clemens Gröpl², and Mihyun Kang¹

¹ Humboldt-Universität zu Berlin, Germany
`{bodirsky,kang}@informatik.hu-berlin.de`

² Freie Universität Berlin, Germany
`groepl@inf.fu-berlin.de`

Abstract. We present an expected polynomial time algorithm to generate a labeled planar graph uniformly at random. To generate the planar graphs, we derive recurrence formulas that count all such graphs with n vertices and m edges, based on a decomposition into 1-, 2-, and 3-connected components. For 3-connected graphs we apply a recent random generation algorithm by Schaeffer and a counting formula by Mullin and Schellenberg.

1 Introduction

A *planar graph* is a graph which can be embedded in the plane, as opposed to a *map*, which is an embedded graph. There is a rich literature on the enumerative combinatorics of maps, starting with Tutte's census papers, e.g. [20], and an efficient random generation algorithm was recently obtained by Schaeffer [16]. Much less is known about random planar graphs, although they recently attracted much attention [3, 12, 14, 6, 9, 5]. Even the expected number of edges for random planar graphs is not known (both in the labeled and in the unlabeled case), and the gap between known upper and lower bounds is still large [14, 9, 5]. There are also some results on the asymptotic number of labeled planar graphs [3, 14]. If we had an efficient algorithm to generate a planar graph uniformly at random, we could experimentally verify conjectures about properties of the random planar graph. We could also use it to evaluate the average-case running times of algorithms on planar graphs. Denise, Vasconcellos and Welsh [6] introduced a Markov chain having the uniform distribution on all labeled planar graphs as its stationary distribution. However, the mixing time is unknown and seems hard to analyze, and is perhaps not even polynomial. Moreover, their algorithm only approximates the uniform distribution.

We obtain the first expected polynomial time algorithm to generate a labeled planar graph uniformly at random.

¹ This research was supported by the Deutsche Forschungsgemeinschaft within the European graduate program 'Combinatorics, Geometry, and Computation' (No. GRK 588/2).

² Most of this work was done while the author was supported by DFG grant Pr 296/3.

Theorem 1. *A random planar graph with n vertices and m edges can be generated uniformly at random in expected time $O(n^{13/2})$ after a deterministic preprocessing of running time $O(n^7(\log n)^2(\log \log n))$. The memory requirement is $O(n^5 \log n)$ bits.*

We believe that the actual generation is much faster in practice, see Section 6. Our result uses known graph decomposition and counting techniques [21, 24] to reduce the counting and generation of labeled planar graphs to the counting and generation of 3-connected *rooted planar maps*, also called *c-nets*.

Usually a planar graph has many embeddings which are non-isomorphic as maps, but some graphs have a unique embedding. A classical theorem of Whitney (see e.g. [7]) asserts that 3-connected planar graphs are *rigid* in the sense that all embeddings in the sphere are combinatorially equivalent. As *rooting* destroys any further symmetries, c-nets are closely related to 3-connected *labeled* planar graphs. Moreover, the ‘degrees of freedom’ of the embedding of a planar graph are governed by its connectivity structure. We exploit this fact by composing a planar graph out of 1-, 2-, and 3-connected components.

The generation procedure first determines the number of components, and how many vertices and edges they shall contain. Each connected component is generated independently from the others, but having the chosen numbers of vertices and edges. To generate a connected component with given numbers of vertices and edges, we decide for a decomposition into 2-connected subgraphs and how the vertices and edges shall be distributed among its parts. So far this approach is similar to the one used in [4], where the goal was to generate random outerplanar graphs. In the planar case we need to go one step further.

Trakhtenbrot [19] showed that every 2-connected graph is uniquely composed of special graphs (called *networks*) of three kinds. Such networks can be combined in series, in parallel, or using a 3-connected graph as a template (see Theorem 2 below). Using this composition we can then employ known results about the counting and generation of 3-connected planar maps.

The concept of rooting plays an important role for the enumeration of planar maps. A *face-rooted* map is one with a distinguished edge which lies on the outer face and to which a direction is assigned. The rooting forces isomorphisms to map the outer face to the outer face, keep the root edge incident to the outer face, and preserve its direction. The enumeration of 3-connected face-rooted unlabeled maps with given numbers of vertices and faces was achieved by Mullin and Schellenberg [13]. We invoke their closed formulas in order to count 3-connected labeled planar graphs with given numbers of vertices and edges. For the generation of 3-connected labeled planar graphs with given numbers of vertices and edges we employ a recent algorithm by Schaeffer [17] running in expected polynomial time.

When we apply the various counting and generation subroutines along the stages of the connectivity decomposition, we must branch with the right probabilities. Instead of explicit (closed-form) counting formulas, which seem difficult to obtain, we derive recurrence formulas that can be evaluated in polynomial

time using dynamic programming. These recurrence formulas can be translated immediately into a generation procedure.

The paper is organized as follows: In the next section we give the graph theoretic background for the decomposition of planar graphs along their connectivity structure. This decomposition guides us when we derive the counting formulas for planar graphs in the following three sections. We analyze the running time and memory requirements of the corresponding generation procedure in Section 7. Some results from an implementation of the counting part are shown in Section 8. We conclude with a discussion of variations of the approach and how to derive a generation procedure for unlabeled planar graphs.

2 Decomposition by Connectivity

Let us recall and fix some terminology [7, 22, 23, 21]. A *graph* will be assumed un-oriented and *simple*, i.e., having no loops or multiple (also called *parallel*) edges; if multiple edges are allowed, the term *multigraph* will be used. We consider labeled graphs whose vertex sets are initial segments of $\mathbb{N}_0$.

Every connected graph can be decomposed into *blocks* by being split at cutvertices. Here a block is a maximal subgraph that is either 2-connected, or a pair of adjacent vertices, or an isolated vertex. The *block structure* of a graph G is a tree whose vertices are the cutvertices of G and the blocks (considered as vertices) of G , where adjacency is defined by containment. Conversely, we will *compose* connected graphs by identifying the vertex 0 of one part with an arbitrary vertex of the other. A formal definition of compose operations is given at the end of this section.

A *network* N is a multigraph with two distinguished vertices 0 and 1, called its *poles*, such that the multigraph N^* obtained from N by adding an edge between its poles is 2-connected. (The new edge is not considered a part of the network.) We can replace an edge uv of a network M with another network X_{uv} by identifying u and v with the poles 0 and 1 of X_{uv} , and iterate the process for all edges of M . Then the resulting graph G is said to have a *decomposition* with *core* M and *components* X_e , $e \in E(M)$.

Every network can be decomposed into (or composed out of) networks of three special types. A *chain* is a network consisting of 2 or more edges connected in *series* with the poles as its terminal vertices. A *bond* is a network consisting of 2 or more edges connected in *parallel*. A *pseudo-brick* is a network N with no edge between its poles such that N^* is 3-connected. (3-connected subgraphs are sometimes called bricks.) A network N is called an *h-network* (respectively, a *p-network*, or an *s-network*) if it has a decomposition whose core is a pseudo-brick (respectively, a bond, or a chain). Trakhtenbrot [19] formulated a canonical decomposition theorem for networks:

Theorem 2 (Trakhtenbrot). *Any network with at least 2 edges belongs to exactly one of the 3 classes: h-networks, p-networks, s-networks. An h-network has a unique decomposition and a p-network (respectively, an s-network) can be*

uniquely decomposed into components which are not themselves p-networks (s-networks), where uniqueness is up to orientation of the edges of the core, and also up to their order if the core is a bond.

A network is *simple* if it is a simple graph. Let $N(n, m)$ be the number of simple planar networks on n vertices and m edges. In view of Theorem 2 we introduce the functions $H(n, m)$, $P(n, m)$, and $S(n, m)$ that count the number of simple planar h-, p-, and s-networks on n vertices and m edges.

Let us define *compose operations* for the three stages $c = 0, 1, 2$ of the connectivity decomposition formally as follows. Assume that M and X are graphs on the vertex sets $[0 \dots k-1]$ and $[0 \dots i-1]$ and we want to compose them by identifying the vertices j of X with the vertices v_j of M , for $j = 0, \dots, c-1$, such that the resulting graph will have $n := k + i - c$ vertices. (No vertices are identified for $c = 0$.) Moreover, let S be a set of $i - c$ vertices from $[c \dots n-1]$ which are designated for the remaining part of X . Let M' be the graph obtained by mapping the vertices of M to the set $[0 \dots n-1] \setminus S$, retaining their relative order. Let X' be the graph obtained by mapping the vertices $[c \dots i-1]$ of X to the set S , retaining their relative order, and mapping j to the image of v_j in M' for $j = 0, \dots, c-1$. Then the result of the compose operation for the arguments $M, (v_0, \dots, v_{c-1}), X$, and S is the graph with vertex set $[0 \dots n-1]$ and edge set $E(M') \cup E(X')$.

We use $G^{(c)}(n, m)$ to denote the number of c -connected planar graphs with n vertices and m edges.

3 Planar Graphs

We show how to count and generate labeled planar graphs with a given number of vertices and edges in three steps. A first easy recurrence formula reduces the problem to the case of connected graphs. In the next section, we will use the block structure to reduce the problem to the 2-connected case. This may serve as an introduction to the method before we go into the more involved arguments of Section 5.

Let $F_k(n, m)$ denote the number of planar graphs with n vertices and m edges having k connected components. Clearly, $F_1(n, m) = G^{(1)}(n, m)$ and $G^{(0)}(n, m) = \sum_{k=1}^n F_k(n, m)$. Moreover,

$$F_k(n, m) = 0 \quad \text{for } m + k < n.$$

We count $F_k(n, m)$ by induction on k . Every graph with $k \geq 2$ connected components can be decomposed into the connected component containing the vertex 0 and a remaining part, using the inverse of the compose operation for $c = 0$ as defined in Section 2. If the split off part has i vertices, then there are $\binom{n-1}{i-1}$ ways to choose its vertex set, as the vertex 0 is always contained in it. The remaining part has $k-1$ connected components. We obtain the recurrence formula

$$F_k(n, m) = \sum_{i=1}^{n-1} \sum_{j=0}^m \binom{n-1}{i-1} G^{(1)}(i, j) F_{k-1}(n-i, m-j).$$

Thus it suffices to count connected graphs. But the counting recurrence also has an analogue for generation: Assume that we want to generate a planar graph G with n vertices and m edges uniformly at random. First, we choose $k \in [1 \dots n]$ with probability proportional to $F_k(n, m)$. Then we choose the number of vertices i of the component containing the vertex 0 and its number of edges j with a joint probability proportional to $\binom{n-1}{i-1} G^{(1)}(i, j) F_{k-1}(n-i, m-j)$. We also pick an $(i-1)$ -element subset $S' \subseteq [1 \dots n-1]$ uniformly at random and set $S := S' \cup \{0\}$. Then we compose G (as explained in Section 2) out of a random connected planar graph with parameters i and j , which is being mapped to the vertex set S , and a random planar graph with parameters $n-i$ and $m-j$ having $k-1$ connected components, which is generated in the same manner.

4 Connected Planar Graphs

In this section we reduce the counting and generation of connected labeled planar graphs to the 2-connected case. Let $M_d(n, m)$ denote the number of connected labeled planar graphs in which the vertex 0 is contained in d blocks. Here we will call them m_d -planars. An m_1 -planar is a planar graph in which 0 is not a cutvertex. Clearly, $G^{(1)}(n, m) = \sum_{d=1}^{n-1} M_d(n, m)$ and

$$M_d(n, m) = 0 \quad \text{for } n < d \text{ or } m < d.$$

In order to count m_d -planars by induction on d (for $d \geq 2$), we split off the largest connected subgraph containing the vertex 1 in which 0 is not a cutvertex. This is done by performing the inverse of the compose operation for $c = 1$ as defined in Section 2. If the split off m_1 -planar has i vertices, then there are $\binom{n-2}{i-2}$ possible choices for its vertex set, as the vertices 0 and 1 are always contained in it. The remaining part is an m_{d-1} -planar. Thus

$$M_d(n, m) = \sum_{i=2}^{n-d+1} \sum_{j=1}^{m-1} \binom{n-2}{i-2} M_1(i, j) M_{d-1}(n-i+1, m-j),$$

and this immediately translates into a generation procedure.

Next we consider m_1 -planars. The *root block* is the block containing the vertex 0. A recurrence formula for m_1 -planars arises from splitting off the subgraphs attached to the root block at its cutvertices one at a time. Thus we consider m_1 -planars such that the root block has b vertices and the c least labeled vertices in the root block are no cutvertices. Let us call them $m_{b,c}$ -planars and denote the number of $m_{b,c}$ -planars with n vertices and m edges by $M_{b,c}(n, m)$. Then $M_1(n, m) = \sum_{b=1}^n M_{b,1}(n, m)$. The initial cases are graphs without cutvertices. We have

$$M_{b,b}(n, m) = \begin{cases} G^{(2)}(n, m) & \text{for } b = n > 2 \\ 1 & \text{for } b = n \in \{1, 2\} \text{ and } m = n - 1 \\ 0 & \text{for } b \neq n. \end{cases}$$

To count $M_{b,c}$ using $M_{b,c+1}$, we split off the subgraph attached to the c -th least labeled vertex in the root block, if it is a cutvertex. This can be any connected planar graph. The remaining part is an $m_{b,c+1}$ -planar. If the split off subgraph has i vertices, then there are $\binom{n-1}{i-1}$ ways to choose them, as the vertex 0 of the subgraph will be replaced with the cutvertex. We obtain the recurrence formula

$$M_{b,c}(n, m) = \sum_{i=1}^{n-1} \sum_{j=0}^{m-1} \binom{n-1}{i-1} G^{(1)}(i, j) M_{b,c+1}(n-i+1, m-j).$$

Again, the generation procedure is straightforward.

5 2-Connected Planar Graphs

In this section we show how to count and generate 2-connected planar graphs. Note that every labeled 2-connected planar graph with n vertices and m edges is obtained from some simple planar network with n vertices and $m-1$ edges by adding an edge between the poles, then choosing $0 \leq x, y \leq n-1$, $x \neq y$, and exchanging the vertices 0 with x and 1 with y . Thus

$$G^{(2)}(n, m) = \begin{cases} \frac{\binom{n}{2}}{m} N(n, m-1) & \text{for } n \geq 3, m \geq 3 \\ 0 & \text{otherwise.} \end{cases}$$

Now we derive recurrence formulas for the number N of simple planar networks. Trakhtenbrot's decomposition theorem implies

$$N(n, m) = \begin{cases} P(n, m) + S(n, m) + H(n, m) & \text{for } n \geq 3, m \geq 2 \\ 0 & \text{otherwise.} \end{cases}$$

p-Networks. Let us call a p-network with a core consisting of k parallel edges a p_k -network, and let $P_k(n, m)$ be the number of p_k -networks having n vertices and m edges. Clearly, $P(n, m) = \sum_{k=2}^m P_k(n, m)$. In order to count p_k -networks by induction on k , we split off the component containing the vertex labeled 2 by performing the inverse of the compose operation for $c=2$ as defined in Section 2. Technically, it is convenient to consider the split off component as a p_1 -network. But note that according to the canonical decomposition, a p_1 -network is either an h- or an s-network. Thus

$$P_1(n, m) = \begin{cases} H(n, m) + S(n, m) & \text{for } n \geq 3, m \geq 2 \\ 0 & \text{otherwise.} \end{cases}$$

The remaining part is a p_{k-1} -network (even if $k=2$). For $k \geq 2$ we have

$$P_k(n, m) = 0 \quad \text{if } n \leq 2 \text{ or } m < k.$$

If a p-network with n vertices is split into a p_1 -network with i vertices and a p_{k-1} -network, there are $\binom{n-3}{i-3}$ ways how the vertex labels $[0 \dots n-1]$ can be

distributed among both sides, as the labels 0, 1, and 2 are fixed. We obtain the recurrence formula

$$P_k(n, m) = \sum_{i=3}^n \sum_{j=2}^{m-1} \binom{n-3}{i-3} P_1(i, j) P_{k-1}(n-i+2, m-j).$$

s-Networks. Let us call an s-network whose core is a path of k edges an s_k -network, and denote the number of s_k -networks which have n vertices and m edges by $S_k(n, m)$. Then $S(n, m) = \sum_{k=2}^m S_k(n, m)$. We use induction on k again, but for s_k -networks we split off the component containing the vertex labeled 0. Again it can be considered as an s_1 -network, and it is either an h- or a p-network, according to the canonical decomposition. Thus

$$S_1(n, m) = \begin{cases} H(n, m) + P(n, m) & \text{for } n \geq 3, m \geq 2 \\ 1 & \text{for } n = 2, m = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The remaining part is an s_{k-1} -network (even if $k = 2$). For $k \geq 2$ we have

$$S_k(n, m) = 0 \quad \text{if } n < k + 1 \text{ or } m < k.$$

Concerning the number of ways how the labels can be distributed among both parts, note that the labels 0 and 1 are fixed, hence the new 0-root for the remaining part can be one out of $n - 2$ vertices, and then the number of choices for the internal vertices of the split off s_1 -network is $\binom{n-3}{i-2}$. We obtain the recurrence formula

$$S_k(n, m) = (n-2) \sum_{i=2}^{n-1} \sum_{j=1}^{m-1} \binom{n-3}{i-2} S_1(i, j) S_{k-1}(n-i+1, m-j).$$

h-Networks. Let us call an h-network whose core is a pseudo-brick on k edges an h_k -network, and denote the number of h_k -networks with n vertices and m edges by $H_k(n, m)$. Then $H(n, m) = \sum_{k=5}^m H_k(n, m)$, as the smallest pseudo-brick has 5 edges. We can order the edges of the core lexicographically by the vertex numbers. A recurrence formula similar to the p- and s-network case arises from replacing the edges of the core with components one at a time and in lexicographic order. To give names to the intermediate stages, let $H_{k,\ell}(n, m)$ be the number of $h_{k,\ell}$ -networks with n vertices and m edges, where an $h_{k,\ell}$ -network is an h_k -network in which the components corresponding to the first ℓ edges of the core are simple edges. Thus $H_{m,m}(n, m)$ is the number of pseudo-bricks with n vertices and m edges, and $H_{k,k}(n, m) = 0$ for $k \neq m$. Applying the recurrence formula derived below for $\ell = k - 1$ down to 0, we can calculate $H_k(n, m) = H_{k,0}(n, m)$, and hence, $H(n, m)$. For the initial case, we have

$$H_{m,m}(n, m) = \frac{(n-2)!}{2} Q(n, m+1),$$

where $Q(n, m)$ denotes the number of c -nets, i.e., rooted 3-connected simple maps, with n vertices and m edges (see the next section): for we assign 0 to the root vertex, 1 to the other vertex of the root edge and the remaining labels to the remaining vertices, and neglect the orientation. To count $H_{k,\ell}$ using $H_{k,\ell+1}$, we split off the ℓ -th component of an $h_{k,\ell}$ -network, i.e., the component replacing the ℓ -th edge of the core. This can be a network of any of the three kinds. Thus

$$H_1(n, m) = \begin{cases} N(n, m) + N(n, m-1) & \text{for } n \geq 3, m \geq 2 \\ 1 & \text{for } n = 2, m = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The remaining part is an $h_{k,\ell+1}$ -network. If the ℓ -th component has i vertices, then there are $\binom{n-2}{i-2}$ ways to choose them, as the vertices 0 and 1 are merged with the endpoints of the ℓ -th edge of the core, respecting their relative order. We obtain the recurrence formula

$$H_{k,\ell}(n, m) = \sum_{i=2}^{n-2} \sum_{j=1}^{m-k+1} \binom{n-2}{i-2} H_1(i, j) H_{k,\ell+1}(n-i+2, m-j+1).$$

6 c -Nets

In the preceding sections, we have shown how to count and generate random planar graphs assuming that we can do so for c -nets, i.e., 3-connected simple rooted maps.

A *counting* formula for $Q(n, m)$ was derived by Mullin and Schellenberg in [13] in terms of given numbers of vertices and faces. Using Euler's formula, it asserts that

$$Q(n, m) = 0 \quad \text{for } n < 4 \quad \text{or } m < n + 2$$

and otherwise

$$\begin{aligned} Q(n, m) = & - \sum_{i=2}^n \sum_{j=n}^m (-1)^{i+j-n} \binom{i+j-n}{i} \binom{i}{2} \\ & \times \left[\binom{2m-2n+2}{n-i} \binom{2n-2}{m-j} - 4 \binom{2m-2n+1}{n-i-1} \binom{2n-3}{m-j-1} \right]. \end{aligned}$$

This concludes the counting task.

A *generation* algorithm for c -nets with given numbers of vertices and edges running in expected polynomial time algorithm is due to Schaeffer et al. [1, 2, 17, 15, 16]. Here we only outline the method. The c -net is obtained by extracting the 3-connected core from a 2-connected map. There is a linear time algorithm to generate 2-connected maps [15], and the extraction is linear as well [16]. If the parameters of the 2-connected map are tuned appropriately, chances are good that the resulting c -net will have the desired parameters. Otherwise the sample is rejected and the procedure restarts. A map with n vertices and m edges is said to

have an *imbalance* x which is defined by $n + 1 = m(\frac{1}{2} + x)$. To obtain a core with m edges and imbalance x , one should select a 2-connected map with imbalance $3x$ and $m/\alpha_0(3x)$ edges, where the *tuning ratio* is $\alpha_0(x) = \frac{(1-2x)(1+2x)}{3(1-2x/3)(1+2x/3)}$ [16, 2]. We have $\alpha_0(x) = \Omega(1/m)$ in the worst case. The expected number of iterations is $O(m^{2/3} + 1/p_\nu)$ for any given number of edges, where the probability p_ν that the core (whose size obeys a bimodal distribution) has around m edges is $p_\nu = \frac{16}{9}\alpha_0(3x)^2 = \Omega(1/m^2)$, and the $O(m^{2/3})$ term accounts for prescribing the exact number of edges. Prescribing also the number of vertices exactly (and not just up to a constant factor as in [16]) increases the running time by another factor $O(n^{1/2})$ (see [15, p. 140] and [17]). Thus a random c-net with m edges and imbalance x can be generated in expected time $O(m^{1+2+1/2+2}) = O(n^{11/2})$.

We conjecture that in fact a much faster generation should be possible based on two grounds: Most c-nets have an imbalance with $|x| \leq 1/2 - \varepsilon$, where $\varepsilon > 0$ is any constant. In this case the tuning ratio α_0 and hitting probability p_ν are bounded by constants and the expected running time reduces to $O(m^{1+2/3+1/2}) = O(n^{13/6})$. Moreover, if we are about to generate many planar graphs, we might store the rejected samples for future use, possibly resulting in a near-linear amortized running time at the expense of a larger (but still polynomial) memory requirement.

7 Running Time and Memory Requirements

In this section we establish a polynomial upper bound on the expected running time and the memory requirement of our algorithm.

A number of dynamic programming arrays has to be precalculated before the actual random generation starts. As an example, consider the recurrence formula for $H_{k,\ell}(n, m)$. The number of entries is $O(n^4)$ for all tables. All entries are bounded by the number of all planar graphs. Therefore the encoding length of each entry is $O(\log(n!38^n)) = O(n \log n)$ [14, 6] and the total space requirement is $O(n^5 \log n)$ bit. The calculation of each entry involves a summation over $O(n^2)$ terms. Using a fast multiplication algorithm, the precomputation time is $O(n^7(\log n)^2(\log \log n))$.

We assume that we can obtain random bits at unit cost. In order to prepare for branching with the right probabilities, we can easily calculate the necessary partial sums in a second pass over the dynamic programming arrays. We can then perform random decisions with the right probabilities in time linear in the encoding length, i.e., in $O(n \log n)$.

The total expected time spent in all calls to Schaeffer's c-net generation algorithm is bounded by $O(n^{13/2})$ (but we believe it is much faster in practice, see Section 6). Similarly, the random decisions for the connectivity decomposition require $O(n^2 \log n)$ time in total. An h -element subset of a k -element ground set can be chosen in $O(h \log k)$ time, hence the total time spent for random decisions for the label assignments during the composition is $O(n^2 \log n)$ as well. The compose operation itself is linear and requires at most $O(n^2)$ total time.

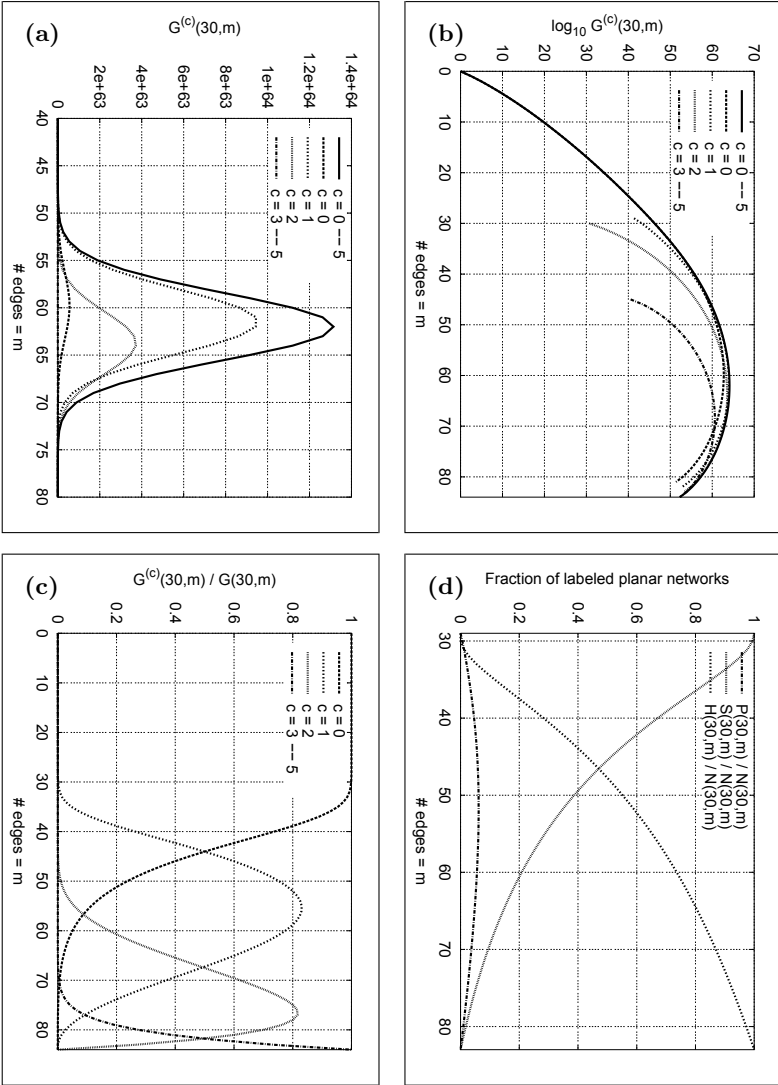


Fig. 1. Some counting results for labeled planar graphs on 30 vertices. The figures show the dependency on the number of edges m and the connectivity c . (a) Number of c -connected labeled planar graphs. (b) Similar in logarithmic scale. (c) Expected connectivity. (d) Expected type of a network (i.e., P , S , or H).

We see that the running time is dominated by $O(n^7(\log n)^2(\log \log n))$ for the preprocessing and $O(n^{13/2})$ (in expectation) for the random generation of c -nets. The space requirement is $O(n^5 \log n)$ bits due to the dynamic programming arrays.

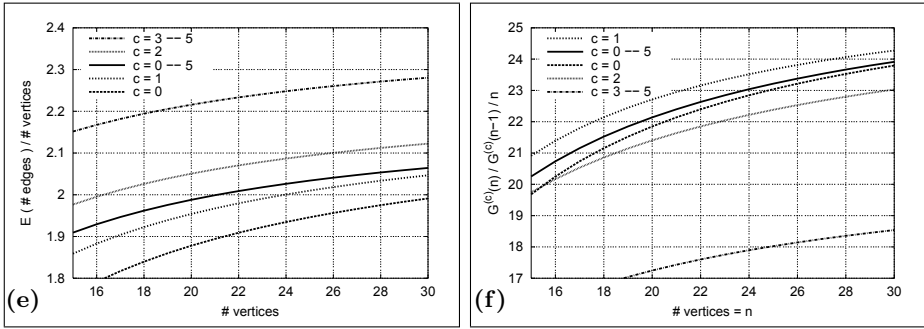


Fig. 2. (e) Edge density of a random labeled planar graph. The limit for general labeled planar graphs is known to be $\geq 13/6$ [9] and ≤ 2.54 [5]. (f) Growth rate of the number of labeled planar graphs.

8 Experimental Results

In this section we report on first computational results from an implementation of the counting formulas. The program was written in C++ using the GMP library for exact arithmetic [10]. A run for 30 vertices completed within one hour on a 1.3 GHz PC using 570 MB RAM. We also checked the recurrences and initial cases in Section 3-6 using an independent counting method. A list of all unlabeled planar graphs with up to 12 vertices was generated by a program of Köthnig [11]. From these the labeled planar graphs were enumerated by ‘brute force’. The unlabeled numbers, in turn, were confirmed by entries in Sloane’s encyclopedia of integer sequences [18] and [13].

Figures 1 and 2 are explained in the legend.

9 Conclusion

We have seen how to count and generate random planar graphs on a given number of vertices and edges using a recursive decomposition along the connectivity structure. Therefore a by-product of our result is that we can also generate *connected* and *2-connected* labeled planar graphs uniformly at random. Moreover it is easy to see that we can count and generate random planar *multigraphs* by only changing the initial values for planar networks as follows:

$$\begin{aligned} N(n, m) &= P(n, m) & \text{for } n = 2, m \geq 2 \\ P_k(n, m) &= 1 & \text{for } n = 2, m = k, k \geq 1. \end{aligned}$$

It seems difficult to simplify our counting recurrences to closed formulas. In this way one could eliminate the need for a preprocessing stage. Using generating functions Bender, Gao and Wormald obtained an *asymptotic* formula for the number of labeled 2-connected graphs [3].

To increase the efficiency of the algorithm one might want to apply a technique where the generated combinatorial objects only have approximately the correct size; this can then be turned into an exact generation procedure by rejection sampling. A general framework to tune and analyze such procedures is developed in [8, 2] and applied to structures derived by e.g. disjoint unions, products, sequences and sets. To deal with planar graphs it needs to be extended to the compose operation used in this paper.

References

1. C. Banderier, P. Flajolet, G. Schaeffer, and M. Soria. Planar maps and Airy phenomena. In *ICALP'00*, number 1853 in LNCS, pages 388–402, 2000.
2. C. Banderier, P. Flajolet, G. Schaeffer, and M. Soria. Random maps, coalescing saddles, singularity analysis, and Airy phenomena. *Random Structures and Algorithms*, 19:194–246, 2001.
3. A. Bender, Z. Gao, and N. Wormald. The number of labeled 2-connected planar graphs. Preprint, 2000.
4. M. Bodirsky and M. Kang. Generating random outerplanar graphs. Presented at ALICE 03, 2003. Journal version submitted.
5. N. Bonichon, C. Gavaille, and N. Hanusse. An information-theoretic upper bound of planar graphs using triangulation. In *In 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, 2003.
6. A. Denise, M. Vasconcellos, and D. Welsh. The random planar graph. *Congressus Numerantium*, 113:61–79, 1996.
7. R. Diestel. *Graph Theory*. Springer-Verlag, New York, 1997.
8. P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer. Random sampling from Boltzmann principles. In *ICALP '02*, LNCS, pages 501–513, 2002.
9. S. Gerke and C. McDiarmid. On the number of edges in random planar graphs. Submitted.
10. The GNU multiple precision arithmetic library, version 4.1.2.
<http://swox.com/gmp/>.
11. I. Köthnig. Personal communication. Humboldt-Universität zu Berlin, 2002.
12. C. McDiarmid, A. Steger, and D. J. Welsh. Random planar graphs. *Preprint*, 2001.
13. R. Mullin and P. Schellenberg. The enumeration of c-nets via quadrangulations. *Journal of Combinatorial Theory*, 4:259–276, 1968.
14. D. Osthus, H. J. Prömel, and A. Taraz. On random planar graphs, the number of planar graphs and their triangulations. *Combinatorial Theory, Series B*, to appear.
15. G. Schaeffer. *Conjugaison d'arbres et cartes combinatoires aléatoires*. PhD thesis, Université Bordeaux I, 1998.
16. G. Schaeffer. Random sampling of large planar maps and convex polyhedra. In *Proc. of the thirty-first annual ACM symposium on theory of computing (STOC'99)*, pages 760–769, Atlanta, Georgia, May 1999. ACM press.
17. G. Schaeffer. Personal communication, 2002.
18. N. J. A. Sloane. The on-line encyclopedia of integer sequences.
<http://www.research.att.com/~njas/sequences/index.html>, 2002.
19. B. A. Trakhtenbrot. Towards a theory of non-repeating contact schemes. *Trudi Mat. Inst. Akad. Nauk SSSR*, 51:226–269, 1958. [In Russian].
20. W. Tutte. A census of planar maps. *Canad. J. Math.*, 15:249–271, 1963.

21. T. Walsh. Counting labelled three-connected and homeomorphically irreducible two-connected graphs. *J. Combin. Theory*, 32:1–11, 1982.
22. T. Walsh. Counting nonisomorphic three-connected planar maps. *J. Combin. Theory*, 32:33–44, 1982.
23. T. Walsh. Counting unlabelled three-connected and homeomorphically irreducible two-connected graphs. *J. Combin. Theory*, 32:12–32, 1982.
24. T. Walsh and V. A. Liskovets. Ten steps to counting planar graphs. In *Eighteenth Southeastern International Conference on Combinatorics, Graph Theory, and Computing, Congr. Numer.*, volume 60, pages 269–277, 1987.

Online Load Balancing Made Simple: Greedy Strikes Back^{*}

Pilu Crescenzi¹, Giorgio Gambosi², Gaia Nicosia³, Paolo Penna^{4**}, and
Walter Unger⁵

¹ Dipartimento di Sistemi ed Informatica, Università di Firenze, via C. Lombroso
6/17, I-50134 Firenze, Italy (piluc@dsi.unifi.it)

² Dipartimento di Matematica, Università di Roma “Tor Vergata”, via della Ricerca
Scientifica, I-00133 Roma, Italy (gambosi@mat.uniroma2.it)

³ Dipartimento di Informatica e Automazione, Università degli studi “Roma Tre”,
via della Vasca Navale 79, I-00146 Roma, Italy (nicosia@dia.uniroma3.it)

⁴ Dipartimento di Informatica ed Applicazioni “R.M. Capocelli”, Università di
Salerno, via S. Allende 2, I-84081 Baronissi (SA), Italy (penna@dia.unisa.it),

⁵ RWTH Aachen, Ahornstrasse 55, 52056 Aachen, Germany
(quax@cs.rwth-aachen.de)

Abstract. We provide a new simpler approach to the on-line load balancing problem in the case of restricted assignment of temporary weighted tasks. The approach is very general and allows to derive on-line distributed algorithms whose competitive ratio is characterized by some combinatorial properties of the underlying graph representing the problem.

The effectiveness of our approach is shown by the hierarchical server model introduced by Bar-Noy *et al* '99. In this case, our method yields simpler and distributed algorithms whose competitive ratio is at least as good as the existing ones. Moreover, the resulting algorithms and their analysis turn out to be simpler. Finally, in all cases the algorithms are optimal up to a constant factor.

Some of our results are obtained via a combinatorial characterization of those graphs for which our technique yields $O(\sqrt{n})$ -competitive algorithms.

1 Introduction

Load balancing is a fundamental problem which has been extensively studied in the literature because of its many applications in resource allocation, processor scheduling, routing, network communication, and many others. The problem is to assign tasks to a set of n processors, where each task has an associated

^{*} A similar title is used in [13] for a facility location problem.

^{**} supported by the European Project IST-2001-33135, Critical Resource Sharing for Cooperation in Complex Systems (CRESCCO). Work partially done while at the Dipartimento di Matematica, Università di Roma “Tor Vergata” and while at the Institut für Theoretische Informatik, ETH Zentrum.

load vector and duration. Tasks must be assigned immediately to exactly one processor, thereby increasing the load of that processor by the amount specified by the corresponding coordinate of the load vector for the duration of the task. Usually, the goal is to minimize the maximum load over all processors.

The *on-line* load balancing problem has several natural applications. For instance, consider the case in which processors represent channels and tasks are communication requests which arrive one by one. When a request is assigned to a channel, a certain amount of its bandwidth is reserved for the duration of the communication. Since channels have limited bandwidth, the maximum load is an important measure here.

Several variants have been proposed depending on the structure of the load vectors, whether we allow *preemption* (i.e., to reassign tasks), whether tasks remain in the system “forever” or not (i.e., permanent vs. temporary tasks), whether the (maximum) duration of the tasks is known, and so on [1,3,4,5,6,14,16] (see also [2] for a survey).

In this paper, we study the on-line load balancing problem in the case of *temporary tasks* with *restricted assignment* and *no preemption*, that is:

- Tasks arrive one by one and their duration is unknown.
- Each task can be assigned to one processor among a subset depending on the type of the task.
- Once a task has been assigned to a processor, it cannot be reassigned to another one.
- Assigning a task to a processor increases the corresponding load by an amount equal to the weight of the task.

The problem asks to find an assignment of the tasks to the processors which minimizes the maximum load over all processors and over time.

Among others, this variant has a very important application in the context of wireless networks. In particular, consider the case in which we are given a set of base stations and each mobile user can only connect to a subset of them: those that are “close enough”. A user may unexpectedly appear in some spot and ask for a connection at a certain transmission rate (i.e., bandwidth). Also, the duration of this transmission is not specified (this is the typical case of a telephone call). Because of the application, it is desirable not to reassign users to other base stations (i.e., to avoid the handover), unless this becomes unavoidable because a user moves away from the transmission range of its current base (in the latter case, we can model this as a termination of the current request and a new request appearing in the new position).

As usual, we compare the cost of a solution computed by an on-line algorithm with the best off-line algorithm which minimizes the maximum load *knowing the entire sequence* of task arrivals and departures. Informally, an on-line algorithm is r -competitive if, at any instant, its maximum processor load is at most r times the optimal maximum processor load.

It is convenient to formulate our problem by means of a bipartite graph with vertices corresponding to processors and possible “task types”. More formally, let $P = \{p_1, \dots, p_n\}$ be a set of processors and let $\mathcal{T} \subseteq 2^P$ be a set of *task types*.

We represent the set of task types by means of an *associated bipartite graph* $G_{P,\mathcal{T}}(X_{\mathcal{T}} \cup P, E_{\mathcal{T}})$, where $X_{\mathcal{T}} = \{x_1, \dots, x_{|\mathcal{T}|}\}$ and

$$E_{\mathcal{T}} = \{(x_i, p_j) \mid p_j \text{ belongs to the } i\text{-th element of } \mathcal{T}\}.$$

A *task* t is a pair (x, w) , where $x \in X_{\mathcal{T}}$ and w is the positive integer weight of t . The set of processors to which t can be *assigned* is $P_t = \{p \mid (x, p) \in E_{\mathcal{T}}\}$, that is, the set of nodes of $G_{P,\mathcal{T}}(X_{\mathcal{T}} \cup P, E_{\mathcal{T}})$ that are adjacent to x . In our example of mobile users above, the type of a task corresponds to the user's position¹. In general, we consider two tasks which can be assigned to the same set of processors as belonging to the same type.

We follow the intuition that “nice” graphs may yield better competitive ratios, as one can see with the following examples:

General case. We do not assume anything regarding the possible task types.

So, the graph must contain all possible task types corresponding to any subset of processors, that is, $\mathcal{T} = 2^P$. Under this assumption, the best ratio achievable is $\Theta(\sqrt{n})$ [3,5] (see also [17]), while the greedy algorithm is exactly $\frac{3n^{2/3}}{2}(1 + o(1))$ -competitive [3], thus not optimal.

Identical machines. There is only one task type since a task can be assigned to any of the machines. Therefore, the graph is the complete bipartite graph $K_{1,n}$ and the competitive ratio of the problem is $\Theta(2 - 1/n)$. This ratio is achieved by the greedy algorithm [11,12], which is optimal [4].

Hierarchical servers. Processors are totally ordered and the type of a task corresponds to the “rightmost” processor that can execute that task. The set $\mathcal{T}$ contains one node per processor and the i -th node of $\mathcal{T}$ is adjacent to all processors j , with $1 \leq j \leq i$. There exists a 5-competitive algorithm and the greedy algorithm is at least $\Omega(\log n)$ -competitive.

Noticeably, one can consider the first two cases as the two extremes of our problem, because of both the (non-) optimality of the greedy algorithm and the (non-) constant competitive ratio of the optimal on-line algorithm. From this point of view, the latter problem is somewhat in between.

A related question is whether the greedy approach performs badly because of the fact that it must decide where to assign a task only based on *local* information (i.e., the current load of those processors that can execute that task). Indeed, the optimal algorithm in [5, Robin-Hood Algorithm] requires the computation of (an estimation of) the off-line optimum, which seems hard to compute in this local fashion. The algorithms in [7], too, require the computation of a quantity related to the optimum which depends on the current assignment of tasks of several types (see [7, Algorithm Continuous and Optimal Lemma]).

The idea of exploiting combinatorial properties of the graph $G_{P,\mathcal{T}}(X_{\mathcal{T}} \cup P, E_{\mathcal{T}})$ has been first used in [9]. In particular, the approach in [9] is based on the construction of a suitable subgraph which is used by the greedy algorithm (in

¹ Clearly, this is a simplification of the reality where other constraints must also be taken into account.

place of the original one). This subgraph is the union of a set of complete bipartite subgraphs (called clusters). So, this method can be seen as a modification of the greedy algorithm where the topology of the network is taken into account in order to limit the possible choices of the greedy algorithm². Therefore, the resulting algorithms only use “local information” as the greedy one does.

Several topologies have been considered in [9] for which the method improves over the greedy algorithm and matches the lower bound of the problem(s). In all such cases, however, the improvement is only by a constant factor, since the greedy algorithm was already $O(1)$ -competitive.

The main contribution of this paper (see Sect. 2) is a new approach to the problem based on the construction of a suitable subgraph to be used by the greedy algorithm. In this sense, our work is similar in spirit to [9]. However, the results here greatly improve over the method in that paper. Indeed, we show that:

- Some problems cannot be optimally solved with the solution in [9], while our approach does yield optimal competitive ratios.
- Our approach subsumes the one in [9] since the latter can be seen as a special case of the one presented here.

Also, our method yields the first example in which there is a significant improvement w.r.t. the greedy algorithm. This arises from the relevant case of hierarchical topologies, for which we attain a competitive ratio of 5 (4 for unweighted tasks³), while the greedy algorithm is at least $\Omega(\log n)$ in both cases. Table 1 summarizes the results obtained for these topologies. Even though, when $n \rightarrow \infty$, we achieve the same competitive ratio of [7], our algorithms and their analysis turn out to be much simpler. (Actually, for fixed n , our analysis yields strictly better ratios.)

We then turn our attention to the general case. In general, it might be desirable to automatically compute the best subgraph, as this would also give a simple way to test the goodness of our method w.r.t. a given graph. Unfortunately, one of our results is the NP-hardness of the problem of computing an optimal or even a c -approximate solution, for some constant $c > 1$.

In spite of this negative result, we demonstrate that a “sufficiently good” subgraph can be obtained very easily in many cases. We first provide a *sufficient condition* for obtaining $O(\sqrt{n})$ -competitive algorithms with our technique: the existence of a b -matching in (a suitable subgraph of) the graph $G_{\mathcal{P}, \mathcal{T}}$, for some constant b independent of n . Notice that, the lower bound for the general case is $\Omega(\sqrt{n})$, which applies to randomized algorithms [3] and to sequences of tasks of length polynomial in n [17]. By using this result, we obtain a $(2\sqrt{n} + 2)$ -

² This approach is somewhat counterintuitive since the algorithm improves when adding further restrictions to it (not to the adversary). This is reminiscent of the well-known Braess’ paradox [8,15], where the removal of some edges from a graph unexpectedly improves the latency of the flow at Nash equilibrium.

³ We denote the version of these problems in which all tasks have weight one by *unweighted*.

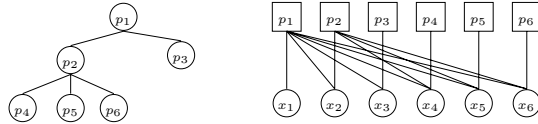


Fig. 1. An example of tree hierarchy (left) and the corresponding bipartite graph (right).

competitive distributed algorithm for the hierarchical server version in which processors are ordered as in a rooted tree (see the example in Fig. 1). A $\Omega(\sqrt{n})$ lower bound for centralized on-line algorithms also applies to this restriction [7], thus implying the optimality of our result. Additionally, we can achieve the same upper bound also when the ordering of the processors is given by *any* directed graph. This bound is only slightly worse than the $2\sqrt{n} + 1$ given by the Robin-Hood algorithm in [5]. All algorithms obtained with our technique can be

Table 1. Performance of our method in the case of hierarchical servers.

	Our Method	Previous Best	Greedy
Weighted	$5n/(n + 4)$ [Th. 5]	5 [7]	$\Omega(\log n)$ [folklore]
Unweighted	$4n/(n + 4)$ [Th. 5]	4 [7]	$\Omega(\log n)$ [folklore]

considered distributed in that they compute a task assignment only based on the current load of those processors that can be used for that task. This is a rather appealing feature since, in applications like the mobile networks above, introducing a global communication among bases for every new request to be processed may turn out to be unfeasible. Additionally, in several cases considered here (linear and tree hierarchical topologies), the construction of the subgraph is used solely for the analysis, while the actual on-line algorithm does not require any pre-computation (although the algorithm is different from the greedy one and it implements the subgraph used in the analysis).

Finally, we believe that our analysis is per se interesting since it translates the notion of adversary into a combinatorial property of the subgraph we are able to construct during an off-line preprocessing phase. As a by-product, the analysis of our algorithms is simpler and more intuitive.

Roadmap. We introduce some notation and definitions in Sect. 1.1. The technique and its analysis are presented in Sect. 2. The hardness results are given in Sect. 2.2. We give a first application to the hierarchical topologies in Sect. 3. The application to the general case is described in Sect. 4 where we provide sufficient conditions for $O(\sqrt{n})$ -competitiveness. These results are used in Sect. 4.1 where we obtain the results on generalized server hierarchies. Finally, in Sect. 5 we discuss some further features of our algorithms and present some open problems.

Due to lack of space, some of the proofs are only sketched or omitted in this version of the paper. The omitted proofs are contained in [10].

1.1 Preliminaries and Notation

An *instance* σ of the on-line load balancing problem with processors P and task types $\mathcal{T}$ is defined as a sequence of **new**($\cdot, \cdot$) and **del**($\cdot$) commands. In particular: (i) **new**(x, w) means that a new task of weight w and type $x \in \mathcal{T}$ is created; (ii) **del**(i) means that the task created by the i -th **new**($\cdot, \cdot$) command of the instance is deleted.

As already mentioned, we model the problem by means of a bipartite graph $G_{P, \mathcal{T}}(X_{\mathcal{T}} \cup P, E_{\mathcal{T}})$, where $\mathcal{T}$ depends on the problem version we are considering. For the sake of brevity, in the following we will always omit the subscripts ' $P, \mathcal{T}$ ' and ' $\mathcal{T}$ ' since the set of processors and the set of task types will be clear from the context. Given a graph $G(V, E)$, $\Gamma_G(v)$ denotes the *open* neighborhood of the node $v \in V$. So, a task of type x can be assigned to $\Gamma_G(x)$.

We will distinguish between the *unweighted* case, in which all tasks have weight 1, and the *weighted* case, in which the weights may vary from task to task. We also refer to (un-)weighted tasks to denote these variants.

Given an instance σ , a *configuration* is an assignment of the tasks of σ to the processors in P , such that each task t is assigned to a processor in P_t . Given a configuration C , we denote with $l_C(i)$ the *load* of processor p_i , that is, the sum of the weights of all tasks assigned to it. In the sequel, we will usually omit the configuration when it will be clear from the context. The load of C is defined as the maximum of all the processor loads and is denoted with $l(C)$. Given an instance $\sigma = \sigma_1 \cdots \sigma_n$ and given an on-line algorithm $\mathcal{A}$, let $C_h^{\mathcal{A}}$ be the configuration reached by $\mathcal{A}$ after processing the first h commands. Moreover, let C_h^{off} be the configuration reached by the optimal off-line algorithm after processing the first h commands. Let also $\text{opt}(\sigma) = \max_{1 \leq h \leq n} l(C_h^{\text{off}})$ and $l_{\mathcal{A}}(\sigma) = \max_{1 \leq h \leq n} l(C_h^{\mathcal{A}})$.

An on-line algorithm $\mathcal{A}$ is said to be *r-competitive* if there exists a constant b such that, for any instance σ , it holds that $l_{\mathcal{A}}(\sigma) \leq r \cdot \text{opt}(\sigma) + b$. An on-line algorithm $\mathcal{A}$ is said to be *strictly r-competitive* if, for any instance σ , it holds that $l_{\mathcal{A}}(\sigma) \leq r \cdot \text{opt}(\sigma)$.

A simple on-line algorithm for the load-balancing problem described above is the *greedy algorithm* that assigns a new task to the least loaded processor among those processors that can serve the task. That is, whenever a **new**(x, w) command is encountered and the current configuration is C , the greedy algorithm looks for the processor p_i in $P_{t=(x, w)}$ such that $l_C(i)$ is minimal and assigns the new task $t = (x, w)$ to p_i . (Ties are broken arbitrarily.)

2 (Sub-)graphs and (Sub-)greedy Algorithms

In the sequel we will describe an on-line load balancing algorithm whose competitive ratio depends on some combinatorial properties of $G(X \cup P, E)$. The two main ideas used in our approach are the following:

1. We remove some edges on G and then we apply the greedy algorithm *to the resulting bipartite graph*;
2. While removing edges we try to balance the number of processors used by tasks of type $x \in X$ and the number of processors that the adversary can use to assign the same set of tasks in the original graph G .

First of all, let us observe that our method aims to obtain a good competitive ratio by *adding further constraints* to the original problem: this indeed corresponds to remove a suitable set of edges from G . Choosing which edges to remove depends on some combinatorial properties we want the resulting bipartite graph to satisfy. Before giving a formal description of such properties, we will describe the basic idea behind our approach.

The main idea. Let us consider a generic iteration of an algorithm that has to assign a task of type $x \in X$. Assume that our algorithm takes into account a set $U(x)$ of processors and assigns the task to the least loaded one. In order to evaluate the competitive ratio of this approach we need to know which set of processors $A(x)$ an adversary can use to assign the overall load currently in $U(x)$ (as we will see in the sequel the competitive ratio of our algorithm is roughly $|A(x)|/|U(x)|$). In the following, we will show how the set $A(x)$ is determined by the choices of our algorithm in the previous steps (see Fig. 2).

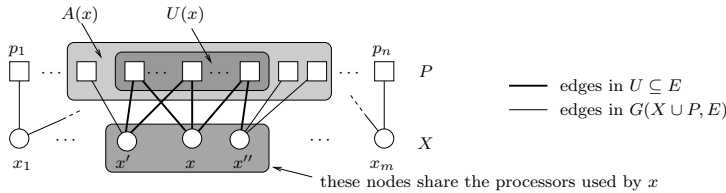


Fig. 2. The main idea of the sub-greedy algorithm is to balance the ratio between the number of used processors $|U(x)|$ and the number of processors available $|A(x)|$ to the adversary.

2.1 Analysis

In this section we formalize the idea above and we provide the performance analysis of the resulting algorithm.

Definition 1 (Used Processors). For any $x \in X$, we define a non-empty set $U(x) \subseteq \Gamma_G(x)$ of used processors. Moreover, given a processor $p \in P$, we denote by $U^{-1}(p)$ those vertices in X that have p as an available processor, i.e. $U^{-1}(p) = \{x \mid p \in U(x)\}$.

Definition 2 (Adversary Processors). For any $x \in X$ we denote by $A(x)$ those processors that an off-line adversary can use to balance the load assigned to $U(x)$. In particular, $A(x) = \bigcup_{p \in U(x)} \bigcup_{x' \in U^{-1}(p)} \Gamma_G(x')$.

Notice that the set $U(x)$ specifies a subset of the edges in $G(X \cup P, E)$ incident to x . By considering the union over all $x \in X$ of these edges and the resulting bipartite subgraph, we have the following:

Definition 3 (Sub-Greedy Algorithm). For any bipartite graph $G(X \cup P, E)$ and for any subset of edges $U \subseteq E$, the sub-greedy algorithm is defined as the greedy on-line algorithm applied to $G_U = G(X \cup P, U)$.

Remark 1. It is easy to see that the sub-greedy algorithm is a special case of the cluster algorithm in [9], since the latter imposes each connected component of G_U to be a complete bipartite graph [9, Definition of Cluster].

It is clear that the performance of the sub-greedy algorithm will depend on the choice of the set $U \subseteq E$. In particular, we can characterize its competitive ratio in terms of the ratio between the set of adversary processors $A(x)$ and the set of used processors $U(x)$. Let us consider the following quantities:

$$\rho_w(U) = \max_{x \in X} \left\{ \frac{|A(x)| - 1}{|U(x)|} \right\}, \quad \rho_u(U) = \max_{x \in X} \left\{ \frac{|A(x)|}{|U(x)|} \right\}.$$

Then, the following two results hold.

Theorem 1. The sub-greedy algorithm is strictly $(1 + \rho_w(U))$ -competitive in the case of weighted tasks.

Proof. Let p_i be the processor with the highest load and let $t = (w, x)$ be the last task assigned to p_i by the sub-greedy algorithm. Since t has been assigned to p_i whose load, before the arrival of t , was $l(i) - w$, we have that each processor in $U(x)$ had load at least $l(i) - w$. So, the overall load of $U(x)$ is at least $|U(x)|(l(i) - w) + w$. We now consider the number of processors that any off-line strategy can use to spread such load. Such a number is equal to $|A(x)|$, which implies that the optimal off-line solution has measure at least

$$l^* \geq \max \left\{ \frac{|U(x)l(i) - w(|U(x)| - 1)}{|A(x)|}, w \right\}.$$

The worst case is when we equate the two quantities, that is $l^* \geq w = \frac{|U(x)l(i)|}{|A(x)| + |U(x)| - 1}$, which implies the following bound on the competitive ratio

$$\frac{l(i)}{l^*} \leq \frac{|A(x)| + |U(x)| - 1}{|U(x)|} \leq 1 + \rho_w(U).$$

Hence, the theorem follows.

Theorem 2. *The sub-greedy algorithm is $\rho_u(U)$ -competitive (resp., strictly $\lceil \rho_u(U) \rceil$ -competitive) in the case of unweighted tasks.*

Proof. Let us consider a generic iteration of the sub-greedy algorithm in which a task t arising in $x \in X$ has been assigned to $p_i \in U(x)$. Since t has been assigned to p_i whose load, before the arrival of t , was $l(i) - 1$, we have that each processor in $U(x)$ had load at least $l(i) - 1$. This implies that the overall number of tasks in $U(x)$, after the arrival of t , was at least $|U(x)|(l(i) - 1) + 1$. Let us also observe that the number of processors to which the off-line optimal solution can assign these tasks is at most $|A(x)|$. Thus, the optimal off-line solution has measure at least

$$l^* \geq \frac{|U(x)|(l(i) - 1) + 1}{|A(x)|} \geq \frac{l(i) - 1}{\rho_u(U)} + \frac{1}{|A(x)|}. \quad (1)$$

By contradiction, let us suppose that $l(i) > l^* \lceil \rho_u(U) \rceil$. Then, since both $l(i)$ and l^* have integer values, we have that $l(i) - 1 \geq l^* \lceil \rho_u(U) \rceil \geq l^* \rho_u(U)$. This leads to the following contradiction:

$$l^* \geq \frac{l(i) - 1}{\rho_u(U)} + \frac{1}{|A(x)|} \geq \frac{l^* \rho_u(U)}{\rho_u(U)} + \frac{1}{|A(x)|} > l^*.$$

We have thus proved that the sub-greedy algorithm is strictly $\lceil \rho_u(U) \rceil$ -competitive. Finally, Eq. 1 implies

$$l(i) < l^* \rho_u(U) + \frac{1}{\rho_u(U)} \leq l^* \rho_u(U) + 1,$$

where the last inequality follows from the fact that $\rho_u(U) \geq 1$. So, the sub-greedy algorithm is also $\rho_u(U)$ -competitive. Hence, the theorem follows.

We next show the limits of our approach as it is and we generalize it in order to handle more cases. First, consider the bipartite graph $G(X \cup P, E)$ with $X = \{x_1, x_2, \dots, x_n\} \cup \{x_0\}$ and $E = \{(x_i, p_i) \mid 1 \leq i \leq n\} \cup \{(x_0, p_i) \mid 1 \leq i \leq n\}$. It is easy to see that any subset of edges U yields $\rho_w(U) = n - 1$. However, a rather simple idea might be to separate the high-degree vertex x_0 from the low-degree vertices $x_1, x_2, \dots, x_n$. So, tasks of type x_0 are processed *independently* from tasks of type $x_1, x_2, \dots, x_n$. It is possible to prove that this algorithm has a constant competitive ratio. This idea leads to the following:

Definition 4 (sub-greedy*). *Let $X_1, X_2, \dots, X_k$ be any partition of the set X of the task types vertices of $G(X \cup P, E)$. Also let $G_i = G(X_i \cup P, E_i)$ be the corresponding induced subgraph, and let $U_i \subseteq E_i$ for $1 \leq i \leq n$. We denote by sub-greedy* the algorithm assigning tasks of type in X_i as the sub-greedy algorithm on the subgraph of G_i corresponding to U_i , and with only these tasks as input (i.e., independently of tasks of other types).*

In the sequel we denote by $\rho_w(U_i, G_i)$ the quantity $\rho_w(U)$ computed w.r.t. the graph G_i and subset of edges $U_i \subseteq E_i$.

Theorem 3. *The sub-greedy* algorithm is strictly $(k + \rho_w^*(U))$ -competitive in the case of weighted tasks, where k is the number of subgraphs and $\rho_w^*(U) = \sum_{i=1}^k \rho_w(U_i, G_i)$.*

Proof. Given a sequence of tasks σ , let $\sigma(i)$ denote the subsequence containing tasks whose type is in X_i . Also let $l(j)$ denote the load of processor p_j at some time step and $l^i(j)$ the load at the same time step w.r.t. tasks corresponding to X_i only. Then, the definition of sub-greedy* and Theorem 1 imply $\max_{1 \leq j \leq n} l^i(j) \leq \text{opt}(\sigma(i))(1 + \rho_w(U_i, G_i))$, for $1 \leq i \leq k$. It then holds that

$$\max_{1 \leq j \leq n} l(j) \leq \sum_{i=1}^k \max_{1 \leq j \leq n} l^i(j) \leq \sum_{i=1}^k \text{opt}(\sigma(i))(1 + \rho_w(U_i, G_i)) \quad (2)$$

$$\leq k \cdot \text{opt}(\sigma) + \text{opt}(\sigma) \sum_{i=1}^k \rho_w(U_i, G_i), \quad (3)$$

where the last inequality follows from the fact that $\text{opt}(\sigma(i)) \leq \text{opt}(\sigma)$.

The above theorem will be a key-ingredient in deriving algorithms for the general case (see Sect. 4).

2.2 Computing Good Subgraphs

From Theorems 1-2 it is clear that, in order to attain a good competitive ratio, it is necessary to select a subset of edges $U \subseteq E$ such that $\rho(U)$ is as small as possible. Similarly, Theorem 3 implies that U should minimize $k + \rho_w^*(U)$, when considering the sub-greedy* algorithm.

We now rewrite the sets $A(x)$ and $U(x)$ by looking at the *open* neighborhood operator $\Gamma_G(\cdot)$. In particular, we have that $U(x) = \Gamma_{G_U}(x)$ and $A(x) = \Gamma_G(\Gamma_{G_U}(\Gamma_{G_U}(x)))$. When considering the weighted case, this leads to the following optimization problem:

Problem 5 *Min Weighted Adversary Subgraph (MWAS).*

Instance: A bipartite graph $G(X \cup P, E)$.

Solution: A subgraph $G_U = G(X \cup P, U)$, such that $U \subseteq E$ and, for every $x \in X$, $|\Gamma_{G_U}(x)| \geq 1$.

Measure: $\rho_w(U, G) = \max_{x \in X} \frac{|\Gamma_G(\Gamma_{G_U}(\Gamma_{G_U}(x)))| - 1}{|\Gamma_{G_U}(x)|}$.

Problem 6 *Min Weighted Adversary Multi-Subgraph (MWAMS).*

Instance: A bipartite graph $G(X \cup P, E)$.

Solution: A partition $X_1, X_2, \dots, X_k$ of X and a collection $U = \{U_1, \dots, U_k\}$ of subsets of edges $U_i \subseteq E_i$, where E_i denotes the set of edges of the subgraph $G_i = G(X_i \cup P, E_i)$ induced by X_i , such that, for every $1 \leq i \leq k$ and $x \in X_i$, $|\Gamma_{G_{U_i}}(x)| \geq 1$.

Measure: $k + \rho_w^*(U, G) = k + \sum_{i=1}^k \rho_w(U_i, G_i)$.

Similarly, the Min Unweighted Adversary Subgraph (MUAS) problem and the Min Unweighted Adversary Multi-Subgraph (MUAMS) problem are defined by replacing ρ_w with ρ_u in the two definitions above, respectively.

It is possible to construct a reduction showing the NP-hardness of all these problems (see [10]). Also, the same reduction is a gap-creating reduction, thus implying the non existence of a PTAS for any such problem. In particular, we obtain the following result:

Theorem 4. *The MUAS and MWAS problems cannot be approximated within a factor smaller than $7/6$ and $3/2$, respectively, unless $P = NP$. Moreover MUAMS and MWAMS cannot be approximated within a factor smaller than $11/10$ and $5/4$, respectively, unless $P = NP$.*

3 Application to Hierarchical Server Topologies

In this section we apply our method to the hierarchical server topologies introduced in [7]. In particular, we consider the *linear* hierarchical topology: Processors are ordered from p_1 (the most capable processor) to p_n (the least capable processor) in decreasing order with respect to their capabilities. So, if a task can be assigned to processor p_i , for some i , then it can also be assigned to any p_j with $1 \leq j < i$. We can therefore consider task types corresponding to the intervals $\{p_1, p_2, \dots, p_i\}$, for each $1 \leq i \leq n$.

The resulting bipartite graph $G(X \cup P, E)$ is given by $X = \{x_1, \dots, x_n\}$, $P = \{p_1, \dots, p_n\}$ and $E = \{(x_i, p_j) \mid x_i \in X, p_j \in P, j \leq i\}$. We denote this graph as K_n^{hst} .

We next provide an efficient construction of subgraphs of K_n^{hst} .

Lemma 1. *For any positive integer n , there exists a U such that $\rho_w(U, K_n^{hst}) \leq (4n - 2)/(n + 2)$ and $\rho_u(U, K_n^{hst}) \leq (4n)/(n + 2)$. Moreover, the set U can be computed in linear time.*

Proof. For each $1 \leq i \leq n$, we define the set $U(x_i)$ as

$$U(x_i) = \{p_{\lceil i/2 \rceil}, p_{\lceil i/2 \rceil + 1}, \dots, p_i\} = \begin{cases} \{p_{i/2}, p_{i/2+1}, \dots, p_i\} & \text{if } i \text{ is even,} \\ \{p_{(i+1)/2}, p_{(i+1)/2+1}, \dots, p_i\} & \text{otherwise.} \end{cases}$$

Clearly, $|U(x_i)| = i/2 + 1$ if i is even, and $|U(x_i)| = (i + 1)/2$ otherwise. Moreover, $|A(x_i)| = \max_{i \leq j \leq n} \{j \mid U(x_i) \cap U(x_j) \neq \emptyset\}$. It is easy to see that $|A(x_i)| \leq 2i$, thus implying

$$\rho_w(U) = \max_{1 \leq i \leq n} \frac{|A(x_i)| - 1}{|U(x_i)|} \leq \max_{1 \leq i \leq n} \frac{4i - 2}{i + 1} \leq \frac{4n - 2}{n + 1}.$$

A better bound can be obtained by distinguishing two cases: 1) $i \leq n/2$ and 2) $i \geq n/2 + 1$. In case 1) we apply the bound above, while for 2) we simply use $|A(i)| \leq n$; in both cases we obtain $\rho_w(U) \leq (4n - 2)/(n + 2)$.

With a similar proof we can show that the same construction yields $\rho_u(U) \leq (4n)/(n + 2)$.

An immediate application of Lemma 1 combined with Theorems 1-2 is the following result:

Theorem 5. *For linear hierarchy topologies the sub-greedy algorithm is strictly $(5n)/(n+2)$ -competitive in the case of weighted tasks, and $(4n)/(n+2)$ -competitive and strictly 4-competitive for unweighted tasks.*

Notice that our approach improves over the 5-competitive (respectively, 4-competitive) algorithm for weighted (respectively, unweighted) tasks given in [7].

Remark 2. Observe that, if we impose the subgraph to be a set of complete bipartite graphs as in [9], then K_n^{hst} does not admit a construction yielding $O(1)$ -competitive algorithms. So, for these topologies, the sub-greedy algorithm constitutes a significant improvement w.r.t. the result in [9].

4 The General Case

In this section we provide a sufficient condition for obtaining $O(\sqrt{n})$ -competitive algorithms. This applies to the hierarchical server model when the order of the servers is a tree. Thus, in this case our result is optimal because of the $\Omega(\sqrt{n})$ lower bound [7].

We first define an overall strategy to select the set $U(x)$ depending on the degree $\delta(x)$ of x :

High degree (easy case): $\delta(x) \geq \sqrt{n}$. In this case we use *all of its adjacent vertices in P* . Since $|U(x)| = \delta(x) \geq \sqrt{n}$, we have $|A(x)|/|U(x)| \leq \sqrt{n}$.

Low degree (hard case): $\delta(x) < \sqrt{n}$. For low degree vertices our strategy will be to choose a *single* processor p_x^* in $\Gamma_G(x) \subseteq P$. The choice of this element must be carried out carefully so to guarantee $|A(x)| \leq \sqrt{n}$. For instance, it would suffice that p_x^* does not appear in any other set $U(x')$.

Then, our next idea will be to partition the graph $G(X \cup P, E)$ into two subgraphs $G_l(X_l \cup P, E_l)$ and $G_h(X_h \cup P, E_h)$ containing low and high degree vertices, respectively. Notice that, if we are able to have a $f(n)$ -competitive algorithm for the low degree graph, then we have a $O(\sqrt{n} + f(n))$ -competitive algorithm for our problem (see Theorem 3).

We next focus on low degree graphs and we provide sufficient conditions for $O(\sqrt{n})$ -competitive algorithms.

Theorem 6. *If $G_l(X_l \cup P, E_l)$ admits a b -matching, then the sub-greedy* algorithm is at most $((b+1)\sqrt{n}+2)$ -competitive.*

Proof. Let U be a b -matching for G_l . It is easy to see that in G_l $|A(x)| \leq b\sqrt{n}$, for all $x \in X_l$. Thus, $\rho_w(U, G_l) \leq b\sqrt{n}$. By definition of G_h , $\rho_w(E_h, G_h) \leq \sqrt{n}$. We can thus apply Theorem 3 with $k = 2$, $U_1 = U$ and $U_2 = E_h$. Hence the theorem follows.

Theorem 7. *If $G(X \cup P, E)$ admits a b -matching, then the sub-greedy* algorithm is at most $(2\sqrt{bn} + 2)$ -competitive.*

Proof Sketch. Define low-degree vertices as those $x \in X$ such that $|\Gamma_G(x)| \leq \sqrt{n/b}$. Subgraphs G_l and G_h are defined accordingly. The existence of a b -matching U yields $\rho_w(U, G_l) \leq b\sqrt{n/b}$. By definition of G_h , $\rho_w(E_h, G_h) \leq \sqrt{bn}$. We can thus apply Theorem 3 with $k = 2$, $U_1 = U$ and $U_2 = E_h$. Hence the theorem follows.

Theorem 8. *If $G(X \cup P, E)$ admits a matching, then the sub-greedy algorithm is at most $\delta_{\max}$ -competitive, where $\delta_{\max} = \max_{x \in X} |\Gamma_G(x)|$.*

Proof. Let U be a matching for G . Then, $|A(x)| \leq |\Gamma_G(x)|$ and $|U(x)| = 1$, for any $x \in X$.

4.1 Generalized Hierarchical Server Topologies

We now apply these results to the hierarchical model in the case in which the ordering of the servers forms a tree. Figure 1 shows an example of this problem version: processors are arranged on a rooted tree and there is a task type x_i for each node p_i of the tree; a task of type x_i can be assigned to processor p_i or to any of its ancestors.

We first generalize this problem version to a more general setting:

Definition 7. *Let $H(P, F)$ be a directed graph. The associated bipartite graph $G_H(X \cup P, E)$ is defined as $X = \{x_1, x_2, \dots, x_n\}$, and $(x_i, p_j) \in E$ if and only if $i = j$ or there exists a directed path in H from p_i to p_j .*

We can model a tree hierarchy by considering a rooted tree T whose edges are directed *upward*. We then obtain the following:

Theorem 9. *For any rooted tree $T(P, E)$ the corresponding graph $G_T(X \cup P, E)$ admits a matching. In this case, the sub-greedy* algorithm is always at most $(2\sqrt{n} + 2)$ -competitive. Moreover, the sub-greedy algorithm is at most h -competitive, where h is the height of T .*

Proof. It is easy to see that $M = \{(x_i, p_i) \mid 1 \leq i \leq n\}$ is a matching for $G_T(X \cup P, E)$. We can thus apply Theorem 6 with $b = 1$.

Theorem 10. *Let $H(P, F)$ be any directed graph representing an ordering among processors, and let $G_H(X \cup P, E)$ be the corresponding bipartite subgraph. Then, the sub-greedy* algorithm is at most $(2\sqrt{n} + 2)$ -competitive.*

Proof Sketch. We first reduce every strongly connected component of H to a single vertex, since processors of this component are equally powerful: if a task can be assigned to a processor of this component, then it can also be assigned to any other processor of the same component. (Equivalently, this transformation

does not affect G_H .) So, we can assume H being not acyclic. We then greedily construct a matching U by repeating the following three steps: 1) for a p_i with no outgoing edges, include the edge (x_i, p_i) in U ; 2) remove p_i and x_i from both H and G_H .

Since H is acyclic, such a vertex p_i must exist. Moreover, in G_H , p_i is adjacent to x_i only (otherwise, p_i must have one out-going edge in H). Removing p_i and x_i from G_H yields the graph corresponding to $H \setminus \{p_i\}$. After step 3), we are left with a new H' and $G_{H'}$ which enjoys the same property as H . So, we can iterate this procedure until all vertices in H are removed. Since the number of task types equals the number of vertices of H , this method yields a matching for G_H .

5 Conclusions and Open Problems

We have presented a novel technique which allows to derive on-line algorithms with a simple modification of the greedy one. This modification preserves the good feature of deciding where to assign a task solely based on the current load of processors to which that task can be potentially assigned to. Indeed, the pre-computation of the subgraph required by our approach is performed off-line given the graph representing the problem constraints. Additionally, for several cases we have considered here, this subgraph is only used in the analysis, while the resulting algorithms are simple modifications of the greedy *implementing* the subgraph: the construction of Lemma 1 yields an algorithm performing a greedy choice on the rightmost half of the available processors $\Gamma_G(x_i) = \{p_1, \dots, p_i\}$. So, this algorithm can be implemented even without knowing n . A similar argument applies to the sub-greedy* algorithm with the subgraph of Theorem 9: in this case knowing n is enough to decide whether a vertex as “low-degree” or not; in the latter case the matching (x_i, p_i) yields a fixed assignment for tasks corresponding to type x_i .

In general, the adopted strategy of the sub-greedy* algorithm depends on the type of the task and on the current load of the adjacent processors in the appropriate subgraph. Since the algorithm assigns tasks corresponding to different subgraphs *independently*, it must be able to compute the load of a processor w.r.t. a subset X_i ; this can be easily done whenever tasks are specified as pairs (x, w) .

So, our algorithms are distributed and, for the generalized hierarchical topologies, their competitive ratio is only slightly worse than the $2\sqrt{n} + 1$ upper bound provided by the Robin-Hood algorithm [5]. Also, for tree hierarchical topologies, our analysis yields a much better ratio whenever the height h of the tree is $o(\sqrt{n})$ (e.g., for balanced trees).

An interesting direction for future research might be that of characterizing the competitive ratio of distributed algorithms under several assumptions on the graph $G(X \cup P, E)$: 1) G is *unknown*, 2) G is uniquely determined by n , but n is unknown, 3) G is known.

A related question is: under which hypothesis does our technique yield optimal competitive ratios?

Acknowledgements. The fourth author wishes to thank Amotz Bar-Noy for a useful discussion and for bringing the work [7] to his attention.

References

1. S. Albers. Better bounds for on-line scheduling. *Proc. of the 29th ACM Symp. on Theory of Computing (STOC)*, pages 130–139, 1997.
2. Y. Azar. *On-line load balancing*, chapter in “On-line Algorithms - The state of the Art”, A. Fiat and G. Woeginger (eds.). Springer Verlag, 1998.
3. Y. Azar, A. Broder, and A. Karlin. Online load balancing. *Theoretical Computer Science*, 130:73–84, 1994.
4. Y. Azar and L. Epstein. On-line load balancing of temporary tasks on identical machines. *Proc. of the 5th Israeli Symposium on Theory of Computing and Systems (ISTCS)*, pages 119–125, 1997.
5. Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. Online load balancing of temporary tasks. *Journal of Algorithms*, 22:93–110, 1997.
6. Y. Azar, J. Naor, and R. Rom. The competitiveness of online assignments. *Journal of Algorithms*, 18:221–237, 1995.
7. A. Bar-Noy, A. Freund, and J. Naor. On-line load balancing in a hierarchical server topology. *SIAM Journal on Computing*, 31(2):527–549, 2001. Preliminary version in Proc. of the 7th Annual European Symposium on Algorithms, ESA’99.
8. D. Braess. Ueber ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968.
9. P. Crescenzi, G. Gambosi, and P. Penna. On-line algorithms for the channel assignment problem in cellular networks. In *Proc. of the 4th ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing (DIALM)*, pages 1–7, 2000. Full version to appear in Discrete Applied Mathematics.
10. P. Crescenzi, G. Gambosi, and P. Penna. On-line load balancing made simple: Greedy strikes back. Technical report, Università di Salerno, 2003. Electronic version available at <http://www.dia.unisa.it/~penna>.
11. R. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
12. R. Graham. Bounds on multiprocessor timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.
13. S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1998.
14. E. Tardòs J.K. Lenstra, D.B. Shmoys. Approximation algorithms for scheduling unrelated parallel machines. *Math. Programming*, 46:259–271, 1990.
15. J. D. Murchland. Braess’s paradox of traffic flow. *Transportation Research*, 4:391–394, 1970.
16. S. Phillips and J. Westbrook. Online load balancing and network flow. *Algorithmica*, 21(3):245–261, 1998.
17. Serge Y. Ma and A. Plotkin. An improved lower bound for load balancing of tasks with unknown duration. *Information Processing Letters*, 62(6):301–303, 1997.

Real-Time Scheduling with a Budget

Joseph (Seffi) Naor¹, Hadas Shachnai^{2*}, and Tami Tamir³

¹ Computer Science Dept., Technion, Haifa 32000, Israel
`naor@cs.technion.ac.il`,

² Bell Labs, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974.
`hadas@research.bell-labs.com`,

³ Dept. of Computer Science and Eng., Box 352350, Univ. of Washington, Seattle, WA 98195.
`tami@cs.washington.edu`

Abstract. Suppose that we are given a set of jobs, where each job has a processing time, a non-negative weight, and a set of possible time intervals in which it can be processed. In addition, each job has a processing cost. Our goal is to schedule a feasible subset of the jobs on a single machine, such that the total weight is maximized, and the cost of the schedule is within a given budget. We refer to this problem as *budgeted real-time scheduling (BRS)*. Indeed, the special case where the budget is *unbounded* is the well-known real-time scheduling problem. The second problem that we consider is *budgeted real-time scheduling with overlaps (BRSO)*, in which several jobs may be processed simultaneously, and the goal is to maximize the time in which the machine is utilized. Our two variants of the real-time scheduling problem have important applications, in vehicle scheduling, linear combinatorial auctions and QoS management for Internet connections. These problems are the focus of this paper.

Both BRS and BRSO are strongly NP-hard, even with unbounded budget. Our main results are $(2 + \varepsilon)$ -approximation algorithms for these problems. This ratio coincides with the best known approximation factor for the (unbudgeted) real-time scheduling problem, and is slightly weaker than the best known approximation factor of $e/(e - 1)$ for the (unbudgeted) real-time scheduling with overlaps, presented in this paper. We show that better ratios (or simpler approximation algorithms) can be derived for some special cases, including instances with unit-costs and the budgeted *job interval selection problem (JISP)*. Budgeted JISP is shown to be APX-hard even when overlaps are allowed and with unbounded budget. Finally, our results can be extended to instances with multiple machines.

1 Introduction

In the well-known *real-time scheduling* problem (also known as the *throughput maximization* problem), we are given a set of n jobs; each job J_j has a processing

* On leave from the Computer Science Dept., Technion, Haifa 32000, Israel.

time p_j , a non-negative weight w_j , and a set of time intervals in which it can be processed (given as either a window with release and due-dates or as a discrete set of possible processing intervals). The goal is to schedule a feasible subset of the jobs on a single machine, such that the overall weight of the scheduled jobs is maximized. In this paper we consider two variants of this problem.

In the *budgeted real-time scheduling (BRS)* problem, each job J_j has a processing cost c_j . A budget B is given, and the goal is to find a maximum weight schedule, among the feasible schedules whose total processing cost is at most B . In *real-time scheduling with overlaps (RSO)*, the jobs are scheduled on a single *non-bottleneck* machine, which can process simultaneously several jobs. The goal is to maximize the overall time in which the machine is utilized (i.e., processes at least one job).¹ In the budgeted case (*BRSO*), each job J_j has a processing cost c_j . The goal is to maximize the time in which the machine is utilized, among the schedules with total processing cost at most B .

In our study of BRS, RSO and BRSO, we distinguish between *discrete* and *continuous* instances. In the discrete case, each job J_j can be scheduled to run in one of a given set of n_j intervals $I_{j,\ell}$ ($\ell = 1, \dots, n_j$). The special case where each job has at most k intervals, i.e., $\forall j, n_j \leq k$, is called *JISP_k*. In the continuous case, job J_j has release date r_j , due date d_j , and a processing time p_j . It is possible to schedule J_j in any interval $[s_j, e_j]$ such that $s_j \geq r_j$, $e_j \leq d_j$, and $e_j - s_j = p_j$. We consider also *JISP₁*, where each job can be processed in the single interval $I_{j,1} = [r_j, d_j]$, and $p_j = d_j - r_j$.

We consider general (discrete and continuous) instances, where each job has processing time p_j , a weight w_j , and a processing cost c_j . For some variants we study also classes of instances in which (i) jobs have unit-costs (that is, $c_j = 1 \forall j$), or (ii) for all the jobs $w_j = p_j$.

The BRS and BRSO problems extend the classic real-time scheduling problem to model the natural goal of gaining the maximum available service for a given budget. In particular, the following practical scenarios yield instances of our problems.²

Multi-Vehicle Scheduling on a Path: The vehicle scheduling problem arises in many applications, including robot handling in manufacturing systems and secondary storage management in computer systems (see e.g. [KN-01]). Suppose that a fleet of vehicles needs to service requests on a path. There is an operation cost to each vehicle, and a segment on the line in which the vehicle can provide service. Our objective is to assign the vehicles to service requests on line segments such that the total length of the union of the line segments, i.e., the part of the line which is covered, is maximized, yet the overall cost is within some budget constraints.

Combinatorial Auctions: In auctions used in e-commerce, a buyer needs to complete an order for a given set of goods. There is a collection of *sellers*, each

¹ Note that job weights have no effect on the objective function.

² Other applications, including transmission of continuous-media data and crew scheduling, are given in [NST-03].

offers a subset (or *bundle*) of the goods at some cost. Each of the goods g_i is associated with a weight w_i , which indicates its priority in the order. The buyer needs to satisfy a fraction of the order of maximum weight, by selecting a subset of the offers, such that the total cost is bounded by the buyer's budget, B . In auctions for *linear* goods (see, e.g., in [T-00]), we have an ordered list of m goods $g_1, \dots, g_m$, and the offers should refer to bundles of the form $g_i, g_{i+1}, \dots, g_{j-1}, g_j$. Note that while selecting a subset of the offers we allow overlaps, i.e., the buyer may acquire more than the needed amount from some good; however, this does not decrease the cost of any of the offers. Thus, we get an instance of the *BRSO* problem, where any job J_j can be processed in one possible time interval.

QoS Upgrade in a Network: Consider an end-to-end connection between s and t that uses several Internet service providers (ISP). Each ISP provides a basic service (for free) and to upgrade the service one needs to pay; that is, an ISP can decrease the delay in its part of the path for a certain cost. (See, e.g., [LORS-00, LO-02].) The end-to-end delay is additive (over all ISP-s). We have a budget and we need to decide on how to distribute it between the ISP-s. In certain scenarios, an ISP may need to choose to upgrade only a portion of the part of the $s - t$ path that it controls, however, it has the freedom to choose which portion. In this problem instance, “jobs” (upgraded segments) are allowed to overlap.

1.1 Our Results

We give hardness results and approximation algorithms for *BRS*, *RSO*, and *BRSO*. Specifically, we show that continuous *RSO* is strongly NP-hard.³ In the discrete case, both *BRS* and *BRSO* are shown to be APX-hard, already for instances where $\forall j, n_j \leq k$ ($JISP_k$), and where all the intervals corresponding to a job have the same length, for any $k \geq 3$. In Section 3, we present a $(2 + \varepsilon)$ -approximation algorithm for *BRS* (both discrete and continuous). We build on the framework of Jain and Vazirani [Va-01] for using Lagrangian relaxation in developing approximation algorithms. Our algorithm is based on a novel combination of Lagrangian relaxation with efficient search on the set of feasible solutions. We show that a simple Greedy algorithm yields a 4-approximation for *BRS* with unit costs, where $w_j = p_j \forall j$.

In Section 4, we give a $(2 + \varepsilon)$ -approximation algorithm for continuous inputs of *BRSO*, and a $(3 + \varepsilon)$ -approximation for discrete inputs, using the Lagrangian relaxation technique. For *RSO* we present a Greedy algorithm that achieves the ratio of 2. An improved ratio of $e/(e - 1)$ is obtained by a randomized algorithm (where e denotes the base of the natural logarithm). For $JISP_1$, we obtain an optimal solution for instances of *BRSO* with unit costs, and a *fully polynomial time approximation scheme (FPTAS)* for arbitrary costs. (Note that $JISP_1$ is weakly NP-hard. This can be shown by reduction from Knapsack [GJ-79].) Finally, in Section 5 our results are shown to extend to instances of *BRS* and *BRSO* in which the jobs can be scheduled on multiple machines.

³ The continuous real-time scheduling problem (with no overlaps) is known to be strongly NP-hard [GJ-79].

The approximation technique that we use for deriving our $(2 + \varepsilon)$ -approximation results (see in Section 2) is shown to apply to a fundamental class of budgeted maximization problems, including *throughput maximization in a system of dependent jobs*, which generalizes the *BRS* problem (see in [NST-03]). We show that, using the technique, any problem in the class which has an LP based ρ -approximation with unbounded budget, can be approximated within factor $\rho + \varepsilon$ in the budgeted case, for any $B \geq 1$.

Due to space constraints we state some of the results without proofs.⁴

1.2 Related Work

To the best of our knowledge, the *budgeted* real-time scheduling problem is studied here for the first time. There has been extensive work on real-time scheduling, both in the discrete and the continuous models. Garey and Johnson (cf. [GJ-79]) showed that the continuous case is strongly NP-hard, while the discrete case, JISP, was shown by Spieksma [S-99] to be APX-hard, already for instances of $JISP_k$, where $k \geq 2$. Bar-Noy et al. [BG⁺99,BB⁺00] and independently Berman and DasGupta [BD-00] presented 2-approximation algorithms for the discrete case,⁵ and $(2 + \varepsilon)$ ratio in the continuous case. As shown in [BB⁺00], this ratio holds for arbitrary number of machines. While none of the existing techniques has been able to improve upon the 2 and $(2 + \varepsilon)$ ratios for general instances of the real-time scheduling problem, improved bounds were obtained for some special cases. In particular, Chuzhoy et al. [COR-01] considered the unweighted version, for which they gave an $(e/(e - 1) + \varepsilon)$ -approximation algorithm, where ε is any constant. For other special cases, they developed polynomial time approximation schemes. Finally, some special cases of JISP were shown to be polynomially solvable (see, e.g., in [AS-87,B-99]).

We are not aware of previous work on the *RSO* and *BRSO* problems. Since overlaps are allowed and the goal is to maximize the overall time in which the machine is utilized, these problems can be viewed as maximum coverage problems. In previous work on budgeted covering (see, e.g., [KMN-99]), the covering items are *sets*; once a set is selected, the covered elements are uniquely defined. In contrast, in *RSO* (and *BRSO*) the covering items are *jobs*, and we can choose the time segments (= elements) that will be covered by a job, by determining the time interval in which this job is processed.

2 Approximation via Lagrangian Relaxation

We describe below the general approximation technique that we use for deriving our results for *BRS* and *BRSO*. Our approach builds on the framework

⁴ The detailed proofs are given in [NST-03].

⁵ A 2-approximation for *unweighted* JISP is obtained by a Greedy algorithm, as shown in [S-99].

developed by Jain and Vazirani [Va-01][pp. 250-251] (see also [Ga-96]), for using Lagrangian relaxations in approximation algorithms. Our approach applies to the following class of *subset selection* problems. The input for any problem in the class consists of a set of elements $A = \{a_1, \dots, a_n\}$; each element $a_j \in A$ is associated with a weight w_j , and the cost of adding a_j to the solution set is $c_j \geq 1$. We have a budget $B \geq 1$. The goal is to find a subset of the elements $A' \subseteq A$ satisfying a given set of constraints (including the budget constraint), such that the total weight is maximized. We assume that any problem Π in the class satisfies the following property. **(P1)** Let A' be a feasible solution for Π ; then, any subset $A'' \subseteq A'$ is also a feasible solution.

Denote by $x_j \in \{0, 1\}$ the indicator variable for the selection of a_j . The integer program for Π has the following form.

$$\begin{aligned}
 (\Pi) \quad & \text{maximize} && \sum_{a_j \in A} w_j x_j \\
 & \text{subject to :} && \text{Constraints : } C_1, \dots, C_r \\
 & && \sum_j c_j x_j \leq B.
 \end{aligned}$$

In the linear relaxation we have $x_j \in [0, 1]$. The Lagrangian relaxation of this program is

$$\begin{aligned}
 (L - \Pi(\lambda)) \quad & \text{maximize} && \lambda \cdot B + \sum_{a_j \in A} (w_j - c_j \lambda) x_j \\
 & \text{subject to :} && \text{Constraints : } C_1, \dots, C_r
 \end{aligned}$$

Assume that $\mathcal{A}_\pi$ is a ρ -approximation algorithm to the optimal integral solution for $L - \Pi(\lambda)$, for any value of $\lambda > 0$. Thus, there exist values $\lambda_1 < \lambda_2$ such that $\mathcal{A}_\pi$ finds integral ρ -approximate solutions $\mathbf{x}_1, \mathbf{x}_2$ for $L - \Pi(\lambda_1), L - \Pi(\lambda_2)$, respectively, and the budgets used in these solutions are B_1, B_2 , where

$$B_2 < B < B_1. \tag{1}$$

Let W_1, W_2 denote the weights of the solutions $\mathbf{x}_1, \mathbf{x}_2$, then $W_i = \lambda_i B + \sum_{a_j \in A} (w_j - c_j \lambda_i) \mathbf{x}_{ij}$, $i \in \{1, 2\}$, $1 \leq j \leq n$. W.l.o.g, we assume that $W_1, W_2 \geq 1$.

Following the framework of [Va-01], we require that $\mathcal{A}_\pi$ satisfies the following property. Let $\alpha = (B - B_2)/(B_1 - B_2)$, then the convex combination of the solutions $\mathbf{x}_1, \mathbf{x}_2$, namely, $\mathbf{x} = \alpha \mathbf{x}_1 + (1 - \alpha) \mathbf{x}_2$ is a (fractional) ρ -approximate solution that uses the budget B . This is indeed the case if, for example, the solutions $\mathbf{x}_1, \mathbf{x}_2$ are obtained from a primal-dual algorithm. In this case, a convex combination of the dual solutions corresponding to $\mathbf{x}_1$ and $\mathbf{x}_2$ can be used to prove this property. This will be heavily used in our algorithms for the *BRS* and *BRSO* problems. Our goal is to find a feasible integral solution whose weight is close to the weight of $\mathbf{x}$. We show that for the class of subset selection problem

that we consider here, by finding ‘good’ values of λ_1, λ_2 , we obtain an integral solution that is within factor $\rho + \varepsilon$ from the optimal. The running time of our algorithm is dominated by the complexity of the search for λ_1, λ_2 and the running time of $\mathcal{A}_\pi$.

We now summarize the steps of the algorithm, $\mathcal{A}_L$, which gets as input the set of elements $a_1, \dots, a_n$, an accuracy parameter $\varepsilon > 0$ and the budget $B \geq 1$. Let $c = \sum_j c_j$ denote the total cost of the instance.

1. Let $\varepsilon' = \varepsilon/c$.
2. Define the *modified weight* of an element a_j to be $w'_j = w_j/c_j$.
Let $\omega_1 \leq \dots \leq \omega_R$ be the set of R distinct values of modified weights.
3. Find in $(0, \omega_R)$ the values of $\lambda_1 < \lambda_2$
satisfying (1), such that $\lambda_2 - \lambda_1 \leq \varepsilon'$.
4. Output the (feasible) integral solution found by $\mathcal{A}_\pi$ for $L - \Pi(\lambda_2)$.

Analysis: The heart of our approximation technique is the following theorem.

Theorem 1. *For any $0 < \varepsilon'$ and λ_1, λ_2 satisfying (1), if $0 < \lambda_2 - \lambda_1 < \varepsilon'$, then $W_2 \geq W_1 - \varepsilon'c$, where c is the total cost of the instance.*

Proof. We note that for a fixed value of λ we can omit from the input elements a_j for which $w'_j = w_j/c_j \leq \lambda$. We denote by S_i the *feasible* set of modified weights for λ_i , i.e., the set of values ω_ℓ satisfying $\omega_\ell \geq \lambda_i$; then $S_2 \subseteq S_1$. Let $A_i \subseteq S_i$ be the set of elements selected by $\mathcal{A}_\pi$ for the solution, for the given value λ_i . Then, $W_1 = \lambda_1 B + \sum_{A_1} (w_j - c_j \lambda_1)$ and $W_2 = \lambda_2 B + \sum_{A_2} (w_j - c_j \lambda_2)$. We handle two cases separately.

- (i) The feasible sets for λ_1, λ_2 are identical, that is, $S_1 = S_2$. Then

$$W_2 \geq \lambda_2 B + \sum_{A_1} (w_j - c_j \lambda_2) \geq \lambda_1 B + \sum_{A_1} (w_j - c_j (\lambda_1 + \varepsilon')) = W_1 - \varepsilon' \sum_{A_1} c_j \geq W_1 - \varepsilon' c$$

The leftmost inequality follows from the fact that all the elements in A_1 were feasible also with λ_2 . (Note that we can guarantee that the inequality is satisfied, by comparing W_2 with the weight of A_1 at λ_2 , and by taking the subset which gains the maximum of these two weights.)

- (ii) The feasible set for λ_1 contains some elements whose modified weights that are not contained in S_1 , that is, $S_2 \subset S_1$. For simplicity, we assume that $S_1 = \{\omega_{\ell+1}, \omega_{\ell+2}, \dots, \omega_R\}$, while $S_2 = \{\omega_{\ell+2}, \dots, \omega_R\}$, that is, for some $1 \leq \ell < R$, $\omega_{\ell+1} \in S_1$ and $\omega_{\ell+1} \notin S_2$. In general, several modified weight values may be contained in S_1 but not in S_2 . A similar argument can be applied for this case. Denote by $\hat{A}_1$ the subset of elements in A_1 whose modified weights are equal to $\omega_{\ell+1}$. Then,

$$\begin{aligned} W_2 &\geq \lambda_2 B + \sum_{A_1 \setminus \hat{A}_1} (w_j - c_j \lambda_2) \geq \lambda_1 B + \sum_{A_1 \setminus \hat{A}_1} (w_j - c_j (\lambda_1 + \varepsilon')) \\ &= \lambda_1 B + \sum_{A_1} (w_j - \lambda_1 c_j) - \sum_{\hat{A}_1} (w_j - \lambda_1 c_j) - \varepsilon' \sum_{A_1 \setminus \hat{A}_1} c_j \\ &= W_1 - \sum_{\hat{A}_1} c_j (\omega_{\ell+1} - \lambda_1) - \varepsilon' \sum_{A_1 \setminus \hat{A}_1} c_j \geq W_1 - \varepsilon' \sum_{A_1} c_j \geq W_1 - \varepsilon' c \end{aligned}$$

The first inequality is due to the fact that the set of elements $A_1 \setminus \hat{A}_1$ was available with λ_2 , and that Π satisfies property **(P1)**; the second inequality follows from the difference $(\lambda_2 - \lambda_1)$ being bounded by ε' ; the last inequality follows from $(\omega_{\ell+1} - \lambda_1) < \varepsilon'$. This completes the proof. $\square$

Let $0 < \varepsilon < 1$ be an input parameter, then taking $\varepsilon' = \varepsilon/c$, we get from Theorem 1 that

$$W_2 \geq (W_1 - \varepsilon'c)\alpha + W_2(1 - \alpha) \geq (W_1\alpha + W_2(1 - \alpha)) - \varepsilon'c \geq (W_1\alpha + W_2(1 - \alpha))(1 - \varepsilon).$$

Finally, since $\mathbf{x}$ gives a ρ -approximation to the optimal, we get

Theorem 2. *Algorithm $\mathcal{A}_L$ achieves an approximation factor of $(\rho + \varepsilon)$ for Π .*

Implementation: Note that to obtain a $(\rho + \varepsilon)$ -approximation, we need to find the values of $\lambda_1, \lambda_2 \in (0, \omega_R)$ that satisfy (1), such that $(\lambda_2 - \lambda_1) < \varepsilon/c$. As $\omega_R = \max_j w_j/c_j$ may be arbitrarily large, a naive search may require exponential number of steps. We show that by allowing a small increase (of ε) in the approximation ratio, we can implement this search in polynomial time. (i) Initially, we guess the weight of an optimal integral solution, W^* , to within factor $(1 - \varepsilon)$. This can be done in $O(\lg(n/\varepsilon))$ steps, since $\max_j w_j \leq W^* \leq n \cdot \max_j w_j$. We then omit from the input any element a_j whose weight is smaller than $\varepsilon W^*/n$. We scale the weights of the remaining elements, so that all the weights are in the range $[1, n/\varepsilon]$. (ii) For any element a_j with $c_j < \varepsilon B/n$, we round up c_j to $\varepsilon B/n$. We scale the other costs, such that all costs are in $[1, n/\varepsilon]$. (iii) We scale accordingly the size of the interval $(0, \omega_R)$.

Now, we argue that in the above scaling and rounding we only slightly decrease the weight of the solution. Indeed, by omitting elements with ‘small’ weights, we decrease the total weight of the elements selected by $\mathcal{A}_\pi$ at most by factor of ε . Also, by rounding up the ‘small’ costs to $\varepsilon B/n$, we get that the total weight obtained by $\mathcal{A}_\pi$ at λ_2 is at least $\lambda_2 B + \sum_{a_j \in A_2} (w_j - (c_j + \frac{\varepsilon B}{n}) \lambda_2) \geq W(1 - \varepsilon)$. Thus, overall we lose a factor of 2ε in the approximation ratio. The overall running time of our search procedure is $O(\lg(n/\varepsilon) \cdot \lg(n^3/\varepsilon^3)) = O(\lg^2(n/\varepsilon^3))$. It follows that the running time of $\mathcal{A}_L$ is $O(\lg^2(n/\varepsilon^3))$ times the running time of algorithm $\mathcal{A}_\pi$.

3 Approximating Bounded Real-Time Scheduling

3.1 A Greedy Algorithm

Consider first the special case of unit cost jobs, where $w_j = p_j \forall j$. Suppose that the budget is $B = k$; thus, we need to select a subset of k non-overlapping jobs, such that machine utilization is maximized. For such instances, we can obtain a constant approximation using an $O(n \log n)$ greedy algorithm. The algorithm $\mathcal{A}_G$ (formulated for continuous inputs) first sorts the jobs in non-increasing order by their processing times. Then, $\mathcal{A}_G$ schedules at most k jobs by scanning the sorted list; that is, while there is available budget, the next job J_j is scheduled in the earliest available time interval in $[r_j, d_j]$.

Theorem 3. $\mathcal{A}_G$ is a $(4 + \varepsilon)$ -approximation for BRS with unit costs where $w_j = p_j \forall j$. In the discrete case, $\mathcal{A}_G$ achieves the ratio of 4.

As we show below, better ratio can be achieved, for *general* instances, by using the Lagrangian relaxation technique.

3.2 A $(2 + \varepsilon)$ -Approximation Algorithm

In the following we derive a $(2 + \varepsilon)$ -approximation for discrete instances of BRS. A similar result can be obtained for the continuous case, by discretizing the instance. Recall that in the discrete case, any job J_j can be scheduled in the intervals $I_{j,1}, \dots, I_{j,n_j}$. We define a variable $x(j, \ell)$ for each interval $I_{j,\ell}$, $1 \leq j \leq n, 1 \leq \ell \leq n_j$. Then the integer program for the problem is:

$$\begin{aligned}
 (BRS) \quad & \text{maximize} && \sum_{j=1}^n \sum_{\ell=1}^{n_j} w_j x(j, \ell) \\
 & \text{subject to:} && \forall j: \sum_{\ell=1}^{n_j} x(j, \ell) \leq 1 \\
 & && \forall t: \sum_{j \in I_{j,t}} x(j, \ell) \leq 1 \\
 & && \sum_{j=1}^n \sum_{\ell=1}^{n_j} c_j x(j, \ell) \leq B.
 \end{aligned}$$

In the linear program $x_j \in [0, 1]$. Taking the Lagrangian relaxation, we get an instance of the *throughput maximization* problem. As shown in [BB⁺00], an algorithm based on the local ratio technique yields a 2-approximation for this problem, in $O(n^2)$ steps. This algorithm has a primal-dual interpretation; thus, we can apply the technique in Section 2 to obtain an algorithm, $\mathcal{A}$, which uses the algorithm for throughput maximization as a procedure.⁶

Theorem 4. Algorithm $\mathcal{A}$ yields a $(2 + \varepsilon)$ -approximation for BRS, in $O(n^2 \lg^2(n/\varepsilon^2))$ steps.

4 Approximation Algorithms for RSO and BRSO

In this section, we present approximation algorithms for the RSO and BRSO problems. In Section 4.1 we consider RSO. We first give a randomized $e/(e-1)$ -approximation algorithm for discrete inputs; then, we describe a greedy algorithm that achieves a ratio of $(2 - \varepsilon)$ for continuous inputs, and $(3 - \varepsilon)$ for discrete inputs. In Section 4.2, we show that the greedy algorithm can be interpreted equivalently as a primal-dual algorithm. This allows us to apply the Lagrangian relaxation framework (Section 2) and to achieve a $(3 + \varepsilon)$ -approximation for BRSO in the discrete case, where all the intervals corresponding to a job have the same length. For continuous inputs we obtain a $(2 + \varepsilon)$ -approximation.

⁶ Note that since $W_1, W_2 \geq \max_j w_j$, in our search for λ_1, λ_2 we can take $\varepsilon' = \varepsilon/n$.

4.1 The *RSO* Problem

In the *RSO* problem, we may select *all* the jobs, and the problem reduces to scheduling the jobs optimally so as to maximize the coverage of the line. Clearly, when $\forall j, p_j = d_j - r_j$, i.e., each job has only one possible interval, the schedule in which all the jobs are selected is optimal. When $\forall j, p_j \leq d_j - r_j$, the problem becomes hard to solve (see in [NST-03]).

Theorem 5. *The *RSO* problem is strongly NP-hard.*

A Randomized $e/(e-1)$ -approximation algorithm. We start with a linear programming formulation of *RSO*. Assume that the input is given in a discrete fashion, and let $b_0, \dots, b_m$ denote the set of start and end points (in sorted order), called *breakpoints*, of the time intervals $I_{j,\ell}$, $j = 1, \dots, n$, $\ell = 1, \dots, n_j$. We have a variable $x(j, \ell)$ for each interval $I_{j,\ell}$. For any pair of consecutive breakpoints b_{i-1} and b_i , the objective function gains $(b_i - b_{i-1})$ times the “coverage” of the interval $[b_{i-1}, b_i]$. Note that we take the minimum between 1 and the total cover, since we gain nothing if some interval is covered by more than one job.

$$\begin{aligned}
 (L - RSO) \quad & \text{maximize} \quad \sum_{i=1}^m \min \left(\sum_{j=1}^n \sum_{\ell \in [b_{i-1}, b_i]} x(j, \ell), 1 \right) \cdot (b_i - b_{i-1}) \\
 & \text{subject to :} \quad \text{For all jobs } J_j: \sum_{\ell=1}^{n_j} x(j, \ell) \leq 1 \\
 & \quad \text{For all } (j, \ell), \ell = 1, \dots, n_j: x(j, \ell) \geq 0.
 \end{aligned}$$

We compute an optimal (fractional) solution to $(L - RSO)$. Clearly, the value of this solution is an upper bound on the value of an optimal integral solution. To obtain an integral solution, we apply randomized rounding to the optimal fractional solution. That is, for every job J_j , the probability that J_j is assigned to interval $I_{j,\ell}$ is equal to $x(j, \ell)$. If $\sum_{\ell=1}^{n_j} x(j, \ell) < 1$, then with probability $1 - \sum_{\ell=1}^{n_j} x(j, \ell)$ job J_j is not assigned to any interval.

We now analyze the randomized rounding procedure. Consider two consecutive breakpoints b and b' . Define for each job J_j , $y_j = \sum_{I_{j,\ell} \ni [b, b']} x(j, \ell)$. Clearly, $\sum_{j=1}^n y_j = \sum_{j=1}^n \sum_{I_{j,\ell} \ni [b, b']} x(j, \ell)$. W.l.o.g., since each job J_j is assigned to a single interval, we can think of all the intervals of J_j that cover $[b, b']$ as a single (virtual) interval that is chosen with probability y_j . The probability that none of the virtual intervals is chosen is $P_0 = \prod_{j=1}^n (1 - y_j)$. Let $r = \min(\sum_{j=1}^n y_j, 1)$. Then,

$$P_0 \leq \prod_{j=1}^n \left(1 - \frac{\sum_{i=1}^n y_i}{n} \right) = \left(1 - \frac{\sum_{i=1}^n y_i}{n} \right)^n < e^{-\sum_{i=1}^n y_i} \leq e^{-r}.$$

Hence, the probability that $[b, b']$ is covered is $1 - P_0 \geq 1 - e^{-r} \geq (1 - \frac{1}{e}) \cdot r \geq (1 - \frac{1}{e}) \cdot \min\left(\sum_{j=1}^n y_j, 1\right)$. Therefore, the expected contribution to the objective function of any interval $[b_{i-1}, b_i]$ is $(1 - 1/e) \cdot \min(\sum_{j=1}^n y_j, 1) \cdot (b_i - b_{i-1})$. By linearity of expectation, the expected value of the objective function after applying randomized rounding is

$$\left(1 - \frac{1}{e}\right) \cdot \sum_{i=1}^m \min\left(\sum_{j=1}^n \sum_{I_{j,\ell} \ni [b_{i-1}, b_i]} x(j, \ell), 1\right) \cdot (b_i - b_{i-1}),$$

yielding an approximation factor of $1 - 1/e \approx 0.63212$.

A Greedy Approximation Algorithm. We now describe a greedy algorithm, which yields a $(2 - \varepsilon)$ -approximation for continuous instances of RSO, and $(3 - \varepsilon)$ -approximation for discrete instances. Assume that $\min_j r_j = 0$, and let $T = \max_j d_j$. Let I be the set of all the jobs in the instance; U is the set of unscheduled jobs. Denote by s_j, e_j the start-time and completion time of the job J_j in the greedy schedule, respectively. Given a partial schedule, we say that J_ℓ is *redundant* if we can remove J_ℓ from the schedule without decreasing the machine utilization. Algorithm Greedy proceeds in the interval $[0, T]$. At time t , we select an arbitrary job, among the jobs J_i with $r_i < t$ and $d_i > t$. We schedule this job such that its contribution to the utilization, starting at time t , is maximized. The following is a pseudocode of the algorithm.

1. $U = I$, $t = 0$;
2. Let $J_j \in U$ be a job having $d_j > t$ and $r_j \leq t$.
 Schedule J_j such that its completion time, e_j , is $\min(t + p_j, d_j)$.
 Remove J_j from U .
 For any redundant job J_ℓ , omit J_ℓ from the schedule and return it to U .
3. Let $F \subseteq U$ be the set of unscheduled jobs, i , having $d_i > e_j$.
 Let $t_F = \min_{J_i \in F} r_i$, and let $t = \max(e_j, t_F)$.
 If $F \neq \emptyset$ and $t < T$ go to step 2.

We use in the analysis the following properties of the greedy schedule.

Property 1. Once an interval $[x_1, x_2] \in [0, T]$ is covered by Greedy, it remains covered until the end of the schedule.

Property 2. When the algorithm considers time t , some job will be selected and scheduled such that for some $\varepsilon > 0$, the machine is utilized in the interval $[t, t + \varepsilon]$.

Property 3. Consider the set U of non-scheduled jobs at the end of the schedule. For any $J_j \in U$ the machine is utilized in the time interval $[r_j, d_j]$.

Theorem 6. *The Greedy algorithm yields a $(2 - \varepsilon)$ -approximation for the RSO problem.*

Proof. Let $S = I \setminus U$ denote the set of jobs scheduled by Greedy, and let $O \subseteq S$ denote the set of scheduled job such that $J_j \in O$ iff J_j overlaps with another scheduled job, J_k , and $e_j > e_k$. (i) By Property 3, for any $J_j \in U$ the machine is utilized in the time interval $[r_j, d_j]$. (ii) For any $J_j \in S$ the machine is utilized in the time interval $[r_j, s_j]$, otherwise Greedy would have scheduled it earlier. (iii) For any $J_j \in O$ the machine is utilized in the time interval $[r_j, d_j]$. This follows from (ii) and from the fact that $e_j = d_j$, otherwise J_j would not overlap with a job with an earlier completion time.

Given the schedule of Greedy, we allow OPT to add jobs of U and to shift the scheduled jobs of S in any way that increases the utilization. Consider the three disjoint sets of jobs, $U, O, S \setminus O$. By the above discussion, the utilization can be increased only by shifting to the left (i.e., scheduling earlier) the jobs of $S \setminus O$. Note that in any time $t \in [0, T]$ there is at most one job, J_j , of $S \setminus O$ (if two or more jobs overlap, then only the one with the earliest completion time is in $S \setminus O$). Let $0 < \varepsilon_j \leq 1$ be such that J_j overlaps in the Greedy schedule in $(1 - \varepsilon_j)$ of its length. Then OPT can shift J_j into an interval in which it does not overlap at all. Hence, OPT can increase the amount of time the machine is utilized in the greedy schedule at most by factor of $(2 - \varepsilon)$. $\square$

The analysis for discrete inputs is similar, only that in this case, by selecting for two jobs overlapping at time t different intervals, and by adding a non-scheduled job to run at t , OPT may triple the utilization obtained by Greedy, thus we get the bound of $(3 - \varepsilon)$.

4.2 The BRSO Problem

As BRSO generalizes the RSO problem, Theorem 5 implies that it is strongly NP-hard. For discrete inputs we show the following (see in [NST-03]).

Theorem 7. *The discrete BRSO is APX-hard, already for instances where $\forall j, n_j \leq k$ (JISP_k), for any $k \geq 3$.*

A Primal-Dual Algorithm. We first present a primal-dual algorithm for RSO, and show that an execution of the Greedy algorithm given in Section 4.1 can be equivalently interpreted as an execution of the primal-dual algorithm. Thus, the primal-dual algorithm finds a 3-approximate solution to RSO. The primal LP is equivalent to $L - RSO$, given in Section 4.1. We have a variable $x(j, \ell)$ for each interval $I_{j, \ell}$, and a variable z_i , $i = 1, \dots, m$ for each interval $[b_{i-1}, b_i]$ defined by consecutive breakpoints. In the dual LP we have a variable y_j for each job J_j , and two variables, p_i and q_i for each interval $[b_{i-1}, b_i]$ defined by consecutive breakpoints.

$$\begin{array}{ll}
(L - RSO - Primal) & \text{maximize} \quad \sum_{i=1}^m z_i \cdot (b_i - b_{i-1}) \\
\text{subject to :} & \text{For all jobs } J_j: \sum_{\ell=1}^{n_j} x(j, \ell) \leq 1 \\
& \text{For all } i = 1, \dots, m : z_i \leq 1 \\
& \text{For all } i = 1, \dots, m : z_i - \sum_{I_{j,\ell} \ni [b_{i-1}, b_i]} x(j, \ell) \leq 0 \\
& \text{For all } j, \ell, i: \quad x(j, \ell), z_i \geq 0.
\end{array}$$

$$\begin{array}{ll}
(L - RSO - Dual) & \text{minimize} \quad \sum_{j=1}^n y_j + \sum_{i=1}^m p_i \\
\text{subject to :} & \text{For all } (j, \ell), \ell = 1, \dots, n_j: y_j - \sum_{I_{j,\ell} \ni [b_{i-1}, b_i]} q_i \geq 0 \\
& \text{For all } i = 1, \dots, m: \quad p_i + q_i \geq (b_i - b_{i-1}) \\
& \text{For all } j, i: \quad y_j, p_i, q_i \geq 0.
\end{array}$$

Given an integral solution for $L - RSO - Primal$, we say that an interval I belongs to it if there is a job that is assigned to I . An integral solution for $L - RSO - Primal$ is *maximal*, if it cannot be extended and if no interval belonging to it is contained in the union of other intervals belonging to it.

Lemma 1. *Any maximal integral solution (x, z) to $L - RSO - Primal$ is a 3-approximate solution.*

Proof. If $[b_{i-1}, b_i]$ is covered by (x, z) , then set $p_i = b_i - b_{i-1}$, otherwise set $q_i = b_i - b_{i-1}$. Clearly, this defines a feasible dual solution in which $\sum_{i=1}^m p_i = \sum_{i=1}^m z_i \cdot (b_i - b_{i-1})$. Thus, it remains to bound $\sum_{j=1}^n y_j$ in this solution.

For each job J_j that is not assigned to any interval in (x, z) , i.e., its intervals are contained in intervals of other jobs, we can set $y_j = 0$. Suppose that for job J_j , $x(j, \ell) = 1$. Consider, for example, an interval $I_{j,\ell'}$, $\ell \neq \ell'$, that contains two consecutive breakpoints b_{i-1} and b_i such that $[b_{i-1}, b_i]$ is not covered by any job. In this case $q_i = b_i - b_{i-1}$ and $y_j \geq q_i$. Thus, in order to bound $\sum_{j=1}^n y_j$, we say that the values of q_i -s that determine the y_j -s “charge” the p_i values corresponding to the breakpoints covered by $I_{j,\ell}$. This can be done since all the intervals in which J_j can be scheduled have the same length. Since our primal solution is maximal, any point is covered by at most two intervals to which jobs are assigned, and therefore any variable p_i can be “charged” by intervals belonging to at most two different jobs. Thus, $\sum_j y_j \leq 2 \sum_i p_i$, proving that $\sum_{i=1}^m p_i = \sum_{i=1}^m z_i \cdot (b_i - b_{i-1}) \leq \sum_{j=1}^n y_j + \sum_{i=1}^m p_i \leq 3 \cdot \sum_{i=1}^m p_i$, meaning that (x, z) is a 3-approximate solution. $\square$

For continuous instances, we can discretize time such that each time slot is of size ϵ . This will incur a $(1 + \epsilon')$ degradation in the objective function where $\epsilon' = \text{poly}(\epsilon, n)$. We can show that for a discrete input obtained from a discretization of a continuous input instance, the primal-dual algorithm yields a 2-approximate solution. Applying the Lagrange relaxation technique (presented in Section 2), we get the following.

Theorem 8. *BRSO can be approximated within factor $(2 + \epsilon)$ in the continuous case, and $(3 + \epsilon)$ in the discrete case, in $O(n^2 \cdot \lg^2(n/\epsilon^2))$ steps.*

4.3 An FPTAS for $JISP_1$

For instances of *BRSO* where $p_j = d_j - r_j$ ($JISP_1$), we use a reduction to the *budgeted longest path problem* in acyclic graph, to obtain an optimal polynomial time algorithm for unit costs, and an FPTAS for general instances. In the budgeted longest path problem, we are given an acyclic graph, $G(V, E)$; each edge $e \in E$ has a length $\ell(e)$, and a cost $c(e)$. Our goal is to find the longest path in G connecting two given vertices s, t , whose price is bounded by a given budget B . The problem is polynomially solvable for unit edge costs, and has an FPTAS for arbitrary costs [Ha-92].

Given an instance of *BRSO* where $\forall j, p_j = d_j - r_j$, we construct the following graph, G . Each job j is represented by a vertex; there is an edge $e = (i, j)$ iff $d_i < d_j$ and $r_i \leq r_j$. The length of the edge is $\ell(e) = \min(d_j - d_i, p_j)$, and its cost is c_j . Note that $\ell(e)$ reflects the machine utilization gained if the deadlines of J_i, J_j are adjacent to each other in the schedule. In addition, each vertex j is connected to the source, s , where $\ell(s, j) = p_j$, $c(s, j) = c_j$, and to a sink t , where $\ell(j, t) = 0$ and $c(s, j) = 0$.

Theorem 9. *There is a schedule achieving utilization of u time units and having cost $b \leq B$ if and only if G contains a path of length u and price b .*

Proof. For a given schedule, sort the jobs in the schedule such that $d_{j_1} \leq d_{j_2} \leq \dots \leq d_{j_w}$. W.l.o.g, the schedule does not include two jobs J_i, J_j such that $r_j < r_i$ and $d_i < d_j$, since in such a schedule we gain nothing from processing J_i . Thus, we can assume that $r_{j_i} \leq r_{j_{i+1}}, \forall 1 \leq i \leq w$. This implies that the graph G contains the path $s, j_1, j_2, \dots, j_w, t$ (the first and last edges in this path exist by the structure of G). Suppose that the utilization of the schedule is u and its cost is b . We show that the length of the corresponding path in G is u and its cost is b . Recall that the edge (j_w, t) has length 0 and costs nothing, thus, we consider only the first w edges in this path. The utilization we gain from scheduling j_i is p_{j_i} if $i = 1$ and $\min(p_{j_i}, d_{j_i} - d_{j_{i-1}})$ if $1 < i \leq w$. This is exactly $\ell(j_{i-1}, j_i)$ (or $\ell(s, j_1)$ for the first vertex in the path). Also, the cost of the schedule is the total processing cost of the scheduled jobs, which is identical to the total cost of edges in the path.

For a given directed path in G , we schedule all the jobs whose corresponding vertices appear on the path. Note that the price of the path consists of the price

of the participating vertices, thus, b is also the price of the schedule. Also, as discussed above $\ell(i, j)$ reflects the contribution of the corresponding job to the utilization, thus the path induces a schedule with the correct utilization and cost. $\square$

5 Multiple Machines

Suppose that we have m machines, and a budget B , which can be distributed in any way among the machines. It can be shown that this model is equivalent to the single machine case, by concatenating the schedules on the m machines to a single schedule in the interval $[0, mT]$, on a single machine. Thus, all of our results carry over to this model. When we have a budget specified for each machine, we show that any approximation algorithm $\mathcal{A}$ for a single machine can be run iteratively on the machines and the remaining jobs, to obtain a similar approximation ratio. Denote this algorithm $\mathcal{A}^*$.

Theorem 10. *If $\mathcal{A}$ is an r -approximation then the iterative algorithm $\mathcal{A}^*$ is an $(r + 1)$ -approximation.*

Note that in most cases $\mathcal{A}^*$ performs better. For example, when we iterate **Greedy** (Section 4.1) for the *RSO* problem, it can be seen that the proof for a single machine is valid also for multiple machines, thus **Greedy** is $(2 - \varepsilon)$ -approximation. In the full version of the paper, we show that our results can be extended to apply also for the case where the processing costs of the jobs are machine dependent, that is, the cost of processing J_j on the k -th machine is c_{jk} , $1 \leq j \leq n$, $1 \leq k \leq m$.

Acknowledgments. We thank Shmuel Zaks for encouraging us to work on *RSO* and its variants. We also thank Magnús Halldórsson and Baruch Schieber for valuable discussions.

References

- [AS-87] E.M. Arkin and E.B. Sliverberg. "Scheduling jobs with fixed start and end times". *Discrete Applied Math.*, 18:1–8, 1987.
- [B-99] 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.
- [BB⁺00] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. of the ACM*, 48:1069–1090, 2001.
- [BG⁺99] A. Bar-Noy, S. Guha, J. Naor and B. Schieber Approximating the throughput of real-time multiple machine scheduling. *SIAM J. on Computing*, 31:331–352, 2001.
- [BD-00] P. Berman and B. DasGupta. "Multi-phase algorithms for throughput maximization for real-time scheduling." *J. of Combinatorial Optimization*, 4:307–323, 2000.

- [COR-01] J. Chuzhoy, R. Ostrovsky, and Y. Rabani. "Approximation algorithms for the job interval selection problem and related scheduling problems." *FOCS*, 2001.
- [GJ-79] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, 1979.
- [Ga-96] N. Garg, A 3-approximation for the minimum tree spanning k vertices, Proceedings of FOCS, 1996.
- [Ha-92] R. Hassin, Approximation schemes for the restricted shortest path problem. In *Mathematics of Operations Research*, 17(1): 36–42, 1992.
- [K-91] V. Kann. Maximum bounded 3-dimensional matching is max SNP-complete. *Information Processing Letters*, 37:27–35, 1991.
- [KN-01] Y. Karuno and H. Nagamochi, A 2-approximation algorithm for the multi-vehicle scheduling problem on a path with release and handling times. *ESA*, 2001.
- [KMN-99] S. Khuller, A. Moss, J. Naor, The budgeted maximum coverage problem. *Information Processing Letters* 70(1): 39–45, 1999.
- [LO-02] D. H. Lorenz and A. Orda, Optimal partition of QoS requirements on unicast paths and multicast trees, *IEEE/ACM Trans. on Networking*, 10(1), 102–114, 2002.
- [LORS-00] D. H. Lorenz, A. Orda, D. Raz, Y. Shavitt, Efficient QoS partition and routing of unicast and multicast, *8th Int. Workshop on Quality of Service*, Pittsburgh, 2000.
- [NST-03] J. Naor, H. Shachnai and T. Tamir Real-time Scheduling with a Budget, <http://www.cs.technion.ac.il/~hadas/PUB/rtbudget.ps>.
- [S-99] F. C. R. Spieksma. "On the approximability of an interval scheduling problem." *J. of Scheduling*, 2:215–227, 1999.
- [T-00] M. Tennenholtz. Some tractable combinatorial auctions. *AAAI/IAAI*, 98–103, 2000.
- [Va-01] V. Vazirani, *Approximation algorithms*, Springer Verlag, 2001.

Improved Approximation Algorithms for Minimum-Space Advertisement Scheduling*

Brian C. Dean and Michel X. Goemans

M.I.T., Cambridge, MA 02139, USA,
bdean@theory.lcs.mit.edu, goemans@math.mit.edu

Abstract. We study a scheduling problem involving the optimal placement of advertisement images in a shared space over time. The problem is a generalization of the classical scheduling problem $P||C_{max}$, and involves scheduling each job on a specified number of parallel machines (not necessarily simultaneously) with a goal of minimizing the makespan. In 1969 Graham showed that processing jobs in decreasing order of size, assigning each to the currently-least-loaded machine, yields a $4/3$ -approximation for $P||C_{max}$. Our main result is a proof that the natural generalization of Graham's algorithm also yields a $4/3$ -approximation to the minimum-space advertisement scheduling problem. Previously, this algorithm was only known to give an approximation ratio of 2, and the best known approximation ratio for any algorithm for the minimum-space ad scheduling problem was $3/2$. Our proof requires a number of new structural insights, which leads to a new lower bound for the problem and a non-trivial linear programming relaxation. We also provide a pseudo-polynomial approximation scheme for the problem (polynomial in the size of the problem and the number of machines).

1 Introduction

We study a scheduling problem whose application is the optimal placement of advertisement images within a shared space over time, typically on a web page. Roughly \$3 billion was spent on web advertising in the first half of 2002 alone [7], so improvements in algorithms for ad placement are of both economic and theoretical interest. In the model we consider here, we have a set $\mathcal{A} = [n] := \{1, \dots, n\}$ of n ads (images of fixed width and varying heights) to schedule within a shared vertical space, typically in the margin of a web page. We must determine the subset of ads to display in each of T occurrences of the page, or T time slots. Ad i has a height h_i and a display count $c_i \leq T$ which represents the number of time slots out of T in which the ad must appear. The goal is to assign each ad i to a set of c_i distinct time slots so as to minimize the maximum height of any occurrence of the page, as illustrated graphically in Figure 1. Mathematically, this means that we need to find $A_i \subseteq [n]$ for $i \in [n]$ such that (i) $|A_i| = c_i$ for every i and (ii) $\max_{t \in [T]} \sum_{i: t \in A_i} h_i$ is minimized. This problem was first posed

* This work was supported by NSF contracts CCR-0098018 and ITR-0121495.

in 1998 by Adler, Gibbons, and Matias [1]. Notice that we do not care about the vertical ordering of the ads within a single time slot.

When $c_i = 1$ for every i , the problem reduces to the classical NP-Hard scheduling problem $P||C_{max}$ with the time slots corresponding to machines, the ads corresponding to jobs and the height h_i corresponding to the processing time p_i . In fact, our ad scheduling problem is very similar to $P||C_{max}$ with *high-multiplicity encoding* of jobs with the same processing time, except that we require that these additional copies be scheduled on different machines. The high-multiplicity encoding of $P||C_{max}$, denoted by $P|M|C_{max}$ (see Clifford and Posner [2]) has been the focus of some study in the 90's, but there are still many open questions surrounding this problem, in particular whether or not the complexity of an optimal solution is of size polynomial in the input. See McCormick et al. [9] and Clifford and Posner [2] for further discussion.

For $P||C_{max}$, Graham [4] shows that processing the jobs in any order and greedily placing each job in sequence on the currently-least-loaded machine is a 2-approximation algorithm and that this is tight. He further shows that greedy processing of jobs with largest processing time first (LPT) is a $4/3$ -approximation algorithm and this is also tight. For the ad scheduling problem, Adler, Gibbons, and Matias [1] introduce the analogue of LPT, the *Largest Size Least Full (LSLF)* algorithm which processes ads in non-increasing order of height and greedily schedules ad i on the c_i currently-least-loaded time slots, and show that it is a 2-approximation algorithm, leaving open the question whether a better approximation factor could be proved. Subsequently, Dawande, Kumar, and Sriskandarajah [3] show that any list processing of ads in a greedy fashion is a 2-approximation algorithm, and that this is tight. Dawande et al. also show that rounding the optimum solution of a trivial linear programming relaxation of the problem leads to a 2-approximation algorithm and that a more sophisticated relaxation can be used to obtain a $3/2$ -approximation algorithm. Previous to this paper, these were the best known approximation algorithms for the ad scheduling problem. One of the main results of this paper is to show that LSLF is a $4/3$ -approximation algorithm, thereby matching Graham's bound for $P||C_{max}$ (this is tight, due Graham's analysis). Our proof is, however, much more elaborate than for the case of $P||C_{max}$.

Theorem 1. *LSLF is a $4/3$ -approximation algorithm for the ad scheduling problem.*

Regarding approximation schemes for $P||C_{max}$, for case when the number of machines is a constant Horowitz and Sahni [6] gave a $(1 + \epsilon)$ -approximation algorithm for any $\epsilon > 0$ (PTAS). This was improved by Hochbaum and Shmoys [5] to all instances of $P||C_{max}$. Since the running time of this algorithm depends polynomially on the number of machines, it is only a pseudopolynomial approximation scheme for $P|M|C_{max}$. We introduce a similar pseudopolynomial approximation scheme for the ad scheduling problem, whose running time is polynomial in n and T .

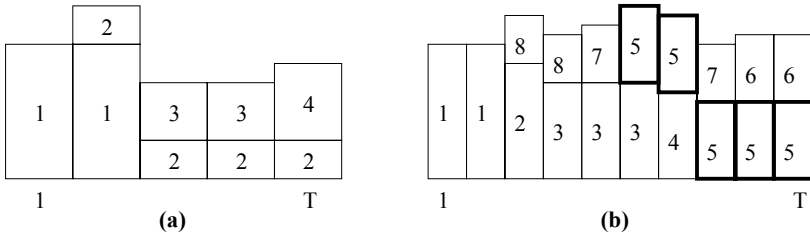


Fig. 1. Sample schedules. The vertical axis represents vertical space and each time slot corresponds to a column along the horizontal axis. (a) LSLF schedule for which Graham's proof fails, (b) Illustration of wrap-around behavior of LSLF when scheduling large ads.

Discussion of Our Results. Graham's proof for the performance guarantee of $4/3$ for LPT differentiates *large* jobs with processing times greater than $OPT/3$ (OPT denotes the makespan of an optimal schedule), from the remaining *small* jobs. He observes the following: (i) if the makespan of the LPT schedule is defined by a machine containing only large jobs, then the LPT schedule will in fact be optimal; (ii) on the other hand, if the makespan of the LPT schedule is defined by a machine whose final job j is small, then the makespan can be written as $L + p_j$, where L denotes the load of that machine prior to j 's scheduling. The $4/3$ bound then follows since $p_j \leq OPT/3$ and since $L \leq OPT$ because greedy scheduling ensures that every other machine must have a load of at least L .

We will show that (i) also holds for the ad scheduling problem: if a maximum-height slot in the LSLF schedule contains only large ads (having heights greater than $OPT/3$), then the LSLF schedule is optimal. However, in (ii) the assumption that $L \leq OPT$ no longer holds for the ad scheduling case, as shown in Figure 1(a).

Regarding (i), we will prove in Theorem 2 the following stronger result: if all ads are large, then LSLF not only minimizes the height of the largest time slot, but it also minimizes the sum of the heights of the largest k time slots for any k over all schedules that place at most 2 ads per time slot (and this is the case indeed for the optimum schedule). To the best of our knowledge, this structural result was not even known for $P||C_{max}$. We use this to devise a stronger lower bound based on the following intuition: consider a set P_k comprising k time slots into which LSLF places the greatest amount of large ad volume. By our structural result, the total large ad volume placed in P_k by LSLF is a lower bound on the large ad volume found in P_k in an optimal schedule. Further, we can lower bound the amount of small ad volume that must appear in P_k by noting that for each small ad i , we must schedule at least $\max(c_i - (T - k), 0)$ copies of i in P_k . Therefore, if we are able to locate such a set P_k for which (i) every small ad i is scheduled by LSLF in exactly $\max(c_i - (T - k), 0)$ slots in P_k , and (ii) the maximum and minimum slot heights in P_k differ by at most $OPT/3$, then the $4/3$ approximation bound would follow. Unfortunately, there are problem

instances in which no such set P_k satisfies these conditions. In these cases, we manage to prove that there exists a subset $\mathcal{A}'$ of our set $\mathcal{A}$ of ads and a value k such that (i) the values of LSLF on $\mathcal{A}$ and $\mathcal{A}'$ are equal, $LSLF(\mathcal{A}) = LSLF(\mathcal{A}')$, and (ii) the above property holds for $\mathcal{A}'$ and k .

We will also present a different $4/3$ -approximation algorithm based on rounding a new linear programming relaxation built on our structural results above. We will show an efficient combinatorial technique for solving this LP, and that its solution is exactly the closed-form expression of our aforementioned lower bound.

Finally, we describe a PTAS based on a combination of dynamic programming and LP rounding, which is similar to other approximation schemes for $P||C_{max}$. Its running time is polynomial in n and T , and therefore only pseudopolynomial. Our two $4/3$ -approximation algorithms, in contrast, can be implemented to run in time polynomial in n with no dependence on T ; details are left for the full version. For that purpose, the output is condensed by listing every distinct assignment of ads to a time slot along with the number of time slots with this assignment.

The extended abstract is structured as follows. In the next section, we study the LSLF schedule and shows that it schedules the large ads extremely well. We also deduce our new lower bound. In section 3, we prove that LSLF produces a schedule within a factor of $4/3$ of the optimum. In Section 4, we introduce our linear programming relaxation, show that its value is equal to our lower bound and derive rounding-based $4/3$ -approximation algorithms. Finally, in Section 5, we sketch a pseudo-polynomial approximation scheme for the problem.

2 Large Ads in the LSLF Schedule

We declare ad i to be *large* if $h_i > OPT/3$, and *small* if $h_i \leq OPT/3$. Observe that the total number of large ad copies in our problem instance cannot exceed $2T$ since any optimal solution must contain no more than two large ads per slot.

Lemma 1. *The LSLF algorithm places at most two large ads in every slot.*

Proof. Consider a division of the large ads into two sets: the set of *huge* ads containing ads i for which $h_i > 2OPT/3$, and the remaining set which we call the *medium* ads. LSLF first schedules the huge ads, one copy per slot. Let P denote the set of slots occupied by these ads, and Q denote the remaining slots. LSLF then schedules the medium ads one by one. We claim by induction that the following two invariants hold during this process: (i) all medium ads are placed solely within slots in Q , and (ii) at most two medium ads are placed in each slot in Q .

Suppose our two invariants hold up to the point where LSLF schedules a particular medium ad i . We must have $c_i \leq |Q|$, otherwise this ad would share a slot with some huge ad in every optimal solution, which clearly cannot happen. Therefore, in principle we have sufficient room to fit all copies of ad i within only the slots in Q . We now claim that there must be c_i slots in Q which currently

contain at most one medium ad. If this were not the case, then the total number of medium ads would exceed $2|Q|$, and this would imply that in every optimal solution either (i) a huge and medium ad must share a slot, or (ii) there must be a slot containing more than two medium ads; both of these situations are clearly impossible. All slots in Q having at most one medium ad are currently shorter than slots in Q having two medium ads and also shorter than slots in P , so LSLF selects these slots for the placement of ad i . This maintains both of our invariants.

Not only does LSLF place at most two large ads in any slot, but it arranges these ads according to a very specific structure. LSLF schedules one row of large ads and then wraps around to schedule another row of large ads in reverse. The wrap-around point is a bit delicate though, as shown in Figure 1(b), because we must avoid placing two copies of the same ad in a common slot. One can argue that the LSLF schedule for the large ads can be constructed in the following simple recursive way. (i) If an ad exists with multiplicity T , then we schedule this ad in every slot and then any remaining large ad one copy per slot; (ii) if there are $2T$ large ad copies, we pair up a copy of one of the maximum-height large ads with a copy of one of the minimum-height large ads, decrease T , update the multiplicities and then recurse; (iii) if there are fewer than $2T$ large ad copies, we place a copy of a maximum-height large ad by itself and recurse.

We refer to the part of a schedule consisting of only large ads as the *base* of the schedule. We have just described a simple recursive way to construct the base of the LSLF schedule. For any schedule S , let $b_1(S) \geq b_2(S) \geq \dots \geq b_T(S)$ denote the height occupied by the base, namely the large ads, in every time slot, ordered in a non-increasing fashion. Also, for $k = 1, \dots, T$, define $B_k(S) = \sum_{i=1}^k b_i(S)$. We claim that LSLF schedules the large ads in a very balanced way, at least as well as in the optimal schedule. This is formalized in the following theorem.

Theorem 2. *Let LSLF denote the schedule produced by the LSLF algorithm, and let Σ be an optimum schedule. Then for all values of k , $B_k(\text{LSLF}) \leq B_k(\Sigma)$.*

Proof. The proof is by contradiction. Suppose we have a schedule S generated by LSLF, an optimal schedule Σ , and a value k for which the lemma fails. We will show how to transform the base of Σ into the base of S without increasing $B_k(\Sigma)$. Based on our recursive characterization of how LSLF schedules the large ads, we will make the bases of S and Σ agree in one slot — for example by making an exchange in Σ so a maximum-height large ad is paired with a minimum-height large ad. We will then recursively transform the remaining $T - 1$ slots of the base of Σ into the remaining $T - 1$ slots of the base of S . Our transformation must consider these three cases:

1. If a large ad i exists for which $c_i = T$, then LSLF will schedule this ad in every slot, and the remaining large ads one copy per slot. In Σ , we will also find ad i scheduled once in every slot, and every other large ad scheduled one copy per slot, since the optimal schedule cannot have more than two large ads per slot. Therefore, if a large ad of multiplicity T exists, then the

base of Σ will be the same as the base of S . We can terminate our recursive transformation once we reach this case.

2. Suppose that $c_i < T$ for every large ad i and that there are exactly $2T$ large ad copies. If there exists a slot in Σ in which a maximum-height large ad is paired with a minimum-height large ad, then we can hold this slot fixed, decrement T and the display counts of the two ads in this slot, and continue our recursive transformation on the remaining slots. Otherwise, suppose we can locate distinct large ads i and j such that a maximum-height large ad is paired with a copy of i and a minimum-height large ad is paired with a copy of j . Swapping ad i and the minimum-height large ad with each other will change the pairing from (Max, i) , (Min, j) to (Max, Min) , (i, j) , and will not decrease $B_k(\Sigma)$ for any k . We may then continue our recursive transformation. If none of the above cases apply, then every instance of a maximum-height or minimum-height large ad must be paired with the same large ad l . However, since $c_l < T$ there must be some slot in which two large ads i and j different from l are paired. Assume $h_i \leq h_j$. If $h_i < h_l$, then we make a swap to change the pairing (Max, l) , (i, j) to (Max, i) , (l, j) . If $h_i \geq h_l$, we make a swap to change (Min, l) , (i, j) into (Min, j) , (i, l) . This exchange also will not decrease $B_k(\Sigma)$ for any k , and will place us in the prior situation where one more swap suffices to pair up the minimum-height and maximum-height large ads.
3. Suppose that $c_i < T$ for every large ad i and that there are fewer than $2T$ large ad copies. If there exists a slot in the base of Σ in which the only large ad is a maximum-height large ad, then we hold this slot fixed, decrement T and the display count of the ad in this slot, and continue our recursive transformation on the remaining slots. Otherwise, we find a slot containing a maximum-height large ad and swap away the remaining large ad in its slot. The details are essentially the same as in case 2.

One can easily prove the following corollaries to Theorem 2.

Corollary 1. *If all ads are large, LSLF produces an optimal schedule.*

Corollary 2. *Consider the schedule produced by LSLF. If a maximum-height slot in this schedule contains only large ads, then this schedule is optimal.*

2.1 A New Lower Bound on OPT

In order to argue approximation bounds we need to find suitable lower bounds on OPT . The maximum ad height, h_{max} , and the average slot height in any schedule, $\sum_i c_i h_i / T$, are both trivial lower bounds on OPT . These lower bounds are sufficient to show that LSLF is a 2-approximation algorithm [3]; however, in order to show that LSLF is a 4/3-approximation algorithm, we introduce a new, more powerful lower bound based on Theorem 2.

Theorem 3. *Let $\mathcal{A}_S$ denote the set of small ads in our problem instance. Then for every k , we have $L_k \leq OPT$, where*

$$L_k = \frac{1}{k} \left(B_k(LSLF) + \sum_{i \in \mathcal{A}_S} h_i \max(c_i - (T - k), 0) \right).$$

Proof. Consider k slots of maximum base height in the optimum schedule Σ . We know from Theorem 2 that the total base height $B_k(\Sigma)$ in these k slots is greater or equal to $B_k(LSLF)$. Now consider any small ad i . The smallest number of copies of i that Σ could possibly assign to these k slots is $\max(c_i - (T - k), 0)$ (since $T - k$ is an upper bound on the number of copies outside of these k slots). Thus the total height in Σ in these k slots is at least $B_k(LSLF) + \sum_{i \in \mathcal{A}_S} h_i \max(c_i - (T - k), 0)$ and the maximum height of any of these slots must be at least the average value L_k .

Observe that the two trivial lower bounds mentioned earlier are dominated by this bound: $h_{max} \leq L_1$ and $\sum_i c_i h_i / T = L_T$.

Although we do not know which ads are large and which are small, we can nevertheless compute a lower bound based on Theorem 3. If we know the index j of the smallest large ad, we can compute the bound $LB(j) = \max_k L_k$ given by the above Theorem. Since we do not know j , compute j^* to be the largest index such that $LB(j^*) \leq 3h_{j^*}$ or $j^* = 0$ if no such index exists, in which case there are no large ads. We claim that $LB(j^*)$ is a lower bound on OPT , and it is at least as good as the one if we knew which were the large ads. Indeed, it is a lower bound since either $h_{j^*} > OPT/3$ in which case j^* is a large ad and $LB(j^*)$ is a lower bound on OPT by Theorem 3, or $h(j^*) \leq OPT/3$ in which case $3h(j^*)$ is a lower bound on OPT and therefore so is $LB(j^*)$. Furthermore, $j^* + 1$ is not the index of a large ad since otherwise $LB(j^* + 1) \leq OPT < 3h_{j^* + 1}$, contradicting the choice of j^* . Thus the unknown index j of the smallest large ad satisfies $j \leq j^*$. Finally, $LB(\cdot)$ can be shown to be non-decreasing (either directly from the formula for L_k given above, or using Theorem 5 in Section 4 and arguing that the way LSLF places job l in $LB(l)$ is feasible for the linear program corresponding to $LB(l - 1)$). The fact that $j \leq j^*$ then implies that $LB(j^*)$ is at least as good a lower bound as $LB(j)$.

3 Analysis of the LSLF Algorithm

Throughout this section, we assume that the time slots are indexed in non-increasing order of base heights: $b_1(LSLF) \geq b_2(LSLF) \geq \dots \geq b_T(LSLF)$. We first need several definitions. If P is a set of slots, we say that a small ad i is P -minimal if exactly $\max(c_i - (T - |P|), 0)$ copies of i are scheduled in P by LSLF. Thus, in the above argument we would like to have that all small ads are P -minimal, where $P = [k]$ is a prefix of the slots. For a time slot t , let $H_i(t)$ denote the height of t immediately after LSLF schedules ad i . For a set of slots P we then define $Min_i(P) = \min\{H_i(t) : t \in P\}$ and

$Max_i(P) = \max\{H_i(t) : t \in P\}$. Finally, we define the *range* of a set of slots as $Range_i(P) = Max_i(P) - Min_i(P)$. For notational simplicity, we will omit the subscript on these quantities when we speak of the final schedule produced by LSLF, so for example $Min(P) = Min_n(P)$.

As alluded to in the introduction, if we could find a value k such that (i) $Range([k]) \leq OPT/3$ and (ii) every small ad is $[k]$ -minimal then it would be easy to show that LSLF produces a schedule of value at most $L_k + OPT/3 \leq \frac{4}{3}OPT$; this is formalized in Lemma 4 below. Unfortunately, such a value k does not always exist, but we will show that we can find a subset of the ads for which the value of LSLF does not change and the above property is satisfied.

In order to guarantee that $Range(P) \leq OPT/3$ for certain sets P , we use the following Lemma and Corollary.

Lemma 2. *Let t and u be slots satisfying $t < u$. If i is a small ad or the last large ad processed, then $H_i(u) \leq H_i(t) + OPT/3$.*

Proof. Since $t < u$, we know that $b_t(LSLF) \geq b_u(LSLF)$, so u will be no taller than t just before scheduling all the small ads. As LSLF schedules small ads, as long as u remains no taller than t the lemma is certainly satisfied. So let us assume that at some point in time we schedule a small ad i in u but not in t , resulting in u becoming taller than t . However, at this point, u will be taller by at most $h_i \leq OPT/3$, so we have $Range_i(\{t, u\}) \leq OPT/3$. We can now observe that $H_i(u) - H_i(t)$ cannot increase as we increment i as long as $H_i(t)$ does not overcome $H_i(u)$. This shows that $Range(\{t, u\}) \leq OPT/3$ at termination.

Corollary 3. *Let k be a maximum-height slot in the final LSLF schedule. If a small ad was scheduled in k , then $Range([k]) \leq OPT/3$.*

We need one more concept. As the algorithm progresses, we will differentiate between *heavy* and *light* time slots with respect to a prefix $P = [k]$ of time slots. We assume that LSLF has already scheduled the large ads; so from now on, we focus on small ads only. At the beginning (so just before scheduling any small ad), we designate all time slots to be heavy; more formally, we say that all slots are $(P, 0)$ -heavy. If we now consider the point in time right after LSLF schedules some (small) ad i , we say that a time slot is t is (P, i) -light if:

- Rule I: Slot t is $(P, i - 1)$ -light.
- Rule II: Ad i is small, not P -minimal, and is scheduled in some slot u for which $H_i(u) \geq H_i(t)$.
- Rule III: Ad i is small, P -minimal, and is scheduled in some $(P, i - 1)$ -light slot u for which $H_i(u) \geq H_i(t)$.

If none of these three conditions apply, then we say slot t is (P, i) -heavy. Let us briefly build some intuition about this definition. As soon as a slot becomes light, it will remain light forever. If a small ad i is not P -minimal, this immediately forces all slots receiving a copy of i to become light. This means that, at the end, all slots are P -heavy if and only if all small ads are P -minimal. Additionally, a

slot becomes light any time we notice that another light slot matches or exceeds it in height, so the heavy slots always dominate the light slots in height. This is formalized by the following lemma.

Lemma 3. *If slot t is (P, i) -heavy and slot u is (P, i) -light, then $H_i(t) > H_i(u)$.*

Proof. Suppose the lemma fails for two slots t and u . Consider the smallest i for which t is (P, i) -heavy, u is (P, i) -light, and $H_i(t) \leq H_i(u)$. We know i is a small ad and that t is $(P, i - 1)$ -heavy by rule I. Consider the following cases:

1. **Slot u is $(P, i - 1)$ -light, and i is not P -minimal.** Since we picked i to be minimal, we know $H_{i-1}(t) > H_{i-1}(u)$, so i must be scheduled in u but not in t . Rule II thus implies that t must be (P, i) -light.
2. **Slot u is $(P, i - 1)$ -light, and i is P -minimal.** Again, since we picked i to be minimal, $H_{i-1}(t) > H_{i-1}(u)$ and i must be scheduled in u but not in t . Rule III thus implies that t must be (P, i) -light.
3. **Slot u is $(P, i - 1)$ -heavy, and i is not P -minimal.** By rule II, we know i is scheduled in some slot v for which $H_i(v) \geq H_i(u)$. However, since $H_i(v) \geq H_i(u) \geq H_i(t)$, rule II implies that t must be (P, i) -light.
4. **Slot u is $(P, i - 1)$ -heavy, and i is P -minimal.** By rule III, we know i is scheduled in some $(P, i - 1)$ -light slot v for which $H_i(v) \geq H_i(u)$. However, since $H_i(v) \geq H_i(u) \geq H_i(t)$, rule III again implies that t must be (P, i) -light.

In all cases, we conclude that t is (P, i) -light, a contradiction.

The following lemma formalizes our informal discussion earlier in this section of a case for which the $4/3$ bound follows easily. As we said earlier, such a prefix of slots does not always exist.

Lemma 4. *If there exists a prefix $P = [k]$ of slots for which $\text{Range}(P) \leq \text{OPT}/3$ and for which all slots are (P, n) -heavy, then $\text{Max}([T]) \leq 4/3\text{OPT}$.*

Proof. Consider the lower bound L_k from Theorem 3. We argue that L_k will be the average slot height within P , since every small ad i must be P -minimal. If any small ad weren't P -minimal, all slots receiving it would have become light. Since the minimum slot height in P is a lower bound on the average slot height in P , we have $\text{Min}(P) \leq L_k$, and since $L_k \leq \text{OPT}$ by Theorem 3, $\text{Min}(P) \leq \text{OPT}$. Finally, since $\text{Range}(P) \leq \text{OPT}/3$, we have $\text{Max}(P) \leq \text{Min}(P) + \text{OPT}/3 \leq 4/3\text{OPT}$.

We are now equipped to give the main result of this paper, a proof of Theorem 1.

Proof. We first describe the core argument at a high level, postponing discussion of a few key technical lemmas. Consider the schedule produced by LSLF on some instance I . Let P be the largest possible prefix of slots one can form such that $\text{Range}(P) \leq \text{OPT}/3$. By Corollary 3, we know that P can at least be made large enough to capture every maximum-height slot. Consider now the following three cases.

1. **All slots are (P, n) -heavy.** In this case, Lemma 4 says that the makespan of the LSLF schedule is at most $4/3OPT$.
2. **All slots are (P, n) -light.** This case is impossible. If all slots are P -light, then Lemma 5 below applied to the last small ad and the slot achieving the maximum height implies that we can extend P to a larger prefix P' for which $Range(P') \leq OPT/3$, thereby contradicting the maximality of P .
3. **Some slots are (P, n) -heavy and some are (P, n) -light.** In this case, we reduce our problem instance I to a strictly smaller instance I' by deleting a carefully chosen subset of the ads. Since I' contains fewer ads, we have $OPT(I') \leq OPT(I)$, and Lemma 6 below shows how to construct I' such that $LSLF(I') = LSLF(I)$. We now claim by induction on the size of our instance that $LSLF(I') \leq 4/3OPT(I')$, so $LSLF(I) = LSLF(I') \leq 4/3OPT(I') \leq 4/3OPT(I)$.

This completes the proof. All that remains is to argue lemmas 5 and 6.

Lemma 5. *If slot t is (P, i) -light where $P = [k]$ then $H_i(t) - H_i(k+1) \leq OPT/3$.*

Proof. We proceed by induction on i and consider 4 different cases.

1. **Ad i is scheduled on t but not on $k+1$.** In this case, we have

$$H_i(t) = H_{i-1}(t) + h_i \leq H_{i-1}(k+1) + h_i \leq H_i(k+1) + h_i \leq H_i(k+1) + OPT/3.$$

2. **We are not in case 1, and t is $(P, i-1)$ -light.** We know that $H_{i-1}(t) - H_{i-1}(k+1) \leq OPT/3$ by induction. Furthermore, since we are not in case 1, we have that $(H_i(t) - H_{i-1}(t)) - (H_i(k+1) - H_{i-1}(k+1)) \leq 0$, which summed up with the previous inequality gives the statement of the lemma.
3. **Slot t is $(P, i-1)$ -heavy and i is P -minimal.** Since t is (P, i) -light, rule III must have applied. This means that i is scheduled on a $(P, i-1)$ -light slot u with $H_i(u) \geq H_i(t)$. Since u must fall into either case 1 or 2 above, we have $H_i(u) - H_i(k+1) \leq OPT/3$. Therefore, $H_i(t) \leq H_i(u) \leq H_i(k+1) + OPT/3$.
4. **Slot t is $(P, i-1)$ -heavy and i is not P -minimal.** So, rule II must have applied and i is scheduled on a slot u with $H_i(u) \geq H_i(t)$. Since i is not P -minimal, there exists $v \geq k+1$ such that i is not scheduled on v . We further consider two cases.
 - a) If v can be chosen to be $k+1$ then

$$H_i(t) \leq H_i(u) = H_{i-1}(u) + h_i \leq H_{i-1}(k+1) + h_i \leq H_i(k+1) + OPT/3.$$

- b) If not, then we can assume that i is scheduled on $k+1$ and then we have

$$\begin{aligned} H_i(t) &\leq H_i(u) = H_{i-1}(u) + h_i \leq H_{i-1}(v) + h_i \\ &\leq H_{i-1}(k+1) + OPT/3 + h_i = H_i(k+1) + OPT/3, \end{aligned}$$

where the last *inequality* follows from Lemma 2 since $v > k+1$.

Lemma 6. *Fix an instance I and a prefix P of slots. Suppose that both (P, n) -heavy and (P, n) -light slots exist. Create a new problem instance I' by deleting all small ads except those ads i having at least one copy scheduled in some (P, i) -heavy slot. Then (i) I' will be a strictly smaller instance than I , (ii) $\text{OPT}(I') \leq \text{OPT}(I)$, and (iii) $\text{LSLF}(I') = \text{LSLF}(I)$.*

Proof. We argue that (i) follows from the existence of (P, n) -light slots. In order for (P, n) -light slots to exist, there must be some non P -minimal small ad i , and every slot receiving a copy of i will be (P, i) -light by rule II. Thus i will be deleted when forming I' . Point (ii) is also straightforward, as deletion of ads can never cause OPT to increase. We therefore focus our attention on (iii).

Consider applying LSLF in parallel to simultaneously construct schedules for I and I' ; we will compare corresponding slots in the two schedules as we do so. At any point in time after scheduling all copies of ad i , let $\mathcal{H}(i)$ denote the set of (P, i) -heavy slots (with respect to the schedule for I) and let $\mathcal{L}(i)$ denote the set of (P, i) -light slots (also with respect to the schedule for I). We inductively argue the following: after scheduling any ad i ,

1. $H_i^I(t) = H_i^{I'}(t)$ for all $t \in \mathcal{H}(i)$, and
2. $H_i^I(t) \geq H_i^{I'}(t)$ for all $t \in \mathcal{L}(i)$.

The superscripts I and I' above refer to the schedule in which we are measuring the height of a slot. Otherwise stated, the heavy slots in the schedules I and I' will always agree in their heights, while the light slots in I 's schedule will always upper bound their corresponding slots in the schedule for I' . Since every slot t maximizing $H_n(t)$ (in either schedule) will belong to $\mathcal{H}(n)$, this will imply $\text{LSLF}(I) = \text{LSLF}(I')$.

The large ads all belong to both I and I' , and will be identically-scheduled for both instances. Consider therefore the insertion of an arbitrary small ad i , assuming that our inductive hypothesis holds for $i - 1$.

- $i \in I'$. Consider the schedule for instance I . Since i was not deleted, it is scheduled in some slot in $\mathcal{H}(i)$, and therefore also in some slot in $\mathcal{H}(i - 1)$. By lemma 3, i is scheduled in every slot in $\mathcal{L}(i - 1)$. The inductive hypothesis and lemma 3 together imply that $H_{i-1}^{I'}(t) < H_{i-1}^{I'}(u)$ for every $t \in \mathcal{L}(i - 1)$ and $u \in \mathcal{H}(i - 1)$; hence, i will also be scheduled in every slot in $\mathcal{L}(i - 1)$ in the schedule for I' . This ensures the invariant is maintained for all $t \in \mathcal{L}(i - 1)$. Since the heights of slots in $\mathcal{H}(i - 1)$ agree between the two schedules at time $i - 1$, ad i will be scheduled in analogous slots in $\mathcal{H}(i - 1)$ among both schedules. Some of these slots will be in $\mathcal{H}(i)$; the rest will move to $\mathcal{L}(i)$. In either case, since the heights of corresponding slots agree, the invariant will also be maintained for these slots.
- $i \notin I'$. In this case, i is not scheduled in any slot in $\mathcal{H}(i)$, so it can only appear in I 's schedule in slots in $\mathcal{L}(i - 1)$ and in $\mathcal{H}(i - 1) \setminus \mathcal{H}(i)$. By the invariant, heights of slots in these two sets in I 's schedule are already upper bounds on their corresponding slots in the schedule for I' , and we can only be strengthening this upper bound. Therefore the invariant is maintained.

4 A New Linear Programming Relaxation

Theorem 2 allows us to give a new linear programming relaxation for the ad scheduling problem. By rounding the solution to this linear program we obtain another $4/3$ -approximation algorithm. The linear program optimally (and fractionally) assigns the *small* ads on top of the base obtained by LSLF so as to minimize the tallest slot. Since we do not know OPT , we do not know which ads are large and which are small, but if the ads are sorted in decreasing order of height, the large ads must comprise some prefix of this list, so we must run our rounding algorithm on every prefix and take the best result. Henceforth, we can therefore assume the large ads are known.

The linear program is the following, where $\mathcal{A}_S$ denotes the set of small ads.

$LP =$ Minimize z
subject to:

$$\begin{aligned} \sum_{t \in [T]} x_{it} &= c_i & i \in \mathcal{A}_S \\ \sum_{i \in [n]} x_{it} h_i &\leq z - b_t(LSLF) & t \in [T] \\ 0 \leq x_{it} &\leq 1 & i \in [n], t \in [T]. \end{aligned} \quad (1)$$

It is not straightforward that this linear program is a lower bound on the optimum, since the optimum schedule might schedule the base differently than LSLF. We know of two ways of arguing that LP is a lower bound on OPT . The first proof is based on (contra)-polymatroids and is given below for completeness; the second follows from Theorems 5 and 6 where a simple combinatorial algorithm is shown to solve the LP optimally and with optimal value $\max_k L_k$, the lower bound given in Theorem 3.

Theorem 4. $LP \leq OPT$.

Proof. Let C denote the set of all vectors $l \in R^T$ such that the linear program with the right-hand-side of equation (1) replaced by l_t (from $z - b_t(LSLF)$) is feasible. By definition of Σ , we have that $s \in C$ where $s_t = OPT - b_t(\Sigma)$. By symmetry of the time slots and the fact that C is a convex set, this implies that

$$C \supseteq P := \text{conv} \{x : x_i = s_{\sigma(i)} \text{ for some permutation } \sigma \text{ of } [T]\} + R_+^T.$$

This latter polyhedral set P is a contra-polymatroid (see [10]) and can be completely described by inequalities in the following way:

$$P = \{x \in R^T : x(S) \geq g(|S|) \text{ for all } S \subseteq [T]\},$$

where $x(S) = \sum_{t \in S} x_t$ and $g(k)$ is the sum of the k smallest s_t , i.e. $g(k) = kOPT - B_k(\Sigma)$ using the notation introduced before Theorem 2. We claim that

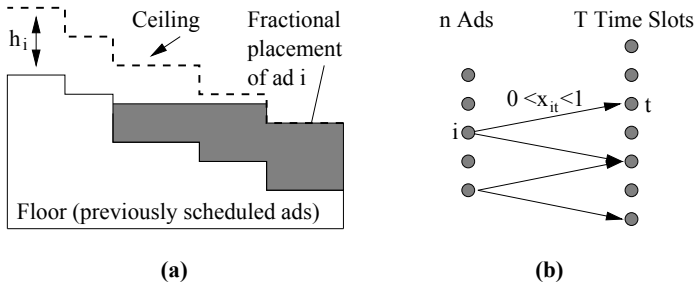


Fig. 2. Combinatorially solving the LP. (a) “Fluid” interpretation of fractional ad placement, (b) Bipartite graph of fractional assignments used for rounding.

the right-hand-side of (1) for $z = OPT$, i.e. $OPT - b_t(LSLF)$, is in P , hence in C , and therefore $LP \leq OPT$. Indeed, for any set S ,

$$\sum_{t \in S} (OPT - b_t(LSLF)) \geq |S|OPT - B_{|S|}(LSLF) \geq |S|OPT - B_{|S|}(\Sigma) = g(|S|)$$

by Theorem 2.

In the next theorem, we show that LP is equal to the lower bound given in Theorem 3. The proof of that theorem actually shows that, for any k , $LP \geq L_k$. This implies that $LP \geq \max_{k \in [T]} L_k$. The converse also holds:

Theorem 5. $LP = \max_{k \in [T]} L_k$.

In order to prove this theorem, we give a simple combinatorial algorithm that solves the LP and show that it has the right value. The algorithm first initializes the height H_t of each slot t to be $b_t(LSLF)$ and processes the small ads in any order. The process of scheduling each small ad i can be thought of as filling the top of the schedule with a fluid, as shown in Figure 2(a). Ad i is fractionally “poured” onto the slot with minimum height until this slot catches up to the second-shortest slot and both are filled together uniformly, and so on. However, we must prevent the height of this fluid in each slot from exceeding h_i ; this is done by imposing a “ceiling” at height h_i over the top of each slot, at which the fluid stops. When a total of c_i units of ad i have been fractionally filled in, we update $H_1 \dots H_T$ and the ceilings, and continue filling in the next ad.

Theorem 6. The “fluid” algorithm generates an optimal solution to LP.

Proof. In order to show that this algorithm optimally solves LP, consider the fractional solution it obtains at the end of its execution and the heights H_t of the time slots. By construction, $z = H_1 \geq H_2 \geq \dots \geq H_T$. Let k be the maximum slot such that $H_k = H_1 = z$. During the execution of the algorithm, if an ad is ever fractionally assigned to any two adjacent slots t and $t + 1$ (that is, if strictly less than its full height is assigned to both slots) then the heights of t

and $t + 1$ will remain equal for the remainder of the algorithm. In other words, since $H_k > H_{k+1}$, no ad was fractionally assigned to k and $k + 1$ simultaneously. Furthermore since the algorithm assigns at least as much of an ad to time slot $t + 1$ compared to t , no ad is assigned to any time slot in $[k]$ unless it is assigned fully to each of the time slots $k + 1, \dots, T$. This means that every small ad is $[k]$ -minimal and we thus have that $H_1 + H_2 + \dots + H_k = kL_k$. The fact that $H_1 = H_2 = \dots = H_k$ by our choice of k implies that $z = L_k$. This simultaneously proves the correctness of the algorithm for solving the LP and of Theorem 5.

Approximation algorithms with performance guarantee of $4/3$ can be obtained by solving this linear program and rounding the solution using classical rounding schemes such as the ones by Lenstra, Shmoys and Tardos [8] or by Shmoys and Tardos [11]. For example, consider in Figure 2(b) the bipartite graph corresponding to the fractional assignments produced by LP. That is, we have an edge from ad i to slot t if $0 < x_{it} < 1$. We assume without loss of generality that the edges in this subgraph form a forest, for if there were an alternating cycle among these edges we could “augment the flow” appropriately (while preserving the amount assigned to any time slot) around such a cycle, maintaining feasibility and optimality of our solution, until the x_{it} value on one of its edges reached either 0 or 1, thereby breaking the cycle. In this graph, the outdegree of every ad i must be at least 2, since we have $\sum_t x_{it} = c_i$ and c_i is an integer. Therefore, we must be able to find an alternating path with endpoints at two different slots t and t' , such that both t and t' have indegree of 1. We round our solution by augmenting flow on such alternating paths until all x_{it} values eventually become integral. During the process we only increase the flow entering a slot t if there is a single fractional edge (i, t) directed into t , so the total increase in height of any such slot will be at most $h_i \leq OPT/3$. Therefore, rounding increases the makespan of our solution by at most the maximum height of a fractionally-scheduled ad, which in this case is at most $OPT/3$. As described, our rounding approach takes time polynomial in n and T , but it is straightforward to eliminate the dependence on T by appropriate grouping of time slots. Further details are left for the full version.

5 A Pseudopolynomial $(1 + \epsilon)$ -Approximation Scheme

We describe briefly a $(1 + \epsilon)$ -approximation scheme whose running time is polynomial in n and T . The algorithm is similar to that of Hochbaum and Shmoys [5] for $P||C_{max}$. Let $\alpha = \epsilon/2$. For this section, we designate ads with heights larger than αOPT large, and the remaining ads small. As in the LP rounding case, we do not know which ads are large, so we must try our algorithm on every prefix of the sorted ads and take the best result. Let us therefore assume the large ads are known.

Our approximation scheme schedules the large ads via dynamic programming. We first run the greedy *LSLF* algorithm to obtain approximate bounds on OPT , so $3LSLF/4 \leq OPT \leq LSLF$. The heights of the large ads are

first rounded up to the nearest multiple of $3\alpha^2 LSLF/4$. Since large ads have heights at least αOPT , this inflates the height of each large ad (and hence also the optimal makespan) by a factor of at most $1 + \alpha$. After rounding, we have at most $\lceil \frac{LSLF}{3\alpha^2 LSLF/4} \rceil = O(1)$ distinct large ad sizes. We can encode each base shape as a vector $(n_0, n_1, \dots, n_{\lceil 4/3\alpha^2 \rceil})$ where n_i gives the number of slots having base height $3i\alpha^2 LSLF/4$, and $\sum n_i = T$. The number of distinct base shapes will therefore be at most $T^{\lceil 4/3\alpha^2 \rceil} = T^{O(1)}$, a polynomial in T . All achievable base shapes can be enumerated via dynamic programming in time polynomial in n and T . We omit further details. We henceforth assume that the base of a $(1 + \alpha)$ -approximate solution is known.

Fixing such a base, we solve LP to fractionally schedule the small ads. The optimal value of LP gives a lower bound on $(1 + \alpha)OPT$, and rounding of the small ads results in an increase of at most αOPT . Thus, our final schedule has makespan at most $(1 + \epsilon)OPT$.

Acknowledgements. We wish to thank the reviewers of this paper for their insightful comments.

References

1. M. Adler, P.B. Gibbons, and Y. Matias (1998). "Scheduling Space-Sharing for Internet Advertising". *Journal of Scheduling*. To appear.
2. J. Clifford, M. Posner (2001). "Parallel Machine Scheduling with High Multiplicity". *Mathematical Programming Ser. A* 89, 359–383.
3. M. Dawande, S. Kumar, and C. Sriskandarajah (2001). "Algorithms for Scheduling Advertisements on a Web Page: New and Improved Performance Bounds" *Journal of Scheduling*. To appear.
4. R. Graham (1969). "Bounds on Multiprocessing Timing Anomalies". *SIAM Journal on Applied Mathematics* 17, 416–429.
5. D. Hochbaum and D. Shmoys (1987). "Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results". *Journal of the ACM* 34, 144–162.
6. E. Horowitz and S. Sahni (1976). "Exact and Approximate Algorithms for Scheduling Nonidentical Processors". *Journal of the ACM* 23, 317–327.
7. Interactive Advertising Bureau (www.iab.net). IAB Internet Advertising Revenue Report.
8. J.K. Lenstra, D.B. Shmoys, and É. Tardos (1990). "Approximation algorithms for scheduling unrelated parallel machines". *Mathematical Programming* 46, 259–271.
9. S. T. McCormick, S. Smallwood, and F. Spieksma (2001). "A Polynomial Algorithm for Multiprocessor Scheduling with Two Job Lengths". *Mathematics of Operations Research* 26(1), 31–49.
10. A. Schrijver (2003). "Combinatorial Optimization - Polyhedra and Efficiency". Springer-Verlag.
11. D.B. Shmoys and É. Tardos (1993). "An Approximation Algorithm for the Generalized Assignment Problem". *Mathematical Programming* 62, 461–474.

Anycasting in Adversarial Systems: Routing and Admission Control

Baruch Awerbuch^{*1}, André Brinkmann^{**2}, and Christian Scheideler¹

¹ Department of Computer Science, Johns Hopkins University, 3400 N. Charles Street, Baltimore, MD 21218, USA, {baruch,scheideler}@cs.jhu.edu

² Heinz Nixdorf Institute and Department of Electrical Engineering, University of Paderborn, 33102 Paderborn, Germany, brinkman@hni.upb.de

Abstract. In this paper we consider the problem of routing packets in dynamically changing networks, using the anycast mode. In anycasting, a packet may have a set of destinations but only has to reach any one of them. This set of destinations may just be given implicitly by some anycast address. For example, each service (such as DNS) may be given a specific anycast address identifying it, and computers offering this service will associate themselves with this address. This allows communication to be made transparent from node addresses, which makes anycasting particularly interesting for dynamic networks, in which redundancy and transparency are vital to cope with a dynamically changing set of nodes. However, so far not much is known from a theoretical point of view about how to efficiently support anycasting in dynamic networks. This paper formalizes the anycast routing and admission control problem for arbitrary traffic in arbitrary dynamic networks, and provides first competitive solutions. In particular, we show that a simple local load balancing approach allows to achieve a near-optimal throughput if the available buffer space is sufficiently large compared to an optimal algorithm. Furthermore, we show via lower bounds and instability results that allowing admission control (i.e. dropping some of the injected packets) tremendously helps in keeping the buffer resources necessary to compete with optimal algorithms low.

Keywords: Adversarial routing, anycasting, online algorithms, load balancing, dynamic networks

^{*} Supported by DARPA grant F306020020550 “A Cost Benefit Approach to Fault Tolerant Communication” and DARPA grant F30602000-2-0526 “High Performance, Robust and Secure Group Communication for Dynamic Coalitions”.

^{**} Supported in part by the DFG-Sonderforschungsbereich 376 “Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen”. Part of the research was done while visiting the Johns Hopkins University, supported by a scholarship from the German Academic Exchange Service (DAAD Doktorandenstipendium im Rahmen des gemeinsamen Hochschulsonderprogramms III von Bund und Ländern).

1 Introduction

This paper studies the problem of supporting anycasting in adversarial networks. The notion of anycasting was first standardized in RFC 1546 [16]. In this RFC, IP anycast is defined as a network service that allows a sender to access the nearest of a group of receivers that share the same anycast address, where “nearest” is defined according to the routing system’s measure of distance. Usually, the receivers in the anycast group are replicas, able to support the same service (e.g. mirrored web servers). RFC 1546 proposes anycast as a means to discover a service location and provide host auto-configuration. For example, by assigning the same anycast address to a set of replicated FTP servers, a user downloading a file need not choose the best server manually from the list of mirrors. The user can simply use the anycast address to directly download the file from the nearest server. In order to aid host auto-configuration, all DNS servers may be given the same anycast address. In this case, a host that is moved to a new network need not be reconfigured with the local DNS address. The host can simply use the global anycast address to access a local DNS server. Service discovery and auto-configuration are seen as vital components of protocols for dynamic networks, and therefore anycasting is seen as a crucial mechanism to ensure robust support for networking services in mobile networks. Since its introduction, anycasting has received considerable attention in the systems community and it has been adopted by all proposed successors of IPv4 (e.g. Pip, SIPP, and IPv6). However, to our surprise, it seems that anycasting has not been investigated by the theory community so far.

Since in highly dynamic networks it may be very hard to predict which may be the nearest server belonging to some anycast address, it seems to be a formidable problem to efficiently support anycasting in dynamic networks, especially for those that are under adversarial control. However, we demonstrate in this paper that even if both the network and the packet injections are under adversarial control, distributed routing strategies can be found for anycasting with a close to optimal throughput. Thus, in principle, anycasting can even be supported in such networks as mobile ad-hoc networks, where connections between users may change quickly and unpredictably.

1.1 Our Approach and Related Results

We measure the performance of our protocols by comparing them with a best possible strategy that knows all actions of the adversary in advance. The performance is measured in terms of communication throughput and space overhead. In order to ensure a high throughput efficiency in dynamic networks, several challenging tasks have to be solved:

- *Routing*: What is the next edge to be traversed by a packet?
- *Queueing*: What is the next packet to be transmitted on an edge? In particular, which destination should be preferred?
- *Admission control*: What is the packet to be dropped if a buffer is full?

The study of adversarial models was initiated, in the context of *queueing alone*, by Borodin et al. [12]. Other work on queueing includes [6,13,14,15,17,18]. In these papers it is assumed that the adversary has to provide a path for every injected packet and reveals these paths to the system. The paths have to be selected so that they do not overload the system. Hence, it remains to find the right queueing discipline (such as furthest-to-go) to ensure that the number of packets in the system (resp. the time needed by packets to reach their destination) is bounded. However, the bounds on the buffer size given in these papers to avoid dropping any packet usually depend on the network size and are sometimes unrealistically high. This motivated Aiello et al. [5] to study the throughput performance of queueing disciplines under the assumption that the routing buffers have a fixed size (i.e. that is independent of network parameters), using an adversary that can inject an unbounded number of packets. In this case, of course, a queueing discipline cannot guarantee the delivery of every injected packet. So the goal is rather to find a queueing strategy whose throughput is as close as possible to a best possible throughput. Aiello et al. show among other results that there are queueing disciplines that are guaranteed to achieve an $\Omega(1/(d \cdot m))$ fraction of the best possible throughput achievable with the same buffer size, where m is the number of edges and d is the longest path injected by the adversary. This upper bound and their lower bound of $O(\sqrt{m})$ for the line that holds for arbitrary greedy protocols seem to indicate that online protocols cannot compete well with best possible protocols when using the same buffer size.

The study of adversarial models was initiated, in the context of *routing*, by Awerbuch, Mansour and Shavit [11] and further refined by [4,7,9,10,14]. In these papers the model is used that the adversary *does not* reveal the paths to the system, and therefore the routing protocol has to figure out paths for the packets by itself. Based on work by Awerbuch and Leighton [10], Aiello et al. [4] show that there is a simple distributed routing protocol that keeps the number of packets in transit bounded in a dynamic network if, roughly speaking, in each window of time the paths selected for the injected packets require a capacity that is below what the available network capacities can handle in the same window of time. Awerbuch et al. [7] generalize this to an adversarial model in which the adversary is allowed to control the network topology and packet injections as it likes, as long as for every injected packet it can provide a schedule to reach its destination. They show that even for the case that the network capacity is fully exploited, if all packets have the same destination, the number of packets in transit is bounded at any time.

With the exception of [5], the weakness of the adversarial models above is that they assume that the adversary never overloads the system with packets. In static networks this may be a reasonable restriction, since one can imagine that in principle it is possible to perform some kind of admission control *before* injecting a packet into the system. However, in highly dynamic networks such as mobile ad-hoc networks, this may not be possible without being too conservative and therefore wasting too much of the already scarce bandwidth. Hence, for dynamic networks it would be highly desirable to have protocols that can handle not

only the routing and queueing part but also packet-level admission control, i.e. dropping packets from either input or intermediate buffers.

Also, we note that all of the above work on adversarial queueing and routing only considered the *unicasting* mode (every packet has a single destination). We consider the more general *anycasting* mode, using a very general adversarial model that gets rid of somewhat artificial restrictions of previously suggested models for dynamic networks. In fact, the only limiting assumptions left in our model are that packets are of atomic nature (i.e. they cannot be split or compressed) and that packets cannot be killed by the adversary. Thus, our upper bounds also apply to other adversarial routing and queueing models suggested so far.

Finally, we note that all approaches in the adversarial routing area, including this current paper, are based on simple load balancing schemes first pioneered by Awerbuch, Mansour and Shavit [11], and refined in [1,2,3,4,7,9,10] for various routing purposes. Our achievement is to demonstrate that balancing even works for anycasting. Also, we use a much more general adversarial network model than was used in previous papers, and we consider the admission control problem.

In order to state our analytical results, we need some notation.

1.2 The Anycast Routing and Admission Control Model

First, we describe the basics of our network model and injection model. We assume that $V = \{1, \dots, n\}$ represents the set of nodes in the system. The selection of the edges is under adversarial control and can change from one time step to the next. We assume that all edges are directed. This does not exclude the undirected edge case, since an undirected edge can be viewed as consisting of two directed edges, one in each direction. Each edge can forward at most one packet in a time step. Each node can have at most Δ incoming and at most Δ outgoing edges at any time. Δ can be seen as the maximum number of active (logical or physical) connections a node can handle at the same time (due to, for example, its hardware restrictions). Apart from this restriction, the adversary can interconnect the nodes in an arbitrary way in each time step. This includes the possibility of connecting the same pair of nodes via several edges.

The adversary does not only control the topology of the network but also the injection of packets. Each anycast packet is given a fixed anycast group at the time of its injection. We allow this group just to be specified implicitly (for example, by an anycast address). Note that for implicitly specified groups, the nodes in the network may have no knowledge about their size. It may even be possible that the group is empty. Thus, our anycast algorithm has to cope with this situation.

The adversary can inject an arbitrary number of packets and can activate an arbitrary number of edges in each time step as long as the number of incoming or outgoing edges at a node does not exceed Δ . In this case, only some of the injected packets may be able to reach their destination, even when using a best possible strategy. Each time an anycast packet reaches one of its destinations, we count it as one *delivery*. The number of deliveries that is achieved by an algorithm is called its *throughput*. We are interested in maximizing the throughput. Since

the adversary is allowed to inject an unbounded number of packets, we will allow routing algorithms to drop packets so that a high throughput can be achieved with a buffer size that is as small as possible.

In order to compare the performance of a best possible strategy with our online strategies, we will use competitive analysis. We assume that both the optimal and the online algorithm are allowed to allocate one buffer in each node for each type of packet. Thus, if there are b different anycast addresses, then a node can allocate up to b different buffers. This will simplify the comparison. However, our competitive results also work if every node only has a single buffer (or a fixed number of buffers). In this case, the buffer overhead for our online algorithm has to be multiplied by b .

Given any sequence of edge activations and packet injections σ , let $\text{OPT}_B(\sigma)$ be the maximum possible throughput (i.e. the maximum number of deliveries) when using a buffer size of B (i.e. each buffer can store up to B packets), and let $A_{B'}(\sigma)$ be the throughput achieved by some given online algorithm A with buffer size B' . We call an online algorithm A (c, s) -competitive if for all σ and all B , A can guarantee that

$$A_{s' \cdot B}(\sigma) \geq c \cdot \text{OPT}_B(\sigma) - r$$

for any $s' \geq s$, where $r \geq 0$ is some value that is independent of σ (but may depend on s , B and n). $c \in [0, 1]$ denotes here the fraction of the best possible throughput that can be achieved by A and s denotes the space overhead necessary to achieve this. If c can be brought arbitrarily close to 1, A is also called $s(\epsilon)$ -competitive (or simply *competitive*), where $s(\epsilon)$ reflects the relationship between s and ϵ with $c = 1 - \epsilon$. Obviously, it always holds that $s(\epsilon) \geq 1$, and the smaller $s(\epsilon)$, the better is the algorithm A .

In the following, B will always mean the buffer size of an optimal routing algorithm.

1.3 New Results

Our new results are arranged in two sections. In Section 2, we demonstrate that if it is allowed to drop packets, a near-optimal throughput can be achieved with a low space overhead. In particular, we present a simple algorithm for anycasting, called *T-balancing algorithm*, that achieves the following result:

For every $T \geq B + 2(\Delta - 1)$, the T -balancing algorithm is $1 + (1 + (T + \Delta)/B)L/\epsilon$ -competitive, where L is the *average* path length used by successful packets in an optimal solution. For $B \geq \Delta$ and $T = O(B)$, this boils down to a competitive ratio of $O(L/\epsilon)$. The result is sharp up to a constant factor.

In Section 3, we demonstrate with the help of lower bounds and instability results that even if the adversary is friendly (i.e. it only injects packets that can be delivered when using a buffer size of B), routing without the ability to drop packets may have a poor performance both with respect to throughput and space overhead.

Some of the proofs are only sketched due to space limitations. Please see [8] for details.

2 Adversarial Anycasting

Let $h_{v,a,t}$ denote the amount of packets in the buffer for anycast address a in node v at the beginning of time step t . $h_{v,a,t}$ will also be called the *height* of the corresponding buffer. The maximum height a buffer can have is denoted by H .

We now present a simple balancing strategy that extends the balancing strategies used by Aiello et al. [4] and Awerbuch et al. [7] by a rule for deleting packets. In every time step $t \geq 1$ the T -balancing algorithm performs the following operations.

1. For every edge (v, w) , determine the anycast address a with maximum $h_{v,a,t} - h_{w,a,t}$ and check whether $h_{v,a,t} - h_{w,a,t} > T$. If so, send a packet for a from v to w (otherwise do nothing).
2. Receive all incoming packets and absorb all packets that reached the destination. Afterwards, receive all newly injected packets. If a packet cannot be stored in a buffer because its height is already H , delete it.

Note that if T is large enough compared to Δ , then packets are guaranteed never to be deleted at intermediate buffers but only at the source. This provides the sources with a very easy rule to perform admission control: if a packet cannot be stored because its buffer is already full, delete it.

Let L denote an upper bound on the (best possible) average path length used by the successful packets in an optimal algorithm with buffer size B , and let Δ denote the maximum number of edges leaving or leading to a node that can be active at any time. We do not demand that these edges have to connect different pairs of nodes. Hence, the result below also extends to dynamic networks with non-uniform edge capacities.

Theorem 1. *For any $\epsilon > 0$ and any $T \geq B+2(\Delta-1)$, the T -balancing algorithm is $1 + (1 + (T + \Delta)/B)L/\epsilon$ -competitive.*

Proof. To simplify the analysis, we prove the competitive ratio for a more general model than our anycast model, called *option set model*.

In the option set model, we have a set of nodes V' with a single buffer each, and all injected packets want to go to the same destination $d \in V'$. The adversary can inject an arbitrary number of packets in each time step. Also, it can activate an arbitrary collection of edge sets $E_1, \dots, E_k \subseteq V' \times V'$, called *option sets*, in each step as long as every node $v \in V' \setminus \{d\}$ has an incoming or outgoing edge in at most Δ many sets. For each option set E_i , the algorithm is allowed to use only one edge in E_i for the transmission of a packet.

This model is indeed more general than our anycast model.

Lemma 1. *Any algorithm that is c -competitive in the option set model is also c -competitive in the anycast model.*

Proof. For this it suffices to show how to transform the anycast model into the option set model. Suppose that A is the set of all anycast addresses. Then we define $V' = V \times A$, i.e. each buffer in the original model represents a node in the option set model. Each edge $e = (v, w)$ that is activated in the anycast model

can then be represented as option set $E_e = \{((v, a), (w, a)) \mid a \in A\}$. Since all packets reaching their destination buffers in the anycast model are absorbed, we can view all of these buffers as a single node in the option set model without affecting the throughput. $\square$

Hence, in the following we only work with the option set model.

Let N be the number of non-destination nodes in the option set model, and let node 0 represent the destination node. The height of node 0 is always 0, since any packet reaching 0 will be absorbed. For each of the remaining nodes we assume that it has H slots to store packets. The slots are numbered in a consecutive way starting from below with 1. Every slot can store at most one packet. After every step of the balancing algorithm we assume that if a node holds h packets, then its first h slots are occupied. The *height* of a packet is defined as the number of the slot in which it is currently stored. If a new packet is injected, it will obtain the lowest slot that is available after all packets that are moved to that node from another node have been placed.

For each successful packet in an optimal algorithm, a schedule can be identified. A schedule $S = (t_0, (e_1, t_1), \dots, (e_\ell, t_\ell))$ consists of a sequence of movements by which the injected packet P is sent from its source node to the destination. It has the property that P is injected at time t_0 , the edges $e_1, \dots, e_\ell$ form a connected path, with the starting point of e_1 being the source of P and the endpoint of e_ℓ being the destination of P , the time steps have the ordering $t_0 < t_1 < \dots < t_\ell$, and edge e_i was available in some option set at time t_i for all $1 \leq i \leq \ell$. Certainly, no two schedules are allowed to use the same option set at the same time. A schedule $S = (t_0, (e_1, t_1), \dots, (e_\ell, t_\ell))$ is called *active* at time t if $t_0 \leq t \leq t_\ell$. The *position* of a schedule at time t is the node at which its corresponding packet would be if it is moved according to S . An edge in an option set is called a *schedule edge* if it belongs to a schedule of a packet. Suppose that we want to compare the performance of the balancing algorithm with an optimal algorithm that uses a buffer size of B . Then the following fact obviously holds.

Fact 1 *At every time step, at most B schedules can have their current position at some node v .*

Next we introduce some further notation. We will distinguish between three kinds of packets: *representatives*, *zombies*, and *losers*. During their lifetime, the packets have to fulfill certain rules. (These rules will be crucial for our analysis. The balancing algorithm, of course, cannot and does not distinguish between these types of packets.) Every injected packet that has a schedule (i.e. that will be delivered by the optimal algorithm) will initially be a representative. Every other injected packet will initially be a zombie. The goal of a representative is to stay with its schedule, and the goal of a zombie is to stay at a slot of height more than $H - B$. Whenever this cannot be fulfilled, the packet is transformed into a loser. Together with Fact 1, this implies the following fact.

Fact 2 *At any time, the number of zombies and representatives stored in a node is at most B .*

If a packet is injected into a full node, then the highest available loser will be selected to take over its role (Fact 2 implies that this is always possible if $H > B$).

Our goal for the analysis is to ensure that a representative always stays with its schedule as long as this is possible. That is, each time the schedule moves, the representative tries to move with it, and otherwise it tries to stay at the current position of the schedule. This implies the following rules for a representative R when the adversary offers an option set containing one of its schedule edges $e = (v, w)$:

1. A packet is sent along e : Then we always select R to be moved along e .
2. No packet is sent along edge e : If w has a loser, then the representative exchanges its role with the highest available loser in w . In this case we will also talk about a *virtual* movement. Otherwise, the representative is simply transformed into a loser. In this case, we will disregard the rest of the schedule (i.e. we will not select a representative for it afterwards and the rest of the schedule edges will simply be treated as non-schedule edges).

Furthermore, if a packet is sent along a non-schedule edge $e = (v, w)$, then we always make sure that none of the representatives is moved out of v but only a loser (which always exists if T is large enough).

The three types of packets are stored in the slots in a particular order. The lowest slots are always occupied by the losers, followed by the zombies and finally the representatives.

Let $h_{v,t}$ be the *height* of node v (i.e. the number of packets stored in the buffer represented by v) at the beginning of time step t , and let $h'_{v,t}$ be its height when considering only the losers. The *potential* of node v at step t is defined as $\phi_{v,t} = \sum_{j=1}^{h'_{v,t}} j = \binom{h'_{v,t}+1}{2}$ and the potential of the system at step t is defined as $\Phi_t = \sum_v \phi_{v,t}$.

First, we study how the potential can change in a single step. Since schedules are not allowed to overlap, every option set contains either one or no schedule edge. To simplify the consideration of these two cases, we consider the option sets given in a time step one by one, starting with option sets without a schedule edge and always assuming the worst case concerning previously considered option sets. Also, when processing these option sets, we always use the (worst case) rule that if a loser is moved to some node w , it will for the moment be put on top of all old packets in w . This will simplify the consideration of option sets with a schedule edge. At the end, we then move all losers down to fulfill the ordering condition for the representatives, zombies, and losers. This will certainly only decrease the potential. Using this strategy, we can show the following result.

Lemma 2. *If $T \geq B + 2(\Delta - 1)$, then any option set that does not contain a schedule edge does not increase the potential of the system.*

Proof. Consider any fixed option set without a schedule edge. If no edge in the given option set is used by a packet, the lemma is certainly true. Otherwise, let $e = (v, w)$ be the edge along which a packet is sent. Note that in this case, $h_{v,t} - h_{w,t} > T$. If $T \geq B + 2(\Delta - 1)$, then even after $\Delta - 1$ removals of packets

from v and the arrival of $\Delta - 1$ packets at w , there are still losers left in v , and the height of the highest of these is higher than the height of w . Hence, we can avoid moving any representative away from the position of its schedule and instead move a loser from v to w without increasing the potential. $\square$

For option sets with a schedule edge (i.e. an edge that still has a representative associated with it), only a slight increase in the potential is caused.

Lemma 3. *If $T \geq B + 2(\Delta - 1)$, then every option set that contains a schedule edge increases the potential of the system by at most $T + B + \Delta$.*

Proof. Consider some fixed option set with a schedule edge $e = (v, w)$. If e is selected for the transmission of a packet, then we can send the corresponding representative along e , which has no effect on the potential.

Otherwise, it must be that either $\delta_e = h_{v,t} - h_{w,t} \leq T$ or $\delta_e > T$ and another edge was preferred. In both cases, the representative R for e has to be moved virtually or transformed into a loser.

First of all, note that our rule of placing new losers on top of the old packets makes sure that the height of the representative in v does not increase. Furthermore, there are two ways for w to lose losers before considering e : either an unused schedule edge to w forced a virtual movement of a representative to w , or a used non-schedule edge from w forced to move a loser out of w . Let s be the number of edges with the former property and ℓ be the number of edges with the latter property. If w had r representatives (and zombies) at the beginning of t , then it must hold that $r + s - (\Delta - \ell) + 1 \leq B$ to ensure that at the end of step t , w has at most B representatives (the $+1$ is due to e). Thus, $r + s + \ell \leq B + \Delta - 1$. Hence, if there is still a loser left in w when considering e , the highest of these must have a height of at least $h_{w,t} - (B + \Delta - 1)$. Therefore, if $h_{w,t} \geq B + \Delta$, then it is possible to exchange places between R and a loser in w so that the potential increases by at most

$$h_{v,t} - (h_{w,t} - (B + \Delta - 1)) = \delta_e + B + \Delta - 1. \quad (1)$$

If $\delta_e \leq T$, this is at most $T + B + \Delta$. If $h_{w,t} < B + \Delta$, then it may be necessary to convert R into a loser. However, since $h_{v,t} - h_{w,t} \leq T$, this increases the potential also by at most $T + B + \Delta$.

Otherwise, δ_e might be quite big, but in this case there must be some other edge $e' = (v', w')$ that won against e because $\delta_{e'} \geq \delta_e$. Since $\delta_{e'} > T$, v' must have a loser even if $\Delta - 1$ losers already left v' and the maximum possible number of losers in v' was converted into representatives. In fact, similar to w above, the height of the highest of the remaining losers in v' must be at least $h_{v',t} - (B + \Delta - 1)$. On the other hand, w' can receive at most $\Delta - 1$ other packets before receiving the packet sent by e' . So the potential drop due to moving the highest available loser in v' to w' is at least

$$(h_{v',t} - B - \Delta + 1) - (h_{w',t} + \Delta) = (h_{v',t} - h_{w',t}) - B - 2\Delta + 1 \geq \delta_e - B - 2\Delta + 1. \quad (2)$$

Subtracting (2) from (1), the increase in potential due to the given option set is at most $(\delta_e + B + \Delta - 1) - (\delta_e - B - 2\Delta + 1) = 2B + 3\Delta - 2$. If $T \geq B + 2(\Delta - 1)$, this is at most $T + B + \Delta$. $\square$

In addition to option sets, also injection events and the transformation of a zombie into a loser can influence the potential. This will be considered in the next two lemmata.

Lemma 4. *Every deletion of a newly injected packet decreases the potential by at least $H - B$.*

Proof. According to Fact 2, the highest available loser in a full node must have a height of at least $H - B$. Since the deletion of a newly injected packet causes this loser to be transformed into a representative or zombie, this decreases the potential by at least $H - B$. (Note that in case of a zombie, it might be directly afterwards converted back into a loser, but this will be considered in the next lemma.) $\square$

If an injected packet is not deleted, this will initially not affect the potential, since it will either become a representative or a zombie. However, a zombie may be converted into a loser.

Lemma 5. *Every zombie can increase the potential by at most $H - B$.*

Proof. Note that zombies do not count for the potential. Hence, the only time when a zombie influences the potential is the time when it is transformed into a loser. Since we allow this only to happen if the height of a zombie is at most $H - B$, the lemma follows.

Now we are ready to prove an upper bound on the number of packets that are deleted by the balancing algorithm.

Lemma 6. *Let σ be an arbitrary sequence of edge activations and packet injections. Suppose that in an optimal strategy, s of the injected packets have schedules and the other z packets do not. Let L be the average length of the schedules. If $H \geq B + 2(\Delta - 1)$, then the number of packets that are deleted by the balancing algorithm is at most*

$$s \cdot \frac{L(T + B + \Delta)}{H - B} + z.$$

Proof. First of all, note that only newly injected packets get deleted. Let p denote the number of option sets with a schedule edge and d denote the number of packets that are deleted by the balancing algorithm. Since

- due to Lemma 2 option sets without a schedule edge do not increase the potential,
- due to Lemma 3 every option set with a schedule edge increases the potential by at most $T + B + \Delta$,
- due to Lemma 4 every deletion of a newly injected packet decreases the potential by at least $H - B$, and
- due to Lemma 5 every zombie increases the potential by at most $H - B$,

it holds for the potential Φ after executing σ that $\Phi \leq p \cdot (T + B + \Delta) + z \cdot (H - B) - d \cdot (H - B)$. Since on the other hand $\Phi \geq 0$, it follows that

$$d \leq \frac{p \cdot (T + B + \Delta)}{H - B} + z.$$

Using in this inequality the fact that the average number of edges used by successful packets is at most L , and therefore the number of injected packets with a schedule, s , satisfies $s \geq p/L$, concludes the proof of the lemma. $\square$

From Lemma 6 it follows that the number of packets that are successfully delivered to their destination by the balancing algorithm must be at least

$$s + z - \left(s \cdot \frac{L(T + B + \Delta)}{H - B} + z \right) - H \cdot N = s \cdot \left(1 - \frac{L(T + B + \Delta)}{H - B} \right) - H \cdot N,$$

where N is the number of (virtual) non-destination nodes. For $H \geq L(T + B + \Delta)/\epsilon + B$ this is at least $(1 - \epsilon)s - r$ for some value r independent of the number of packets successful in an optimal schedule. $\square$

Next we demonstrate that the analysis of the T -balancing algorithm is essentially tight, even when using just a single destination.

Theorem 2. *For any $\epsilon > 0$, $T > 0$, and $L \geq 1$, the T -balancing algorithm requires a buffer size of at least $T \cdot (L - 1)/\epsilon$ to achieve a more than $1 - \epsilon$ fraction of the best possible throughput.*

Proof. Consider a source node s that is connected to a destination d via two paths: one of length 1 and one of length $(L - 1)/\epsilon$. Further suppose packets are injected at s so that $1 - \epsilon$ of the injected packets have a schedule along the short path and ϵ of the packets have a schedule along the long path. Then the average path length is $1(1 - \epsilon) + ((L - 1)/\epsilon) \cdot \epsilon \leq L$. Since each time a packet is moved forward along a node its height (i.e. slot number) must decrease by at least T , a packet can only reach the destination along the long path if s has a buffer of size $H \geq T \cdot (L - 1)/\epsilon$. Hence, such a buffer size is necessary to achieve a throughput of more than $1 - \epsilon$. $\square$

3 Unicasting without Admission Control

In this section we demonstrate that routing without admission control mechanisms seems to be very difficult if not impossible, even in the adversarial unicast setting, and even if an unbounded (or extremely high) amount of resources for the buffering is available.

We will start by defining some properties of online routing algorithms which intuitively seem to be necessary for the successful online delivery of packets. A *priority function* $f : \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \mathbb{N}_0$ gets as arguments two buffer heights and outputs a number determining the priority with which a packet should be sent from one to the other buffer. In a balancing algorithm that uses a priority

function f , the pair with the highest priority wins. We call f *monotonic* if for all $h_1, h_2 \in \mathbb{N}_0$, $f(h_1 + 1, h_2) > f(h_1, h_2)$ and $f(h_1 + 1, h_2 + 1) \geq f(h_1, h_2)$.

Consider a routing algorithm that uses a monotonic priority function to determine a winning buffer pair (h_1, h_2) for each activated edge in the unicast model. If $h_1 \leq h_2$, no packet is allowed to be sent. Otherwise, a packet for that pair (or none) may be sent, but if the buffer corresponding to h_2 is a destination buffer, a packet has to be sent for that pair. Intuitively, these rules seem to be reasonable to ensure a high throughput, and we will therefore call this class of routing algorithms *natural* algorithms.

We start with an observation demonstrating that for adversaries that are unbounded in their injections it is necessary to drop packets in a natural routing algorithm in order to make sure that *any* of the injected packets can be delivered, even if only two different destinations are used. Note that when we speak about algorithms that do not drop packets, this implies that they must have sufficient space to accommodate all injected packets.

Claim. For every natural algorithm that does not drop packets, there is an adversary for unicast injections using just two different destinations that can force the algorithm never to deliver a packet, no matter how high the throughput of an optimal strategy can be.

Proof. The adversary will simply pick one destination as the so-called *dead destination* and will inject so many packets into the system that whenever an edge is offered, a packet will be sent for the dead destination. Hence, the adversary can prevent packets from reaching the good destination, although there may be plenty of opportunities, had the good packets been chosen. On the other side, the adversary will never offer an edge directly to the dead destination. Hence, no packet will ever get delivered. $\square$

Thus, unbounded adversaries seem to be difficult to handle without allowing packets to get dropped. However, what about “friendly” adversaries, i.e. adversaries that only inject packets so that when using an optimal algorithm, only a bounded number of packets are in transit at any time without deleting any? We show that also in this case some natural algorithms have severe problems if packets cannot be dropped.

Theorem 3. *If the adversary is allowed to inject packets for more than one destination, then the adversary can force the T -balancing algorithm to store by a factor of $\Theta(2^{n/4})$ more packets in a buffer than for an optimal algorithm.*

Proof. (Sketch) For the proof it is sufficient to use two destinations, a and b , and to set $B = 1$. Given a node v , the height of its a -buffer is denoted by $h_a(v)$ and the height of its b -buffer is denoted by $h_b(v)$.

We show the theorem by complete induction. Suppose that we can construct a scheme using $2(5 + 2i)$ nodes with two nodes $v_i^{(a)}$ and $v_i^{(b)}$ so that $h_a(v_i^{(a)}) \geq H_i$ and $h_b(v_i^{(a)}) = 0$, and $h_a(v_i^{(b)}) = 0$ and $h_b(v_i^{(b)}) \geq H_i$, where $H_i = 2^i \max\{4T, 3\} - (2^i - 1)(2T + 1)$. Then we can show that 4 more nodes suffice to identify nodes so that the hypothesis above also holds for $i + 1$. The basic

idea is to create “copies” u_a and u_b of $v_i^{(a)}$ and $v_i^{(b)}$ and then to inject schedules for a -packets (resp. b -packets) with path $(v_{i+1}^{(a)}, u_b, u_a, a)$ (resp. $(v_{i+1}^{(b)}, u_a, u_b, b)$). $\square$

The theorem implies together with the results in [7] that only for the case that we have a single destination, the T -balancing algorithm without a dropping rule can be space-efficient under friendly adversaries.

What about other natural algorithms studied in the literature such as algorithms based on exponential priority functions (e.g. [10])? A routing algorithm is called *stable* if the number of packets in transit does not grow unboundedly with time. In order to investigate the stability of natural algorithms, we start with an important property of natural algorithms that allows us to study instability in the option set model (suggested in the proof of Theorem 1) instead of the original unicast model, which is much more difficult to handle.

Theorem 4. *For any natural deterministic algorithm it holds: If it is not stable in the option set model, it is also not stable in the adversarial unicast model.*

Proof. (Sketch) We only show how to get from the anycast to the unicast model. See [8] for details.

Consider any natural deterministic algorithm A that is instable in the anycast setting. Let V be the set of nodes and let $\mathcal{D} = (D_1, \dots, D_N)$ be the set of anycast sets. To prove instability for A in the unicast model, we extend V to $V \cup \{d_1, \dots, d_N\}$, where d_i is the new and only destination node for packets originally having destination set D_i . Let S be the strategy that caused instability for A in the anycast model. We simulate S until a packet of type i is supposed to reach one of its destination nodes D_i . Instead, we will offer now an edge to d_i . If this edge is taken by a packet of type i , we continue with the simulation.

Otherwise, it follows from the definition of a natural algorithm that another packet must have been sent to d_i . This causes the total number of packets stored in the buffers in $\{d_1, \dots, d_N\}$ to increase by one. Then, we remove all packets from V by offering again and again edges to destinations d_i and start from the beginning with the simulation of S .

Thus, either we obtain a perfect simulation of S for the unicast case, in which case A will be instable, or we increase the number of packets in the buffers in $\{d_1, \dots, d_N\}$ in every failed simulation attempt, which will also cause A to become instable. This completes the proof. $\square$

The theorem allows us to show the following result.

Theorem 5. *Natural routing algorithms which are based on exponential priority functions are not stable.*

Proof. By algorithms with exponential priority function we mean algorithms using the potential drop $f(h_1, h_2) = (\phi(h_1) + \phi(h_2)) - (\phi(h_1 - 1)) + \phi(h_2 + 1)$ with $\phi(h) = \sum_{i=1}^h e^{\alpha \cdot i}$ for some $\alpha > 0$ to determine the priority of a packet movement.

We will show that this rule can cause packets not to be delivered under certain circumstances, and that these situations can be generated arbitrarily

often. We assume that the nodes are sorted according to their heights with $h_{N-1} \geq h_{N-2} \geq \dots \geq h_1 \geq h_0$ and that node 0 is the destination node.

Lemma 7. *If the height difference between node $(N-1)$ and node 1 is $h_{N-1} - h_1 \geq \frac{\ln(e+1)}{\alpha}$, a new packet can be injected without any packet leaving the system.*

Proof. We assume that the adversary injects a new packet into node 1, which stores the lowest number of packets and has height h_1 before the injection of the packet. Then the adversary offers an option set with the two links $\{(N-1, 1), (1, 0)\}$. This option set is a valid schedule for the newly injected packet. The algorithm, however, will choose link $(N-1, 1)$ if $e^{\alpha \cdot h_{N-1}} - e^{\alpha \cdot (h_1+1)} > e^{\alpha \cdot h_1} - e^\alpha$, which is true if $h_{N-1} - h_1 > \frac{\ln(e+1)}{\alpha}$. $\square$

The important observation from the previous lemma is that the necessary height difference between the two nodes does not depend on the actual height of these nodes. If it is always possible to create this fixed height difference for a given algorithm and a given number of packets in the system, then the algorithm is not stable. In the next lemma we will show that this is the case.

Lemma 8. *Given a network with at least $\Delta + 1$ nodes it is possible to achieve a difference in height of at least Δ packets between the node with the highest number of packets and the non-destination node with the lowest number of packets without reducing the number of packets, or the algorithm is instable.*

Proof. Pick a set S of Δ non-destination nodes, and consider the following strategy: Suppose that there are two nodes in S of equal height, say v and w . Then we inject a packet in v and offer the option set $\{(v, w)\}$. If the algorithm sends a packet, we offer the option set $\{(v, 0)\}$ and otherwise the option set $\{(w, 0)\}$. This ensures that the injected packet will have a schedule in any case and that the number of packets in S does not change. Furthermore, using the potential function $\phi_u = \sum_{i=1}^{h_u} i$, one can show that this operation increases the potential in S .

Hence, either the number of packets in S goes to infinity or there cannot be two nodes in S of the same height any more. In the latter case, this means that the highest and lowest node in S must have a difference of at least Δ . $\square$

We now assume that we have a network with at least $\frac{\ln(e+1)}{\alpha} + 1$ nodes. From Lemma 8 we know that in this case a height difference of at least $\frac{\ln(e+1)}{\alpha}$ can be created.

After this, the adversary repeats the strategies in Lemma 7 and Lemma 8 again and again. With every iteration, the number of packets in the system will increase by one, which proves the theorem. $\square$

The proof of the theorem immediately implies the following result.

Corollary 1. *Natural routing algorithms which always prefer the buffer with the largest number of packets are not stable.*

We conjecture that any (natural) online algorithm is either instable or requires an exponential buffer size to be stable under friendly adversaries, which would imply together with Theorem 1 that the ability to drop packets can tremendously improve the performance of routing algorithms.

4 Conclusions and Open Problems

In this paper we presented a simple balancing algorithm for anycasting in adversarial systems. Many open questions remain. Although our space overhead is already reasonably low (essentially, $O(L/\epsilon)$), the question is whether it can still be reduced. For example, could knowledge about the location of a destination or structural properties of the network (for instance, it has to form a planar graph) help to get better bounds? Or are there other protocols that can achieve a lower space overhead in general?

References

1. Y. Afek, B. Awerbuch, E. Gafni, Y. Mansour, A. Rosen, and N. Shavit. Slide – the key to polynomial end-to-end communication. *Journal of Algorithms*, 22(1):158–186, 1997.
2. Y. Afek and E. Gafni. End-to-end communication in unreliable networks. In *PODC '88*, pages 131–148, 1988.
3. W. Aiello, B. Awerbuch, B. Maggs, and S. Rao. Approximate load balancing on dynamic and synchronous networks. In *STOC '93*, pages 632–641, 1993.
4. W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén. Adaptive packet routing for bursty adversarial traffic. In *STOC '98*, pages 359–368, 1998.
5. W. Aiello, R. Ostrovsky, E. Kushilevitz, and A. Rosén. Dynamic routing on networks with fixed-size buffers. In *SODA '03*, 2003.
6. M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *FOCS '96*, pages 380–389, 1996.
7. B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler. Simple routing strategies for adversarial systems. In *FOCS '01*, pages 158–167, 2001.
8. B. Awerbuch, A. Brinkmann, and C. Scheideler. Anycasting and multicasting in adversarial systems. Technical report, Dept. of Computer Science, Johns Hopkins University, March 2002. See <http://www.cs.jhu.edu/~scheideler>.
9. B. Awerbuch and F. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *FOCS '93*, pages 459–468, 1993.
10. B. Awerbuch and F. Leighton. Improved approximation algorithms for the multicommodity flow problem and local competitive routing in dynamic networks. In *STOC '94*, pages 487–496, 1994.
11. B. Awerbuch, Y. Mansour, and N. Shavit. End-to-end communication with polynomial overhead. In *FOCS '89*, pages 358–363, 1989.
12. A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. In *STOC '96*, pages 376–385, 1996.
13. D. Gamarnik. Stability of adversarial queues via fluid models. In *FOCS '98*, pages 60–70, 1998.
14. D. Gamarnik. Stability of adaptive and non-adaptive packet routing policies in adversarial queueing networks. In *STOC '99*, pages 206–214, 1999.
15. A. Goel. Stability of networks and protocols in the adversarial queueing model for packet routing. In *SODA '99*, pages 911–912, 1999.
16. C. Partridge, T. Mendez, and W. Milliken. Rfc 1546: Host anycasting service, November 1993.

17. C. Scheideler and B. Vöcking. From static to dynamic routing: Efficient transformations of store-and-forward protocols. In *STOC '99*, pages 215–224, 1999.
18. P. Tsaparas. Stability in adversarial queueing theory. Master's thesis, Dept. of Computer Science, University of Toronto, 1997.

Dynamic Algorithms for Approximating Interdistances

Sergei Bespamyatnikh¹ and Michael Segal²

¹ Department of Computer Science, University of Texas at Dallas,
Richardson, TX 75083, USA

besp@utdallas.edu, <http://www.utdallas.edu/~besp>

² Department of Communication Systems Engineering,
Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
segal@cs.bgu.ac.il, <http://www.cs.bgu.ac.il/~segal>

Abstract. In this paper we present efficient dynamic algorithms for approximation of k^{th} , $1 \leq k \leq \binom{n}{2}$ distance defined by some pair of points from a given set S of n points in d -dimensional space, for every fixed d . Our technique is based on dynamization of well-separated pair decomposition proposed in [11], computing approximate nearest and farthest neighbors [23,26] and use of persistent search trees [18].

1 Introduction

Let S be a set of n points in $\mathbb{R}^d$, $d \geq 1$ and let $1 \leq k \leq \frac{n(n-1)}{2}$. Let $d_1 \leq d_2 \leq \dots \leq d_{\binom{n}{2}}$ be the L_p -distances determined by the pairs of points in S . In this paper we consider the dynamic version of the following optimization problem:

- **Distance selection.** Compute the k -th smallest Euclidean distance between a pair of points of S .

In the dynamic version of the distance selection problem points are allowed to be inserted or deleted and given a number k , $1 \leq k \leq \binom{|S|}{2}$ one wants to answer efficiently what is the k -th smallest distance between a pair of points of S (by $|S|$ we denote the cardinality of the current set of points).

The distance selection problem above received a lot of attention during the past decade. The solution to the distance selection problem can be obtained using a parametric searching. The decision problem is to compute, for a given real r , the sum $\sum_{p \in S} |D_r(p) \cap (S - \{p\})|$, where $D_r(p)$ is the closed disk of radius r centered at p . Agarwal et al. [1] gave an $O(n^{\frac{4}{3}} \log^{\frac{4}{3}} n)$ expected-time randomized algorithm for the decision problem, which yields an $O(n^{\frac{4}{3}} \log^{\frac{8}{3}} n)$ expected-time algorithm for the distance selection problem. Goodrich [22] derandomized this algorithm, at a cost of an additional polylogarithmic factor in the runtime. Katz and Sharir [27] obtained an expander-based $O(n^{4/3} \log^{2+\varepsilon} n)$ -time deterministic algorithm for this problem. By applying a randomized approach Chan [13] was able to obtain an $O(n \log n + n^{2/3} k^{1/3} \log^{5/3} n)$ expected time algorithm for this problem. Bespamyatnikh and Segal [9] considered an approximation version of the distance

selection problem. For a distance d determined by some pair of points in S and for any fixed $0 < \delta_1 \leq 1$, $\delta_2 \geq 1$, the value d' is the (δ_1, δ_2) -approximation of d , if $\delta_1 d \leq d' \leq \delta_2 d$. They [9] present an $O(n \log^3 n / \varepsilon^2)$ runtime solution for the distance selection problem that computes a pair of points realizing distance d' that is either $(1, 1 + \varepsilon)$ or $(1 - \varepsilon, 1)$ -approximation of the actual k -th distance, for any fixed $\varepsilon > 0$. They also present an $O(n \log n / \varepsilon^2)$ time algorithm for computing the $(1 - \varepsilon, 1 + \varepsilon)$ -approximation of k -th distance and show how to extend their solution in order to answer efficiently the queries approximating k -th distance for a static set of points. Agarwal et al. [1] considers a similar problem, where one wants to identify approximate “median” distance, that is, a pair of points $p, q \in S$ with the property that there exist absolute constants c_1 and c_2 such that $0 < c_1 < \frac{1}{2} < c_2 < 1$ and the rank of the distance determined by p and q is between $c_1 \binom{n}{2}$ and $c_2 \binom{n}{2}$. They [1] showed how to solve this problem in $O(n \log n)$ time. Arya and Mount [4] introduced a *balanced box-decomposition tree* (BBD tree) in order to answer efficiently approximate range searching queries. They obtained $O(\log n + \frac{1}{\varepsilon^d})$ query time for d -dimensional point sets using linear space after $O(n \log n)$ preprocessing time. Their results also can be used to solve the decision version of the distance selection problem with $(1 - \varepsilon, 1 + \varepsilon)$ -approximation in $O(n \log n + \frac{n}{\varepsilon^2})$ runtime.

We call an algorithm a *almost-linear-time approximation scheme with almost logarithmic update time (ALTAS-LOG)* of order (c_1, c_2) if it has a preprocessing time of the form $O(n \log^{l_1} n / \varepsilon^{c_1})$, for some constant $l_1 > 0$ and update time of the form $O(\log^{l_2} n / \varepsilon^{c_2})$, for some constant $l_2 > 0$.

In this paper we show an ALTAS-LOG algorithm of order $(2, 2)$ such that given number k , $1 \leq k \leq \binom{|S|}{2}$ it outputs in $O(\log n)$ time a pair of points realizing distance which is the $(1 - \varepsilon^2, 1 + \varepsilon)$ -approximation (or $(1 - \varepsilon, 1 + \varepsilon^2)$ -approximation) of k^{th} distance. More precisely, we show how to construct a data structure in $O(n \log n / \varepsilon^2 + n \log^4 n / \gamma)$ time that dynamically maintains a set of n points in the plane in $O(\log^4 n / \gamma)$ time under insertions and deletions, for any fixed $\gamma > 0$, such that given number k , $1 \leq k \leq \binom{|S|}{2}$ one can compute in $O(\log n)$ time a pair of points realizing distance which is the $(1 - \gamma, 1 + \varepsilon)$ ($(1 - \varepsilon, 1 + \gamma)$)-approximation of k^{th} distance. We also show how to obtain dynamic $(1 - \varepsilon, 1 + \varepsilon)$ -approximation of k^{th} distance by simpler ALTAS-LOG algorithm of order $(2, 0)$ with slightly faster preprocessing time. It should be noted here that approximating the actual k -th distance within the factor $1 + \varepsilon^2$ (or $1 - \varepsilon^2$) is considerable harder than getting $1 + \varepsilon$ (resp. $1 - \varepsilon$) approximation with the same ε dependency in the running time of algorithm. We also generalize our algorithms to work in higher dimensions.

For our best knowledge, the dynamic problem of maintaining exact and approximate k^{th} distance is not studied in literature, except the famous closest pair problem (1^{st} distance selection) with optimal $O(\log n)$ worst-case update time [7] and diameter problem (farthest pair selection) with $O(n^\varepsilon)$ worst-case update time [19], expected $O(\log n)$ update time [20] and $O(b \log n)$ update time [25] that maintains the approximation diameter (the approximation factor depends on the integer constant $b > 0$). One may find our algorithms useful in parametric searching applications, where a set of candidate solutions is defined by the

distances between pairs of points of dynamic set S . For example, Agarwal and Procopiuc [3] (see also [2,14,29]) studied various k -center problems in R^d under L_∞ and L_2 metric: combinations of exact and approximate, continuous and discrete, uncapacitated and capacitated versions. Typically an algorithm performs a search (for example, binary search) on the sorted list of interdistances between data points. Our algorithms provide fast implementation of the search if an approximate solution suffices.

The main contribution of this paper is by developing efficient approximating dynamic algorithm for the well known distance selection problem using an approach that based on well separated pairs decomposition introduced by Callahan and Kosaraju [11] (see also [17]), computing approximate nearest and farthest neighbors [23,26] and persistent binary search trees introduced by Driscoll et al. [18].

This paper is organized as follows. In the next section we briefly describe well-separated pair decomposition. Section 3 is dedicated to the approximating dynamic distance selection problem. Finally we conclude in Section 4.

2 Well-Separated Pair Decomposition

In this section we shortly describe the well-separated pair decomposition proposed by Callahan and Kosaraju [11]. Let A and B be two sets of points in d -dimensional space ($d \geq 1$) of size n and m , respectively. Let s be some constant strictly greater than 0 and let $R(A)$ (resp. $R(B)$) be the smallest axis-parallel bounding box that encloses all the points of A (resp. B). We say that point sets A and B are *well-separated* with respect to s , if $R(A)$ and $R(B)$ can be each contained in d -dimensional ball of some radius r , such that the distance between these two balls is at least sr . One can easily show that for a given two well-separated sets A and B , if $p_1, p_4 \in A$, $p_2, p_3 \in B$ then $\text{dist}(p_1, p_2) \leq (1 + \frac{2}{s})\text{dist}(p_1, p_3)$ and $\text{dist}(p_1, p_2) \leq (1 + \frac{4}{s})\text{dist}(p_4, p_3)$. (For general L_p metric inequality may differ by some multiplicative constant.) Let S be a set of d -dimensional points, and let $s > 0$. A *well-separated pair decomposition* (WSPD) for S with respect to S is a set of pairs $\{(A_1, B_1), (A_2, B_2), \dots, (A_p, B_p)\}$ such that:

- (i) $A_i \subseteq S$ and $B_i \subseteq S$, for all $i = 1, \dots, p$.
- (ii) $A_i \cap B_i = \emptyset$, for all $i = 1, \dots, p$.
- (iii) A_i and B_i are well-separated with respect to s .
- (iv) for any two distinct points p and q in S , there is exactly one pair (A_i, B_i) such that either $p \in A_i$ and $q \in B_i$ or $p \in B_i$ and $q \in A_i$.

The main idea of the algorithm for construction WSPD is to build a binary fair split tree T whose leaves are points of S , with internal nodes corresponding to subsets of S . More precisely, split tree of S is a binary tree, constructed recursively as follows. If $|S| = 1$, its unique split tree consists of the node of S . Otherwise a split tree is any tree with root S and two subtrees that are split trees of the subsets formed by a split of S by an axis-parallel hyperplane into two non-empty subsets. For any node A in the tree, denote its parent (if exists)

by $p(A)$. The outer rectangle of A , denoted by $R(A)$ is either (if A is a root) an open d -cube centered at the center of the bounding box of S with the side size that equals to the largest side $l_{\max}(S)$ of the bounding box of S , or we have a situation when the splitting hyperplane used for the split of $p(A)$ divides $R(p(A))$ into two open rectangles. Let $R(A)$ be the one that contains A . A fair split of A is a split in which the splitting hyperplane is at distance of at least $l_{\max}(A)/3$ from each of the two boundaries of $R(A)$ parallel to it. A split tree formed using only fair splits is called a fair split tree.

Each pair (A_i, B_i) in WSPD is represented by two nodes $v, u \in T$, such that all the leaves in the subtree rooted at v correspond to the points of A_i and all the leaves in the subtree rooted at u correspond to the points of B_i . The paper of Callahan and Kosaraju [11] presents an algorithm that implicitly constructs WSPD for a given set S and separation value $s > 0$ in $O(n \log n + s^d n)$ time such that the number of pairs (A_i, B_i) is $O(s^d n)$. Moreover, Callahan [10] showed to compute WSPD in which at least one of the sets A_i, B_i of each pair (A_i, B_i) contains exactly one point of S . The running time remains the same; however, the number of pairs increases to $O(s^d n \log n)$.

3 Approximating k -th Distance

Our algorithm consists of several stages. At the first stage we compute a WSPD for S with separation constant $s = \frac{12}{\epsilon}$. From each (A_i, B_i) we take any pair $(a_i, b_i) \in (A_i, B_i)$, $1 \leq i \leq p$, $p = O(n)$. Our task now is to find the smallest index j in the sorted list of (a_i, b_i) pairs, such that the sum of cardinalities of all pairs (A_i, B_i) that correspond to this prefix is at least k . Therefore, we sort the distances d'_i between a_i and b_i , $1 \leq i \leq p$. We assume that the pairs (A_i, B_i) are in order of increasing d'_i . Next, for each pair (A_i, B_i) , $1 \leq i \leq p$, $p = O(n)$ we compute the $\alpha_i = |A_i||B_i|$ value, i.e. α_i is the total number of distinct pairs (a, b) , $a \in A_i$, $b \in B_i$. Let

- $m_i = \min_{a \in A_i, b \in B_i} \text{dist}(a, b)$.
- $M_i = \max_{a \in A_i, b \in B_i} \text{dist}(a, b)$ and

Let also l_i , $1 \leq i \leq p$ be a number such that $(1 - \gamma)M_i \leq l_i \leq M_i$, for arbitrary fixed $\gamma > 0$.

As we said above, for a particular k we compute the smallest j such that $\sum_{i=1}^j \alpha_i \geq k$. Let $M' = \max_{i=1}^j M_i$ and let $l' = \max_{i=1}^j l_i$. We claim that l' is the $(1 - \gamma, 1 + \epsilon)$ -approximation of k -th distance.

In what follows we prove the correctness of our algorithm and show how to implement it efficiently.

Lemma 1. $(1 - \gamma)d_k \leq l' \leq (1 + \epsilon)d_k$.

Proof. We observe that the total number of distances defined by pairs (A_i, B_i) , $1 \leq i \leq j$ is at least k because $\sum_{i=1}^j \alpha_i \geq k$. Since M' is the maximum of these distances $M' \geq d_k$ follows. Thus, from $l' \geq (1 - \gamma)M'$ it follows that $l' \geq (1 - \gamma)d_k$. Our goal now is to prove that $M' \leq (1 + \epsilon)d_k$. We recall that all possible pairs of points of S are uniquely represented by pairs (A_i, B_i) in

WSPD. Consider the set of pairs $D = \{(a, b) | a \in A_i, b \in B_i, i \geq j\}$. There is an index r , $j \leq r \leq p$ such that m_r is the smallest distance defined by pairs of D . The total number of pairs in D is larger than $\binom{n}{2} - k$. Therefore, $d_k \geq m_r$. Let t , $1 \leq t \leq j$ be the index such that $M' = M_t$. From the observation in previous section it follows that $M_t \leq (1 + \frac{2}{3})d'_t = (1 + \varepsilon/3)d'_t$. Thus, $M' \leq (1 + \varepsilon/3)d'_j \leq (1 + \varepsilon/3)d'_r$, since the sequence d'_i , $j \leq i \leq p$ is non-decreasing. It follows that $(1 + \varepsilon/3)d'_r \leq (1 + \varepsilon/3)(1 + \varepsilon/3)m_r \leq (1 + \varepsilon)d_k$. So, $l' \leq M' \leq (1 + \varepsilon)d_k$.

Remark 1. Using the similar approximation scheme with decreasing list of d'_i distances and by taking $M_i = \min_{a \in A_i, b \in B_i} \text{dist}(a, b)$ and l_i such $(1 + \gamma)M_i \geq l_i \geq M_i$ we can obtain $(1 - \varepsilon, 1 + \gamma)$ -approximation of the k^{th} distance.

Remark 2. If, instead of computing l_i , we choose d'_j as the value returned by the algorithm, we obtain $(1 - \varepsilon, 1 + \varepsilon)$ -approximation of the k^{th} distance. This is based on fact that $(1 + \varepsilon)d'_j = \max_{1 \leq i \leq j} (1 + \varepsilon)d'_i \geq \max_{1 \leq i \leq j} M_i = M' \geq d_k$.

It remains to show how to implement this algorithm efficiently, i.e. how to compute the values l_i, α_i , $1 \leq i \leq p$. First we show how to compute α_i . In other words we need to compute the cardinalities of A_i and B_i , $1 \leq i \leq p$. Recall that each pair (A_i, B_i) in WSPD is represented by two nodes v_i, u_i of the split tree T . The cardinality of $A_i(B_i)$ equals to the number of leaves in the subtree of T rooted at $v_i(u_i)$. Thus, by postorder traversal of T we are able to compute all the required cardinalities. Bspamyatnikh and Segal [9] showed how to compute the values m_i, M_i , $1 \leq i \leq p$ exactly using Voronoi diagrams [6] and Bentley's [5] logarithmic method. By assuming that the singleton set of each pair (A_i, B_i) in WSPD is $A_i = \{a_i\}$ they reduce the original problem of computing m_i and M_i values to the problem of computing for each a_i , $1 \leq i \leq p$ the nearest and the farthest neighbor in corresponding B_i . Since the computing of all Voronoi Diagrams may lead to undesired $O(n^2)$ runtime factor, they maintain dynamically Voronoi Diagrams while traversing a split tree T in a bottom-up fashion. Let S_v be a subset of S associated with a node v in T . By traversing a split tree T in a postorder fashion starting from leaves they use a partition R_v of S_v into disjoint sets $S_v^1, \dots, S_v^q$ and maintain the Voronoi Diagram VD with corresponding point location data structure PL for each set S_v^j , $1 \leq j \leq q$ in R_v . The sizes of the sets in R_v are different and restricted to be the powers of two. As the consequence the number of such sets is at most $\log n$, i.e. $q \leq \log n$. It can be shown that the total time needed to spend for all described operations is $O(n \log^3 n)$.

Dynamic Updates

The main drawback of the above scheme is the fact that during processing of T the Voronoi Diagram data structures are destroyed, so that at the end of the process we know only the Voronoi Diagram for the entire set S . Suppose that now we insert or delete some leaf from T . It may have influence on a number of other internal nodes. How we can determine now the new values of m_i and

M_i ? Basically, we have two major problems. The first one is how to store the Voronoi Diagram in each one of the internal nodes of T and the second one is how to update it quickly when T changes its structure by insertion of a new point or deletion of an existing point from T . In order to solve the first problem we will use a fully persistent binary search trees described by Driscoll et al. [18]. A fully persistent structure supports any sequence of operations in which each operation applied to any previously existing version. The result of the update is an entirely new version, distinct from all others. Unfortunately we cannot represent a Voronoi Diagram as a collection of a sublinear number of binary search trees and therefore, we need to find a way of computing the values m_i and M_i using another strategy. In fact we are interested in computing values l_i . Let us first consider the L_∞ metric. The points defining M_i should lie on the boundary of the smallest axis-parallel bounding box of set $A_i \cup B_i$. Recall that A_i and B_i are well separated and, thus, the L_∞ diameter of $A_i \cup B_i$ is defined by a pair (p, q) such that $p \in A_i$ and $q \in B_i$.

The computation of m_i , $1 \leq i \leq p$ can be done similarly to the approach described in [8]. Suppose we use a WSPD with $p = O(n \log n)$ and assume $A_i = \{a_i\}$, $1 \leq i \leq p$. For each point a_i we need to find the closest neighbor in corresponding B_i . Consider, for example, the planar case. Let l_1 be a line whose slope is 45° passing through the a_i and l_2 a line whose slope is 135° passing through the a_i . These lines define four wedges: $Q_{top}, Q_{bottom}, Q_{left}, Q_{right}$. For any point p lying in $Q_{left} \cup Q_{right} (Q_{bottom} \cup Q_{top})$ the L_∞ -distance to a_i is defined by the x -distance (y -distance, resp.) to a_i . We perform four range queries, using orthogonal range tree [6] data structure (in coordinate system defined by lines l_1, l_2), each of them corresponding to the appropriate wedge. For each node in a secondary data structure we keep four values $x_{min}, x_{max}, y_{min}, y_{max}$ (computed in the initial coordinate system) of points in corresponding range. Consider for a example wedge Q_{right} . Our query corresponding to Q_{right} marks $O(\log^2 n)$ nodes. The minimum of x_{min} values stored in these nodes define the closest neighbor point to a_i lying in Q_{right} . We proceed similarly with the other wedges. We maintain orthogonal range tree data structures dynamically in a bottom-up fashion while traversing split tree T . In order to merge two data structures we simply insert all the points stored in the smaller range tree into the larger one. However, we are interested in the values of m_i computed for the Euclidean metric. We will use the following two results in order to accomplish our task. The first result has been proposed by Kapoor and Smid [26] that finds, for a given query point $p \in \mathbb{R}^d$ a $(1 + \gamma)$ -approximate L_2 -neighbor of p in a given set of n points in $O(\log^{d-1} n / \gamma^{d-1})$ time using a data structure of $O(n \log^{d-2} n)$ space. They [26] store a set S in a constant number of a range trees, where each range tree stores the points according to its own coordinate system using the construction of Yao [32]. Then, for a given p , they use all the range trees to compute L_∞ nearest neighbors of p in all coordinate systems. One of these L_∞ neighbors is $(1 + \gamma)$ -approximate L_2 nearest neighbor of p . But we still need to compute the values of M_i . The second result is due to Indyk [23] that shows how to compute $(1 - \gamma)$ -approximate farthest neighbor of a given point p by performing a constant number of $(1 + \gamma)$ -approximate nearest neighbors queries. The idea is to construct a set of a constant number of concentric disks (balls)

around the origin. Each point is rounded to the nearest circle (sphere). For each disk (ball) we build an $(1 + \gamma)$ -approximate nearest neighbor data structure for the set of points on corresponding circle (sphere). Next, for each point $p \in S$ and each disk (ball) B_i , the “antipode” p^i of p with respect to B_i is defined as follows. Let p_1 and p_2 be the two points of the intersection of the circle (sphere) of B_i with the line passing through p and origin. Let h_p denote the hyperplane through the origin that is perpendicular to the line through p and origin. The point p^i is the one of the points p_1, p_2 which lies on the side of h_p different from the side containing p . In order to find the furthest neighbor of q , we issue $(1 + \gamma)$ -approximate nearest neighbor query with the point q^i in the data structure for the points on each one of the circles (spheres). Among the points found, we return the one furthest from q . Preprocessing time is $O(d^{O(1)}n)$ plus the cost of initiating a constant number of data structures for $(1 + \gamma)$ -approximate nearest neighbor queries. The query time is bounded by the query time for the $(1 + \gamma)$ -approximate nearest neighbor query.

The good thing in the described algorithms is the fact that all of them can be implemented using orthogonal range search trees, or in other words, binary search trees. This will allow us to make all of them fully persistent using Driscoll et al. [18] algorithm, thus solving our task of storing the appropriate data structure for each of the nodes of T without being destroyed. Generally speaking, ordinary data structures are ephemeral in sense that making a change to the structure destroys the old version, leaving only the new one. In a fully persistent data structure, past versions of the data structure are remembered and can be queried and updated. In [18] a method termed *node copying with displaced storage of changes* was developed that could make red-black tree data structure to become fully persistent, in worst-case time per operation of $O(\log n)$ and worst-case space cost of $O(1)$ per insertion or deletion. Instead of indicating a change to an ephemeral node x by storing the change in the corresponding persistent node x' , Driscoll et al. [18] stores information about the change in some possibly different node that lies on the access path to x' in the new version. Thus the record of the change is in general displaced from the node to which the change applies. The path from the node containing the change information to the affected node is called the *displacement path*. By copying nodes judiciously, Driscoll et al. [18] were able to keep the displacement paths sufficiently disjoint to guarantee an $O(1)$ worst-case space bound per insertion or deletion while having $O(\log n)$ worst-case time bound per access, insertion or deletion.

While traversing a tree T , we maintain all the described data structures for computing $(1 + \gamma)$ -approximate nearest neighbor and $(1 - \gamma)$ -approximate farthest neighbor. We use again Bentley’s [5] logarithmic method as described before. Notice, that each point in S can be inserted at most $O(\log n)$ times into the data structures while traversing T in a bottom-up fashion. Each insertion takes $O(\log^3 n)$ time. To give access to the persistent structure, the access pointers to the roots of the various versions must be stored in a balanced search tree, ordered by index. The total time for maintaining the range trees and computing l_i , $1 \leq i \leq p$ is $O(n \log^4 n)$, since $p = O(n \log n)$, each query takes $O(\log^2 n)$ time and each node contains a logarithmic number of the related data structures. The

above computation can also be generalized to d -dimensional space, $d > 2$. Thus, we have

Theorem 1. *Given a set S of n points in $\mathbb{R}^d$, a number k , $1 \leq k \leq \binom{n}{2}$, $\varepsilon > 0$, $\gamma > 0$ a pair of points realizing $(1 - \gamma, 1 + \varepsilon)$ $((1 - \varepsilon, 1 + \gamma))$ -approximation of d_k can be determined in $O(n \log n / \varepsilon^d + n \log^{d+2} n / \gamma^{d-1})$ time.*

Remark 3. Notice that we can obtain better running time (by logarithmic factor) using orthogonal range trees with the fractional cascading technique [16]. However, in order to allow persistence for the future dynamic updates we use orthogonal range trees avoiding this technique.

Remark 4. We can use a simpler strategy in order to compute the M_i values. We maintain the bounding boxes for sets of points corresponding to the nodes of T . The new bounding box can be computed in $O(1)$ time using the information from the previous steps. It results in a very fast algorithm with $(1 - \frac{1}{\sqrt{2}}, 1 + \varepsilon)$ -approximation of k^{th} distance which can be made dynamic fairly easy.

Remark 5. The runtime of the algorithm presented in [9] and the approximation factor achieved by that algorithm is better than in Theorem 1 for $d = 2$. Moreover, we should note that there is a more efficient algorithm even for $d > 2$. Instead of using Kapoor and Smid data structure [26] for querying approximate nearest neighbor, we can use either Kleinberg [28] or Indyk and Motwani [24] or Kushilevitz et al. [30] or Chan's [15] data structures for the same purpose. For example, using the result by Chan [15] that gave an ALTAS-LOG algorithm of order $(\frac{d-1}{2}, \frac{d-1}{2})$ that achieves $(1 + \varepsilon)$ -approximation for nearest neighbor query instead of Kapoor and Smid [26] ALTAS-LOG algorithm of order $(d - 1, d - 1)$ we obtain a better runtime of the entire algorithm. Unfortunately, the algorithm in [9] and also [15, 24, 28, 30] data structures cannot be made dynamic with a polylogarithmic update time. As we will see later, the result in Theorem 1 can be extended to deal with the dynamic point sets.

Following Remark 2 we also can conclude

Theorem 2. *Given a set S of n points in $\mathbb{R}^d$, a number k , $1 \leq k \leq \binom{n}{2}$, $\varepsilon > 0$, a pair of points realizing $(1 - \varepsilon, 1 + \varepsilon)$ -approximation of d_k can be determined in $O(n \log n / \varepsilon^d)$ time.*

It remains to check what happens with the tree T when a new point is inserted or some existing point is deleted. By $\sigma(v)$, $v \in T$ we denote the subset of points associated with v at some instance in the sequence of updates. If v has two children w_1 and w_2 then $\sigma(v) = \sigma(w_1) \cup \sigma(w_2)$. If v is a leaf, then $|\sigma(v)| = 1$. Since the fair split property depends on the value of $l_{max}(\sigma(v))$. Each time we insert a new point, this may increase the value of $l_{max}(\sigma(v))$ for all its ancestors in T , and the fair split property may be violated. Deletion of a point will not increase the value of $l_{max}(\sigma(v))$ for any of its ancestors, and hence can be performed on any fair split tree without restructuring. Callahan [10] shows that we can deal with the updates by maintaining a labeled binary tree T in which each node satisfies the following invariants:

1. For all internal nodes v with children w_1 and w_2 , there is a fair cut that partitions $R(v)$ into two rectangles R_1 and R_2 , such that $\sigma(w_1) = \sigma(v) \cap R_1$, $\sigma(w_2) = \sigma(v) \cap R_2$, $R(w_1)$ can be constructed from R_1 by applying a sequence of fair splits and $R(w_2)$ can be constructed from R_2 by applying a sequence of fair splits.
2. For all leaves v , $\sigma(v) = \{p\}$, and $R(v) = p$.

To insert a point p into this structure, we first retrieve the deepest internal node v in T such that $p \in R(v)$, ignoring the case in which p lies outside the rectangle at the root node. Let R_1 and w_1 have the same meaning as in the first invariant. Assume w.l.o.g. that $p \in R_1$. The way we chose v guarantees that $p \notin R(w_1)$. Now we introduce a new internal node u , which replaces w_1 as a child of v . We insert w_1 along with its subtree as a child of u , and insert a new leaf u' as the other child of u , where $\sigma(u') = \{p\}$. Finally we construct a rectangle $R(u)$ satisfying the first invariant. To delete the point p , we simply find a leaf v such that $\sigma(v) = \{p\}$, delete v , and compress out the internal node $p(v)$. Callahan [10] proves that once we have determined where to insert a point p , we may perform such an insertion in constant time, while preserving the invariants of the tree. Using the directed topology tree of Frederickson [21], Callahan has been able to maintain T in $O(\log n)$ time, where n is the current size of the point set. Generally speaking only $O(\log n)$ nodes of T can be affected during insertion or deletion of a point and therefore we can maintain the persistent structures associated with these nodes at sublinear cost.

Another problem that we have to deal with is the fact that introduction of a single new point can require the creation of many new pairs. Callahan [10] proposed an idea to predict all but a constant number of the new pairs ahead of time. The way to do it is to introduce *dummy points* where appropriate. Let $\tilde{S}$ be a set of dummy points. Such points will not be counted in $\sigma(v)$ for any $v \in T$, but the tree T will have the same structure and rectangle labels as a fair split tree of $S \cup \tilde{S}$. For efficiency we introduce only a constant number of dummy points for each well-separated pair $\{v, w\}$, such that $\sigma(v)$ and $\sigma(w)$ are not-empty. Since the number of new pairs is constant we can compute and maintain the relevant persistent structures efficiently.

The only missing thing is how to perform a query, i.e. how, for a given value of k , we can find the approximate k^{th} distance? We maintain a balanced binary search tree T' for distances d'_i as defined before. Suppose that we build a binary tree T' with the leaves corresponding to $d'_1, \dots, d'_p$. Each internal node $v \in T_r$ will keep three values: $\sum_{i=q_1}^{q_2} \alpha_i$, $\sum_{i=q_2+1}^{q_3} \alpha_i$, where $\alpha_{q_1}, \dots, \alpha_{q_2}$ ($\alpha_{q_2+1}, \dots, \alpha_{q_3}$) are the values that correspond to the leaves of the left subtree (resp. right subtree) of a tree rooted by v , and the third value $L_v = \max_{i=q_1}^{q_3} l_i$ (or $R_v = \min_{i=q_1}^{q_3} r_i$, $(1+\gamma)m_i \geq r_i \geq m_i$). Clearly, the construction of this tree T' with the augmented values can be computed in $O(p)$ time. We associate with each node $v \in T'$ an index j_v , such that d'_{j_v} corresponds to the rightmost leaf in the subtree rooted at v . Given a value k , we traverse T' starting from the root towards its children. We need to find a node u , with the smallest j_u such that $\sum_{i=1}^{j_u} \alpha_i \geq k$. It can be done in $O(\log n)$ time, by simple keeping the total number of nodes to the left of the current searching path. At each node where the path goes right, we

collect the value $L_v(R_v)$ stored in the left subtree. At the end, we report the maximal (minimal) of the collected $L_v(R_v)$ values. If T' is implemented as a balanced binary search tree then the update of the values r_i and l_i can be done in logarithmic time. Moreover, while updating T the new pairs may appear (and the previous pairs may disappear). Thus, we need to update the corresponding d'_i values in T' together with L_i, R_i, α_i values. The whole process in the plane can be accomplished in $O(\log^4 n)$ time since we have a logarithmic number of affected nodes in T , each query/update takes $O(\log^2 n)$ time and each node contains at most logarithmic number of associated data structures.

Therefore we can conclude by the following.

Theorem 3. *Given a set S of n points in $\mathbb{R}^d$, $\varepsilon > 0$, $\gamma > 0$ we can construct a data structure in time $O(n \log n / \varepsilon^d + n \log^{d+2} n / \gamma^{d-1})$ and $O(n \log n / \varepsilon^d)$ space with $O(\log^{d+2} n / \gamma^{d-1})$ update time for insertions/deletions of points such that given a number k , $1 \leq k \leq \binom{n}{2}$, a pair of points realizing $(1 - \gamma, 1 + \varepsilon)$ $((1 - \varepsilon, 1 + \gamma))$ -approximation of d_k can be determined in $O(\log n)$ time.*

Theorem 4. *Given a set S of n points in $\mathbb{R}^d$, $\varepsilon > 0$, we can construct a data structure in time $O(n \log n / \varepsilon^d)$ and $O(n / \varepsilon^d)$ space such that given a number k , $1 \leq k \leq \binom{n}{2}$, a pair of points realizing $(1 - \varepsilon, 1 + \varepsilon)$ -approximation of d_k can be determined in $O(\log n)$ time under insertions and deletions of points.*

4 Conclusions

We studied the dynamic problem for computing k -th Euclidean interdistance between n points in $\mathbb{R}^d$. The dynamization makes the problem more complicated. We are not aware of any other algorithms for exact or approximate solutions. We designed two efficient algorithms for maintaining a set of points and answering distance queries. The algorithms are based on the well-separated pair decomposition by Callahan and Kosaraju [11] and persistent data structures for approximate nearest/farthest neighbor. Both algorithms answer the queries in $O(\log n)$ time. The first algorithm provides $(1 - \varepsilon, 1 + \varepsilon)$ approximation and the second one provides a two parameters approximation $(1 - \varepsilon, 1 + \gamma)$ (or $(1 - \gamma, 1 + \varepsilon)$). It would be interesting to reduce the dependence of our algorithms on ε and γ .

References

1. P. Agarwal, B. Aronov, M. Sharir, S. Suri, "Selecting distances in the plane", *Algorithmica*, 9, pp. 495–514, 1993.
2. P. Agarwal, M. Sharir, E. Welzl "The discrete 2-center problem", *Proc. 13th ACM Symp. on Computational Geometry*, pp. 147–155, 1997.
3. P.K. Agarwal and C.M. Procopiuc, "Exact and Approximation Algorithms for Clustering", in *Proc. SODA '98*, pp. 658–667, 1998.
4. S. Arya and D. Mount, "Approximate range searching", in *Proc. 11th ACM Symp. on Comp. Geom.*, pp. 172–181, 1995.
5. J. Bentley, "Decomposable searching problems", *Inform. Process. Lett.*, 8, pp. 244–251, 1979.

6. M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, "Computational Geometry: Algorithms and Applications", Springer-Verlag, 1997.
7. S. Bespamyatnikh, "An Optimal Algorithm for Closest-Pair Maintenance", *Discrete Comput. Geom.*, 19, pp. 175–195, 1998.
8. S. Bespamyatnikh, K. Kedem, M. Segal and A. Tamir "Optimal Facility Location under Various Distance Function", in *Workshop on Algorithms and Data Structures'99*, pp. 318–329, 1999.
9. S. Bespamyatnikh and M. Segal "Fast algorithm for approximating distances", *Algorithmica*, 33(2), pp. 263–269, 2002.
10. P. Callahan "Dealing with higher dimensions: the well-separated pair decomposition and its applications", Ph.D thesis, Johns Hopkins University, USA, 1995.
11. P. Callahan and R. Kosaraju "A decomposition of multidimensional point sets with applications to k -nearest neighbors and n -body potential fields", *Journal of ACM*, 42(1), pp. 67–90, 1995.
12. P. Callahan and R. Kosaraju "Faster Algorithms for Some Geometric Graph Problems in Higher Dimensions", in *Proc. SODA'93*, pp. 291–300, 1993.
13. T. Chan "On enumerating and selecting distances", *International Journal of Computational Geometry and Applications*, 11, pp. 291–304, 2001.
14. T. Chan "Semi-online maintenance of geometric optima and measures", in *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, pp. 474–483, 2002.
15. T. Chan "Approximate nearest neighbor queries revised", *Discrete and Computational Geometry*, 20, pp. 359–373, 1998.
16. B. Chazelle and L. Guibas, "Fractional Cascading: I. A data structuring technique, II. Applications", *Algorithmica*, 1, pp. 133–162, 163–192, 1986.
17. S. Govindarajan, T. Lukovzski, A. Maheshwari and N. Zeh, "I/O Efficient Well-Separated Pair Decomposition and its Applications", In *Proc. of the 8th Annual European Symposium on Algorithms*, pp. 220–231, 2000.
18. J. Driscoll, N. Sarnak, D. Sleator and R. Tarjan "Making data structures persistent", *Journal of Computer and System Sciences*, 38, pp. 86–124, 1989.
19. D. Eppstein, "Dynamic Euclidean minimum spanning trees and extrema of binary functions", *Discrete and Computational Geometry*, 13, pp. 111–122, 1995.
20. D. Eppstein, "Average case analysis of dynamic geometric optimization", in *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 77–86, 1994.
21. G. Frederickson "A data structure for dynamically maintaining rooted trees", in *Proc. 4th Annu. Symp. on Disc. Alg.*, pp. 175–184, 1993.
22. M. Goodrich, "Geometric partitioning made easier, even in parallel", *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pp. 73–82, 1993.
23. P. Indyk, "High-dimensional computational geometry", Ph.D. thesis, Stanford University, pp. 68–70, 2000.
24. P. Indyk and R. Motwani "Approximate nearest neighbors: towards removing the curse of dimensionality", in *Proc. 30th ACM Symp. Theory of Comput.*, 1998.
25. R. Janardan, "On maintaining the width and diameter of a planar point-set online", *Int. J. Comput. Geom. Appls.*, 3, pp. 331–344, 1993.
26. S. Kapoor and M. Smid, "New techniques for exact and approximate dynamic closest-point problems", *SIAM J. Comput.*, 25, pp. 775–796, 1996.
27. M. Katz and M. Sharir, "An expander-based approach to geometric optimization", *SIAM J. Comput.*, 26(5), pp. 1384–1408, 1997.
28. J. Kleinberg "Two algorithms for nearest-neighbor search in high dimensions", in *Proc. 29th ACM Symp. Theory of Comput.*, pp. 599–608, 1997.
29. D. Krznaric "Progress in hierarchical clustering and minimum weight triangulation", *Ph. D. thesis*, Lund University, 1997.

30. E. Kushlevitz, R. Ostrovsky and Y. Rabani “Efficient search for approximate nearest neighbor in high dimensional spaces”, in *Proc. 30th ACM Symp. Theory of Comput.*, 1998.
31. J. Salowe, “ L_∞ interdistance selection by parametric searching”, *Inf. Process. Lett.*, 30, pp. 9–14, 1989.
32. A. C. Yao “On constructing minimum spanning trees in k -dimensional spaces and related problems”, in *SIAM Journal on Computing*, 11, pp. 721–736, 1982.

Solving the Robots Gathering Problem

Mark Cieliebak¹, Paola Flocchini², Giuseppe Prencipe³, and Nicola Santoro⁴

¹ ETH Zurich, cieliebak@inf.ethz.ch

² University of Ottawa, flocchini@site.uottawa.ca

³ University of Pisa, prencipe@di.unipi.it

⁴ Carleton University, santoro@scs.carleton.ca

Abstract. Consider a set of $n > 2$ simple autonomous mobile robots (decentralized, asynchronous, no common coordinate system, no identities, no central coordination, no direct communication, no memory of the past, deterministic) moving freely in the plane and able to sense the positions of the other robots. We study the primitive task of gathering them at a point not fixed in advance (GATHERING PROBLEM). In the literature, most contributions are simulation-validated heuristics. The existing algorithmic contributions for such robots are limited to solutions for $n \leq 4$ or for restricted sets of initial configurations of the robots. In this paper, we present the first algorithm that solves the GATHERING PROBLEM for *any* initial configuration of the robots.

1 Introduction

We consider a distributed system of autonomous mobile robots that are able to freely move in the two-dimensional plane. Due to their autonomy, the coordination mechanisms used by the robots to perform a task (i.e., solve a problem) must be totally *decentralized*, i.e., no central control is used. The problem we consider is *gathering* (or rendez-vous, or point-formation): all robots must gather at one point; the choice of the point is not fixed in advance.

Gathering is one of the basic interaction primitives in systems of autonomous mobile robots, and has been studied in robotics and in artificial intelligence [4,9,11]. Mostly, the problem is approached from an experimental point of view: algorithms are designed using mainly heuristics, and then tested either by means of computer simulations or with real robots. Neither proofs of correctness of the algorithms, nor any analysis of the relationship between the problem to be solved, the capabilities of the robots employed, and the robots' knowledge of the environment are given. Recently, concerns on computability and complexity of coordination problems have motivated *algorithmic* investigations, and the problems have also been approached from a *computational* point of view [2,7,8,12,14].

The solution to the GATHERING PROBLEM obviously depends on the capabilities of the robots. The research interest is on a very weak model of autonomous robots: the robots are anonymous (i.e., identical), have no common coordinate system, are oblivious (i.e., they do not remember previous observations and calculations), and have no means of direct communication. Initially, they are in

a waiting state. They wake up independently and asynchronously, observe the other robots' positions, compute a point in the plane, move towards this points (but may not reach it¹), and become waiting again. Details of the model are given in Section 2. For these robots, the GATHERING PROBLEM is defined as follows:

Definition 1. *Given n robots $r_1, \dots, r_n$, arbitrarily placed in the plane, with no two robots at the same position, make them gather at one point in a finite number of cycles.*

This GATHERING PROBLEM is *unsolvable* for such weak robots [13]; this is rather surprising considering the fact that a variety of other tasks (e.g. forming a circle) are solvable. Also, if the robots are asked only to move “very close” to each other, this task is easily solved: each robot computes the center of gravity² of all robots, and moves towards it.

The reason the same solution (i.e., moving towards the center of gravity) does not work for the GATHERING PROBLEM is because the center of gravity is not invariant with respect to robots' movements towards it. Recall that the robots act independently and asynchronously from each other, and that they have no memory of the past; once a robot makes a move towards the center of gravity, the position of the center of gravity changes; hence a robot (even the same one) observing the new configuration will compute and move towards a different point.

An obvious solution strategy would then be to choose as destination a point that, unlike the center of gravity, is *invariant* with respect to the robots' movements towards it. The only known point with such a property is the *Weber* (or *Fermat* or *Torricelli*) *point*: the unique point in the plane that minimizes the sum of the distances between itself and all positions of the robots [10,15]. This point does not change when moving any of the robots straight towards it. Unfortunately, it has been proven in [3] that the Weber point is not expressible as an algebraic expression involving radicals since its computation requires finding zeroes of high-order polynomials even for the case $n = 5$ (see also [6]). In other words, the Weber point is *not computable* by radicals for $n \geq 5$ [3], and thus it cannot be used to solve the GATHERING PROBLEM.

The problem becomes solvable if we change the nature of the robots: if we assume a common coordinate system, gathering is possible even with limited visibility [8]; if the robots are synchronous and movements are instantaneous, gathering has a simple solution [14] and can be achieved even with limited visibility [2]. On the other hand, without changing the robots' nature, they clearly must have some additional ability to solve the GATHERING PROBLEM. One such ability is *multiplicity detection*: a robot can detect whether at a point there is none, one, or more than one robot; if there is more than one robot, we say that

¹ That is, a robot can stop before reaching its destination point, e.g. because of limits to the robot's motion energy.

² For n points $p_1, \dots, p_n$ in the plane, the center of gravity is $c = \frac{1}{n} \sum_{i=1}^n p_i$.

there is *strict multiplicity* at that point. In the following, we will assume that the robots can detect multiplicities.

Even with multiplicity detection, the problem is surprisingly difficult and was, up to now, unsolved. It is actually unsolvable for $n = 2$ robots [13,14]. Simple solution algorithms exist for $n = 3$ and $n = 4$ robots. For $n \geq 5$ there are two *partial* solutions [5], i.e., algorithms that work for restricted sets of initial configurations. In particular, the first one works if the robots are initially in a biangular configuration (i.e., there exists a point c , and ordering of the robots, and two angles α, β such that the angles between adjacent robots w.r.t. c are either α or β , and the angles alternate; refer to Section 2 and Figure 2); the second algorithm works if in the initial scenario the positions of the robots do not form a regular n -gon (i.e., all robots are on a circle and the distances between each two adjacent robots are equal). Although the two sets of configurations together cover all possible input configurations, the two algorithms can not be integrated nor combined to solve the GATHERING PROBLEM in general.

In this paper, we present the first algorithm that solves the GATHERING PROBLEM for *any* initial configuration of the robots; all calculations performed by the robots can be computed by radicals. Due to space limitations, we only sketch the algorithm and the main ideas for its correctness. The complete algorithm and detailed proofs can be found in the full version of this paper.

2 Terminology, Notation, and Basic Tools

In this section, we introduce terminology and notation, and define the basic concepts used in our algorithm.

Autonomous Mobile Robots

A robot is a mobile computational unit provided with sensors, and it is viewed as a point in the plane. Once activated, the sensors return the set of all points in the plane occupied by at least one robot. In particular, for each such point the sensor outputs whether one or more than one robot is located there (*multiplicity detection*). This forms the current *local view* of the robot. The local view of each robot also includes a unit of length, an origin (which we assume w.l.o.g. to be the position of the robot in its current observation), and a coordinate system (e.g. Cartesian). There is no a priori agreement among the robots on the unit of length, the origin, or the coordinate systems.

A robot is initially in a *waiting* state (*Wait*). Asynchronously and independently from the other robots, it *observes* the environment (*Look*) by activating its sensors. The sensors return a snapshot of the world, i.e., the set of all points that are occupied by at least one other robot, with respect to the local coordinate system. The robot then *calculates* its destination point (*Compute*) according to its deterministic algorithm (the same for all robots), based only on its local view of the world. It then *moves* towards the destination point (*Move*); if the destination point is the current location, the robot stays still. A move may stop before

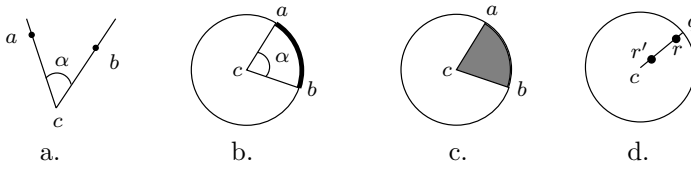


Fig. 1. (a) Convex angle $\alpha = \angle(a, c, b)$. (b) Arc (thick line) and (c) sector (grey part) defined by $\angle(a, c, b)$. (d) Two robots, r and r' , on the same radius.

the robot reaches its destination, e.g. because of limits to the robot's motion energy. The robot then returns to the waiting state. The sequence *Wait - Look - Compute - Move* forms a *cycle* of a robot.

The robots are *fully asynchronous*, i.e., the amount of time spent in each state of a cycle is finite but otherwise unpredictable. In particular, the robots do not have a common notion of time. As a result, robots can be seen by other robots while moving, and thus computations can be made based on obsolete observations. The robots are *oblivious*, meaning that they do not remember any observation nor computations performed in previous cycles. The robots are *anonymous*, meaning that they are a priori indistinguishable by their appearance, and they do not have any kind of identifiers that can be used during the computation. Finally, the robots have *no means of direct communication*: any communication occurs in a totally implicit manner, by observing the other robots' positions.

There are two limiting assumptions concerning *infinity*: **(A1)** The amount of time required by a robot to complete a cycle is not infinite, nor infinitesimally small. **(A2)** The distance traveled by a robot in a cycle is not infinite, nor infinitesimally small (unless it brings the robot to the destination point). As no other assumptions on space exist, the distance traveled by a robot in a cycle is unpredictable.

Notation and Definitions

Basic Notation. In general, r indicates any robot in the system (when no ambiguity arises, r is used also to represent the point in the plane occupied by that robot). A *configuration* of the robots at a given time instant t is the set of positions in the plane occupied by the robots at time t .

For the following definitions, refer also to Figure 1. Given two distinct points a and b in the plane, $[a, b)$ denotes the half-line that starts in a and passes through b , and $[a, b]$ denotes the line segment between a and b . Given two half-lines $[c, a)$ and $[c, b)$, we denote by $\angle(a, c, b)$ the convex angle (i.e., the angle that is at most 180°) centered in c and with sides $[c, a)$ and $[c, b)$. The intersection between the circumference of a circle $\mathcal{C}$ and an angle α at the center of $\mathcal{C}$ is denoted by $\text{arc}(\alpha)$, and the intersection between α and $\mathcal{C}$ is denoted by $\text{sector}(\alpha)$.

Given a circle $\mathcal{C}$ with center c and radius Rad , and a robot r , we say that r is *on* $\mathcal{C}$ if $\text{dist}(rc) = \text{Rad}$, where $\text{dist}(ab)$ denotes the Euclidean distance between

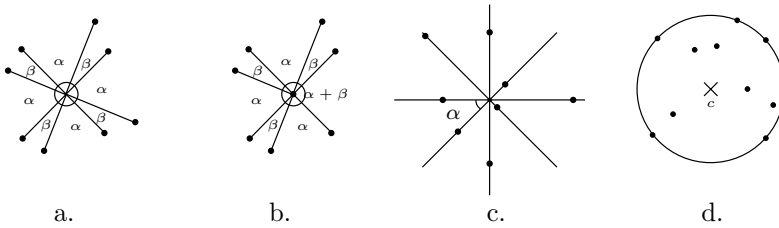


Fig. 2. (a) General biangular and (b) degenerated biangular configuration of 8 points. (c) General equiangular configuration. (d) The smallest enclosing circle of 10 points on the plane.

point a and b (i.e., r is on the circumference of $\mathcal{C}$); if $\text{dist}(rc) < \text{Rad}$, we will say that r is *inside* $\mathcal{C}$. Given two distinct robots r and r' , with r inside $\mathcal{C}$, let q be the intersection between the circumference of $\mathcal{C}$ and $[c, r)$. We say that r and r' are on the same radius if $r' \in [c, q]$.

Biangular Configurations. A set of n robots is in *general biangular configuration* if there exists a point b , the *center*, an ordering of the robots, and two angles $\alpha, \beta > 0$, such that each two adjacent robots form an angle α or β w.r.t. b , and the angles alternate (see Figure 2.a). The robots are in *degenerated biangular configuration* if there is a robot r , an ordering of the other robots, and two angles $\alpha, \beta > 0$, such that each two adjacent robots (without r) form an angle α or β w.r.t. r , and the angles alternate, except for one “gap” where the angle is $\alpha + \beta$ (see Figure 2.b). A general biangular configuration becomes degenerated if one of the robots, namely r , moves to the center b .

Similarly, we say that the robots are in a *general equiangular configuration* if there exists a point e , the *center*, an ordering of the robots, and an angle α such that each two adjacent robots form an angle α w.r.t. e (see Figure 2.c). Note that equiangular configurations can be “almost” considered a special case of biangular configurations: the only difference is that in a biangular configuration there is always an even number of robots, while in an equiangular configuration there can be an odd number of robots. Hence, from now on we will only refer to biangular configurations.

If a set of $n \geq 3$ points P is in general or degenerated biangular configuration, then the center of biangularity b is unique, can be computed in polynomial time, and is invariant under straight movement of any of the points in its direction; that is, it does not change if any of the points move towards b [1].

Smallest Enclosing Circles. Given a set of n distinct points P in the plane, the *smallest enclosing circle* of the points is the circle with minimum radius such that all points from P are inside or on the circle (see Figure 2.d). We denote it by $\text{SEC}(P)$, or SEC if set P is unambiguous from the context. The smallest enclosing circle of a set of n points is unique and can be computed in polynomial time [16].

Obviously, the smallest enclosing circle of P remains invariant if we remove all or some of the points from P that are inside $SEC(P)$. In fact, the following lemma shows that we can even remove all but at most three points from P without changing $SEC(P)$.

Lemma 1. *Given a set P of n points, there exists a subset $S \subseteq P$ such that $|S| \leq 3$ and $SEC(S) = SEC(P)$.*

String of Angles. Given n distinct points $p_1, \dots, p_n$ in the plane, let SEC be the smallest enclosing circle of the points, and c be its center. For an arbitrary point p_k , $1 \leq k \leq n$, we define the *string of angles* $SA(p_k)$ by the following algorithm (refer to Figure 3.a):

```

Compute-SA( $p_k$ )
 $p := p_k, i := 1;$ 
While  $i \neq n + 1$  Do
   $p' := \text{Succ}(p);$ 
   $SA[i] := \angle(p, c, p');$ 
   $p := p'; i := i + 1;$ 
Return  $SA$ .
  
```

Here, all angles are oriented clockwise (note that the robots do not have a common coordinate system; however, each robot can locally distinguish between a clockwise and counterclockwise orientation). The successor of p , computed by $\text{Succ}(p)$, is (refer to Figure 3.b)

- either the point $p_i \neq p$ on $[c, p]$, such that $\text{dist}(c, p_i)$ is minimal among all points $p_j \neq p$ on $[c, p]$ with $\text{dist}(c, p_j) > \text{dist}(c, p)$, if such a point exists; or
- the point $p_i \neq p$ such that there is no other point inside $\text{sector}(\angle(p, c, p_i))$, and there is no other point on the line segment $[c, p_i]$.

Instead of $SA(p_k)$, we write SA if we do not consider a specific point p_k . Given p_k , procedure $\text{Succ}()$ defines unique successors, and thus $\text{Compute-SA}(p_k)$ defines a unique string of angles. Given two starting points p_k and p_ℓ , then $SA(p_k)$ is a cyclic shift of $SA(p_\ell)$. Given an angle α in SA , then we can associate it with its defining point; i.e., if $\alpha = \angle(p, c, p')$, then we say that α is *associated* to p , and we write $p = \mathbf{r}(\alpha)$. Alternatively, since α is stored in SA , say at position i (i.e., $SA[i] = \alpha$), we denote the point associated to α by $\mathbf{r}(i)$, saying that $\mathbf{r}(i)$ is the point associated to position i in SA . We define the *reverse string of angles* $revSA$ in an analogous way: it is the string of angles with all angles counterclockwise oriented (i.e., $revSA$ is the reverse of SA). We say that SA (resp. $revSA$) is *general* if it does not contain any zeros; otherwise, at least two points are on a line starting in c (a radius), and we call the string of angles *degenerated*.

Given two strings $s = s_1, \dots, s_n$ and $t = t_1, \dots, t_n$, we say that s is *lexicographically smaller* than t if there exists an index $k \in \{1, \dots, n\}$ such that $s_i = t_i$ for all $1 \leq i < k$, and $s_k < t_k$. We write $s <_{lex} t$. Let LexMinString be the lexicographically minimal string among all strings of angles (in both orientations),

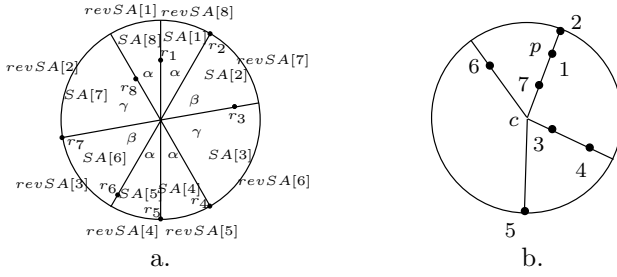


Fig. 3. (a) String of angles computed by `Compute_SA(r1)`. With $\alpha = 25^\circ$, $\beta = 60^\circ$ and $\gamma = 70^\circ$, we have $SA(r_1) = \langle \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma, \alpha \rangle = \langle 25^\circ, 60^\circ, 70^\circ, 25^\circ, 25^\circ, 60^\circ, 70^\circ, 25^\circ \rangle$, $LexMinString = \langle \alpha, \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma \rangle$, $\tau(SA[3]) = \tau(\gamma) = \tau(3) = r_3$, $StartSet = \{4, 8\}$, and $revStartSet = \emptyset$. (b) Routine `Succ(p)` with clockwise orientation. The points are numbered according to routine `Succ()`; that is `Succ(1)=2`, `Succ(2)=3`, and so on. Note that `Succ(7)=1`

i.e., $LexMinString := \min(\{SA(p_i) \mid 1 \leq i \leq n\} \cup \{revSA(p_i) \mid 1 \leq i \leq n\})$. Let $StartSet$ be the set of all indices in SA where $LexMinString$ starts, i.e., $StartSet = \{i \mid 1 \leq i \leq n, SA(p_i) = LexMinString\}$, and let $revStartSet$ be the set of all indices in $revSA$ where $LexMinString$ starts.

Robot Motion and Critical Points

In our algorithm, we use four different types of “move” operations; in each, when a robot moves, it moves in a straight line.

The basic operation is `moveTo(p)`, where a robot r moves towards point p (recall that, although restricted by assumption **A2**, the robot may enter the waiting state before reaching p).

In the operation `moveToIfFreeWay(p)`, the robot r moves towards p only if no other robot is between r and p ; otherwise, r does not move at all. This operation is used to avoid that the moving robot creates an (unintended) point with strict multiplicity. Note that, if all robots in the system are moving towards p and only this type of moves are executed, then strict multiplicity can only occur at p .

The remaining two types of movement are crucial to control the swap of a non-biangular configuration into a biangular one, due to robots’ movements. To introduce them, we need the notion of *critical points*, defined as follows:

Definition 2. Given n robots and a point p in the plane, a point x is a critical point for the movement of robot r towards p if $x = p$, or if x is on the half-line from p to r and the configuration of the robots becomes biangular when r is at position x .

A pair of points (y, z) is a critical pair for the movements of robots r' and r'' towards destinations p' and p'' respectively, if $(y, z) = (p', p'')$, or if y is on the

half-line from p' to r' , z is on the half-line from p'' to r'' , and the configuration of the robots is biangular when r' is at position y and r'' is at position z .

The operation `moveStepwiseTo( $p$ )` requires the robot r to first compute all critical points for its movement towards p , and then to move towards the first critical point on its way towards and stop there.

With operation `moveStepwiseTo( $(r', p')$ ,  $(r'', p'')$ )` we coordinate the movement of two robots r' and r'' which move in the direction of points p' and p'' , respectively. We compare the number of critical points between the robots and their destinations. The robot with most critical points ahead is allowed to move; if they have the same number, they both move. Once allowed, if the robot is between two critical points it moves to the next one; if it is already at a critical point it moves towards half the distance to the next critical point.

Finally, given a circle C with center c , we extend our four types of move operations and allow robots to move onto or away from C . In particular, we say that a robot moves to circle C (`moveTo( $C$ )`, `moveToIfFreeWay( $C$ )`, `moveStepwiseTo( $C$ )`) if the destination point of the robot is the intersection of C and the half-line starting from the center of C and going through the position of the robot (note that the robot does not move at all if it is already at this intersection point). Moreover, we define what a movement into the inside of circle C is (`moveTo(into  $C$ )`, `moveToIfFreeWay(into  $C$ )`, `moveStepwiseTo(into  $C$ )`): if the robot is already inside C , it does not move at all. Otherwise, it moves to the point p that is half on its way towards c , the center of the circle. For two robots r' and r'' , we define `moveStepwiseTo( $r'$ ,  $r''$ ,  $C$ )` and `moveStepwiseTo( $r'$ ,  $r''$ , into  $C$ )` accordingly.

3 The Solution Algorithm

In this section, we describe the algorithm that solves the GATHERING PROBLEM for arbitrary initial configurations of $n \geq 5$ robots, and discuss its correctness.

3.1 Description

At a high level, the strategy of the algorithm is as follows. Initially all robots are in distinct locations; that is, in the initial configuration, there is no point with strict multiplicity. Our algorithm ensures that at any time during the execution there is at most one point with strict multiplicity; moreover, such a point will eventually be generated. Once this occurs, the robots that are already at that point remain there, while all other robots move towards this unique point.

If the (initial) configuration is biangular, then all robots move towards the center of biangularity. The future configuration remains biangular until two (or more) robots reach the center. When this occurs, a unique point with strict multiplicity has been created.

In all other configurations, we select a strict subset of the robots; the selection is done using the string of angles of the robots w.r.t. the center of their smallest

enclosing circle. If we can elect a unique robot, it will go to some other robot creating a unique point with strict multiplicity. Otherwise, the selected robots move towards the center of the smallest enclosing circle, ensuring that the circle does not change because of these movements. If no biangular configuration is created during these movements, two (or more) robots reach the center of the circle, and we have a unique point with strict multiplicity.

One of the difficult and crucial components of the algorithm is the use of appropriate move operations to ensure the following: if a biangular configuration is created during the movements of some robots, then *all* robots have to become aware of it in their next *Look* state, ensuring that they will gather at the center of biangularity. The difficulty arises from the asynchrony, obliviousness and autonomy of the robots; the component is crucial to avoid that some robots move towards the center of biangularity while others still move towards the center of the circle (possibly destroying biangularity).

The main algorithm is shown in Algorithm 1. In the algorithm we use four different subroutines; their behavior differs depending on the value of s , the cardinality of the set $StartSet \cup revStartSet$ (therefore, s denotes the number of starting positions of *LexMinString* in SA and $revSA$).

Algorithm 1 Algorithm GATHER

```

 $\mathcal{Z} :=$  Observed Configuration;
 $SEC :=$  Smallest Enclosing Circle of all robots;
 $c :=$  Center of  $SEC$ ;
 $InnerC :=$  Circle with center  $c$  and radius  $\frac{\text{radius of } SEC}{2}$ ;
5: Case  $\mathcal{Z}$  Is Such That:
    • There is One point  $m$  with strict multiplicity:
        moveToIfFreeWay( $m$ ).
    • The robots are in general (resp. degenerated) biangular configuration:
         $b :=$  Center of general (resp. degenerated) biangularity;
10:    moveToIfFreeWay( $b$ ).
    • default:
        If No robot is at  $c$  Then
             $SA := (\text{Compute\_SA});$  %String of angles of all robots%
             $StartSet, revStartSet :=$  Indices Where Lex. Minimal String Starts;
15:             $s := |StartSet \cup revStartSet|;$ 
            If  $SA$  is general Then Routine1. Else Routine2.
            Else %One robot  $r$  is at  $c$ %
                 $r :=$  robot at  $c$ ;
                 $SA^- :=$  String of angles of all robots except  $r$ ;
20:             $StartSet^-, revStartSet^- :=$  Indices Where Lex. Minimal String Starts;
            If  $SA^-$  is general Then Routine3. Else Routine4.
  
```

In the following, we first discuss the main properties of *LexMinString*, and then we sketch the correctness proof of the algorithm.

3.2 Properties of *LexMinString*

1. *One Starting Position of LexMinString* ($s = 1$): Let $StartSet \cup revStartSet = \{x\}$ and $SA(x) = \alpha_1, \dots, \alpha_n$; then $revSA(x) = \alpha_n, \dots, \alpha_1$, and the following holds:

Lemma 2. *If $StartSet \cup revStartSet = \{x\}$, then either $SA(x) = LexMinString$ or $revSA(x) = LexMinString$.*

This implies that there is a unique starting position and a unique direction for *LexMinString*, yielding a unique ordering of the robots. If all robots are on *SEC*, then we can use this ordering and Lemma 1 to define operation **ElectOne()** to elect the first robot r such that *SEC* remains invariant if r is moved to the inside of *SEC*. If more than one robot is inside *SEC*, then **ElectOneInside()** is used to elect a unique robot that is already inside *SEC* (again, using the uniqueness of *LexMinString*).

2. *Two Starting Positions of LexMinString* ($s = 2$): Let $StartSet \cup revStartSet = \{x, y\}$. The following lemma shows that *LexMinString* can start in each position in only one direction.

Lemma 3. *If $StartSet \cup revStartSet = \{x, y\}$, then it is not possible that $SA(x) = revSA(x) = LexMinString$ or $SA(y) = revSA(y) = LexMinString$.*

If *LexMinString* starts in x and y in the same direction, then the angle between these two positions w.r.t. c is 180° . Moreover, for every robot there is a partner such that their angle is 180° . Recall that $\mathbf{r}(x)$ is the robot associated with index x . Using the starting positions and the direction of *LexMinString*, we define **ElectTwo()** as follows: if $\mathbf{r}(x)$ and $\mathbf{r}(y)$ are on *SEC*, then we elect the “next” pair of robots with an angle of 180° ; otherwise we elect $\mathbf{r}(x)$ and $\mathbf{r}(y)$ themselves.

If *LexMinString* starts in x and y in opposite direction, say $x \in StartSet$ and $y \in revStartSet$, then let γ be the angle between $\mathbf{r}(x)$ and $\mathbf{r}(y)$ w.r.t. c . If $\gamma = 180^\circ$, then **ElectTwo()** elects the first two robots, according to the starting positions and directions of *LexMinString*, that are not both on *SEC*. If $\gamma < 180^\circ$, we define the *opposite robots* of $\mathbf{r}(x)$ and $\mathbf{r}(y)$ to be one or two robots in the half of *SEC* where $\mathbf{r}(x)$ and $\mathbf{r}(y)$ are not (see Figure 4): let ℓ be the line that bisects γ . Then ℓ is a symmetry line for the angles of the robots w.r.t. c . We choose either the robot r that is on line ℓ , if such a robot exists, or we choose the two robots u and v that are closest to ℓ (in terms of their angles w.r.t. c). Observe that the construction of the opposite robots guarantees that c is inside the convex hull of $\mathbf{r}(x)$, $\mathbf{r}(y)$ and their opposite robot(s). Thus, **ElectTwo()** can elect two appropriate robots such that *SEC* remains invariant if they move (using Lemma 1).

Finally, we define routine **ElectPairInside()** that elects the “first” pair of robots that is inside *SEC*. Again, the ordering of the robots is given by the starting positions and orientations of *LexMinString*.

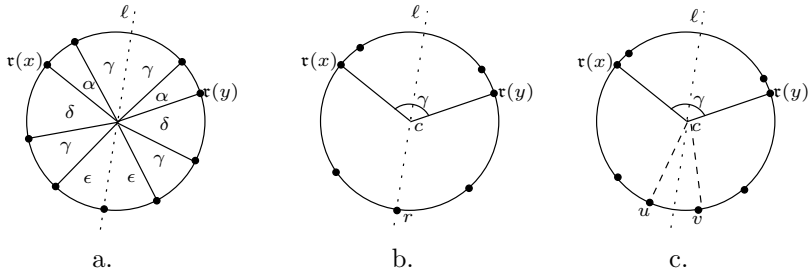


Fig. 4. (a) The line ℓ that runs through c and bisects $\gamma = \angle(\tau(x), c, \tau(y)) = 2\epsilon + 2\gamma$ is a symmetry axis for the angles that the robots form w.r.t. c . In the depicted example, $x \in \text{StartSet}$, $y \in \text{revStartSet}$, and $\text{SA}(x) = \text{revSA}(y) = \text{LexMinString} = \langle \alpha, \gamma, \gamma, \alpha, \delta, \gamma, \epsilon, \epsilon, \gamma, \delta \rangle$. (b) One robot r opposite to $\tau(x)$ and $\tau(y)$. (c) Two robots u and v opposite to $\tau(x)$ and $\tau(y)$.

3. *Many Starting Positions of LexMinString* ($s > 2$): Let $\text{StartSet} = \{x_1, \dots, x_l\}$. Then SA and revSA are periodic. Moreover, if k is the minimum length of a period of SA , then we can divide SA into $\frac{n}{k}$ equal periods, and the angles in each period sum up to $\gamma = \frac{360^\circ}{\frac{n}{k}}$. If the period length is one or two, then the configuration is biangular; hence we can exclude this case in the following, since it is covered in Lines 8–10 of the main algorithm.

We say that two robots r and r' are *equivalent (modulo periodic shift)* if they have the same position in different periods, i.e., if $\angle(r, c, r')$ is a multiple of γ (see the example depicted in Figure 5). If all robots are on SEC , then for any robot r , there are $\frac{n}{k} - 1$ equivalent robots, and they form a regular $\frac{n}{k}$ -gon with c inside. Thus, if at least one robot and all its equivalent robots remain on SEC , then SEC remains invariant (using Lemma 1).

Lemma 4. *If $\text{StartSet} \neq \emptyset$, all robots are on SEC , and the minimum period length of SA is $k \geq 3$, then SEC remains invariant when all robots $\tau(x)$, with $x \in \{\text{StartSet} \cup \text{revStartSet}\}$, move inside SEC .*

Observe that if all robots are on SEC , then equivalent robots cannot be distinguished, hence they act in the same way. In the case of one or two starting positions of *LexMinString*, we were able to elect one or two robots to move, and we used stepwise movements to ensure that these robots stop when the configuration becomes biangular. If there are many starting positions of *LexMinString*, we do not need to apply stepwise movements, as shown by the following lemma.

Lemma 5. *Given a non biangular configuration of the robots such that SA is periodic, then moving any subset of the robots towards c cannot make the configuration become biangular.*

To see this, observe that c is the Weber point of the robots, and that the center of biangularity, if it exists, is the Weber point as well. Thus, since the

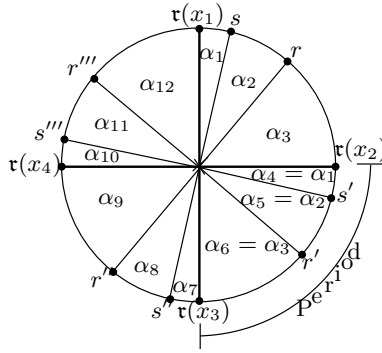


Fig. 5. Example with $|StartSet|=4$, $SA(x_1) = SA(x_i) = LexMinString = \alpha_1, \dots, \alpha_{12}$, and the period of $SA(x_1)$ is $\langle \alpha_1, \alpha_2, \alpha_3 \rangle$. There are $\frac{n}{k} = \frac{12}{3}$ periods, and $\gamma = \alpha_1 + \alpha_2 + \alpha_3 = \frac{360^\circ}{4}$. Thick lines represent the starting points of each of the four periods. Robots r, r', r'' and r''' are *equivalent*, as well as $\tau(x_1), \tau(x_2), \tau(x_3)$ and $\tau(x_4)$, and s, s', s'' and s''' .

Weber point is unique, the robots cannot swap into a biangular configuration if there was none before. This lemma implies that the robots cannot create a biangular configuration while they move towards or away from c , hence we do not need to introduce a stepwise movement in this case.

Correctness Proof (Sketch)

The first thing a robot does when it starts its computation is to check whether there is a point p in the plane with strict multiplicity. If this is the case, the robot simply moves there. Point p will be the final gathering point (Lines 6–7).

Otherwise, the robots check whether the observed configuration of the robots is biangular. In this case, the center of biangularity b is computed, and the robots move there using `moveToIfFreeWay(b)`. As long as none of the robots reaches b , the configuration remains general biangular; hence the algorithm continues to move all robots towards b . By Assumptions A1 and A2, in a finite number of cycles, at least one robot reaches b . If only one robot reaches b , then the configuration becomes degenerated biangular. In this case, the center of degenerated biangularity³ is again b , and all robots continue moving towards b . As soon as two robots reach b , there is a unique point with strict multiplicity, and all robots will gather there.

If the observed configuration is not biangular, then the *SEC* and its center c are computed. The algorithm distinguishes four cases.

³ If a general biangular configuration with center b turns into a degenerated biangular configuration because one of the robots reaches b , then the center of the degenerated biangular configuration is again b .

- (A) **There is no robot at c , and SA is general.** **Routine1** is called, which behaves differently depending on the value of s , the cardinality of $StartSet \cup revStartSet$.

If $s = 1$, then a unique robot r is elected, and it moves stepwise⁴ towards c . Robot r is chosen such that SEC does not change during its movement. When the movement stops, either the configuration is biangular, and Line 8 of the main algorithm applies; or **Routine1** is called again (with r – the only robot inside SEC – again elected), until r reaches c , and **Routine3** applies.

If $s = 2$, at first all robots that are inside SEC move to the circumference of SEC (by repeatedly calling **ElectPairInside()**). Afterwards, only the two robots elected by **ElectTwo()** are allowed to move, and they move towards c . All movements are stepwise, and there are always at most two moving robots, either the robots run into a biangular configuration and stop (Line 8 of the main algorithm then applies), or one of them reaches c and **Routine3** is called, or the two elected robots reach c simultaneously, and c becomes the unique point with strict multiplicity. In the last two cases, c will be the final gathering point.

If $s > 2$, first all robots associated to indices in $StartSet \cup revStartSet$ are elected. Then, all robots that are not elected and that are inside SEC are moved towards the circumference of SEC . Afterwards, all elected robots (and only these) move towards⁵ c (without changing SEC , by Lemma 4), with the only restriction that an elected robot can reach c only if all other elected robots are already inside SEC (note that two robots inside SEC would be sufficient). This is achieved by first calling routine **moveTo(into C)**. In a finite number of cycles at least two robots reach c , creating strict multiplicity there.

- (B) **There is no robot at c , and SA is degenerated.** **Routine2** is called. Recall that, if SA is degenerated, then there is at least one radius of SEC with more than one robot on it. Therefore, due to our definition of SA , the lexicographically minimal string of angles always start with zeros. Moreover, on each radius with at least two robots, one robot is already inside SEC . Similarly to previous cases, different actions are taken depending on the value of s .

If $s = 1$, then the subroutine elects a unique radius rad that has at least two robots lying on it. Let $StartSet = \{x\}$ (the case $revStartSet = \{x\}$ is handled similarly), and rad_x be the radius where $\mathbf{r}(x)$ lies (i.e., $[c, \mathbf{r}(x)]$). Then rad can be chosen as the first radius with at least two robots on it, starting from rad_x and according to the ordering of the robots established by SA . Let r and r' be the first two innermost robots on rad . Then r moves stepwise towards r' , while all other robots do not move. In a finite number of cycles, either a biangular configuration is reached (r stops at the first critical

⁴ Recall that stepwise movement implies that r stops at its first critical point.

⁵ Some of them can already be inside SEC , and others are still on the circumference of SEC .

point on its path towards r') and Line 8 of the main algorithm applies, or r reaches r' and a unique point with strict multiplicity is created.

If $s = 2$, the algorithm works similar to Case (A), except that all operations are done with respect to *InnerC* instead of *SEC*. In particular, first the robots that are inside *InnerC* move out of *InnerC*. If we would move these robots simply to the circumference of *InnerC*, we could obtain unintended points with strict multiplicity, since all robots on the same radius would end up at the same point on *InnerC*. Therefore, we define a sufficient number of distinct positions "just outside" *InnerC* (using the radius of *SEC*) where we move the robots that are on the same radius. Thereby, we ensure that the innermost robots will end up on *InnerC*. Afterwards, the two robots elected by *ElectTwo*() are allowed to move, and they move stepwise towards c , and in a finite number of cycles at least one of them reaches c .

If $s > 2$, then *SA* is periodic (see paragraph on $s > 2$). Again, the algorithm works similar to Case (A), except that all operations are done with respect to *InnerC* instead of *SEC*.

- (C) **There is exactly one robot r at c , and SA^- (the string of angles of all robots except r) is general.** *Routine3* is called. If r is the only robot inside *SEC*, then r chooses an arbitrary robot q on *SEC* and moves stepwise towards it. By this movement, the string of angle becomes degenerated, since r and q are on the same radius. Hence, by (B), r continues to move towards q . If no critical points are on its path towards q , in a finite number of cycles r reaches q and a unique point with strict multiplicity is obtained. Otherwise, r stops at the first critical point it meets. Then a biangular configuration is obtained, and Line 8 of the main algorithm applies.

If there are only two robots r and r' inside *SEC* (with r at c), then r' moves stepwise towards c . The argument follows similarly to the previous paragraph.

If more than two robots are inside *SEC* and SA^- is periodic except for one gap⁶, then all robots inside *SEC* move towards c . By Lemma 5, no biangularity can occur.

If more than two robots are inside *SEC* and SA^- is not periodic except for one gap, then the routine behaves similarly to *Routine1*. The only difference is that in this case all the operations are done using SA^- instead of *SA*; that is, robot r is ignored.

- (D) **There is exactly one robot r at c , and SA^- (the string of angles of all robots except r) is degenerated.** *Routine4* is called. If r is not the only robot inside *InnerC*, then this routine is similar to *Routine3*, except that all operations refer to *InnerC* instead of *SEC*. Otherwise, if r is the only robot inside *InnerC*, then it chooses some an arbitrary index q in $StartSet^- \cup revStartSet^-$. Note that q is always associated to a position in *SA* where *LexMinString* starts. Robot r moves stepwise towards $\mathbf{r}(q)$, while all other robots do not move. As soon as r leaves c , a unique starting

⁶ That is, the string would be periodic if r — the robot at c — would not be at c , but somewhere inside *SEC*.

position of *LexMinString* is obtained in the positions associated to r , since an angle with 0° has been added. Thus, SA is degenerated with no robot at c , and r will be chosen again to move on to q due to Case (B) above.

4 Conclusion

We have presented a deterministic algorithm for the GATHERING PROBLEM for $n \geq 5$ robots that works for all initial configurations of the robots. Some interesting questions are still open. For example, it is not known which other abilities, other than multiplicity detection, would allow the weak robots to solve the GATHERING PROBLEM.

It is known that changing the nature of the robots (e.g. by synchronizing them, or by adding common knowledge on the coordinate system) enables solvability. It is still not known if (and how) removing obliviousness would have the same effect. It would be interesting to explore the relationship between memory and solvability or, for that matter, to study the impact of (weak) explicit communication among the robots.

Acknowledgments. We would like to thank all people that have offered their ideas, comments, suggestions, and (conflicting) conjectures on this problem over the years. Especially, we would like to thank Elmo Welzl and Peter Widmayer.

References

1. L. Anderegg, M. Cieliebak, and G. Prencipe. A Linear Time Algorithm to Identify Equiangular and Biangular Point Configurations. Technical Report TR-03-01, Università di Pisa, 2003.
2. H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. A Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility. *IEEE Transaction on Robotics and Automation*, 15(5):818–828, 1999.
3. C. Bajaj. The Algebraic Degree of Geometric Optimization Problems. *Discrete and Computational Geometry*, 3:177–191, 1988.
4. T. Balch and R. C. Arkin. Behavior-based Formation Control for Multi-robot Teams. *IEEE Transaction on Robotics and Automation*, 14(6), 1998.
5. M. Cieliebak and G. Prencipe. Gathering Autonomous Mobile Robots. In *SIROCCO 9*, pages 57–72, 2002.
6. E. J. Cockayne and Z. A. Melzak. Euclidean Constructibility in Graph-minimization Problems. *Mathematical Magazine*, 42:206–208, 1969.
7. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In *ISAAC '99*, volume 1741 of *LNCS*, pages 93–102, 1999.
8. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Autonomous Mobile Robots With Limited Visibility. In *STACS 2001*, volume 2010 of *LNCS*, pages 247–258, 2001.
9. D. Jung, G. Cheng, and A. Zelinsky. Experiments in Realising Cooperation between Autonomous Mobile Robots. In *ISER*, 1997.

10. Y. Kupitz and H. Martini. Geometric Aspects of the Generalized Fermat-Torricelli Problem. *Intuitive Geometry*, 6:55–127, 1997.
11. M. J. Matarić. Designing Emergent Behaviors: From Local Interactions to Collective Intelligence. In *From Animals to Animats 2: Int. Conf. on Simulation of Adaptive Behavior*, pages 423–441, 1993.
12. G. Prencipe. CORDA: Distributed Coordination of a Set of Autonomous Mobile Robots. In *ERSADS 2001*, pages 185–190, 2001.
13. G. Prencipe. Instantaneous Actions vs. Full Asynchronicity: Controlling and Coordinating a Set of Autonomous Mobile Robots. In *ICTCS 2001*, pages 185–190, 2001.
14. I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *Siam Journal of Computing*, 28(4):1347–1363, 1999.
15. E. Weiszfeld. Sur le Point Pour Lequel la Somme Des Distances de n Points Donnés Est Minimum. *Tohoku Mathematical*, 43:355–386, 1936.
16. E. Welzl. Smallest Enclosing Disks (Balls and Ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, LNCS, pages 359–370. Springer, 1991.

Author Index

- Ageev, Alexander 145
Agrawal, Aseem 527
Albers, Susanne 653
Alfaro, Luca de 1022, 1038
Amir, Amihood 929
Ancona, Davide 224
Antunes, Luís 267
Ariola, Zena M. 871
Arora, Sanjeev 176
Aumann, Yonatan 929
Awerbuch, Baruch 1153
- Bansal, Vipul 527
Baswana, Surender 384
Berger, Noam 725
Bergstra, Jan A. 1
Bespamyatnikh, Sergei 1169
Bethke, Inge 1
Bhatia, Randeep 751
Bläser, Markus 157
Bleichenbacher, Daniel 97
Blom, Stefan 109
Blondel, Vincent D. 739
Bodirsky, Manuel 1095
Bollobás, Béla 725
Bonis, Annalisa De 81
Borgs, Christian 725
Boros, Endre 543
Brinkmann, André 1153
Bruni, Roberto 252
Bugliesi, Michele 969
Busi, Nadia 133
- Cachat, Thierry 556
Campenhout, David Van 857
Carayol, Arnaud 599
Chang, Kevin L. 176
Chattopadhyay, Arkadev 984
Chayes, Jennifer 725
Chekuri, Chandra 189, 410
Chen, Jianer 845
Chuzhoy, Julia 751
Cieliebak, Mark 1181
Coja-Oghlan, Amin 200
Colcombet, Thomas 599
- Cole, Richard 929
Condon, Anne 22
Crafa, Silvia 969
Crescenzi, Pilu 1108
- Dang, Zhe 668
Dean, Brian C. 1138
Demaine, Erik D. 829
Denis, François 452
Doberkat, Ernst-Erich 996
Dooren, Paul Van 739
Droste, Manfred 426
- Eisner, Cindy 857
Elbassioni, Khaled 543
Elkin, Michael 212
Esposito, Yann 452
Even-Dar, Eyal 502
- Faella, Marco 1038
Fagorzi, Sonia 224
Feldmann, Rainer 514
Fiala, Jiří 817
Fiat, Amos 33
Fisman, Dana 857
Flocchini, Paola 1181
Fokkink, Wan 109
Fomin, Fedor V. 829
Fortnow, Lance 267
Fotakis, Dimitris 637
Franceschini, Gianni 316
Freund, Ari 751
- Gabbrielli, Maurizio 133
Gairing, Martin 514
Gál, Anna 332
Gambosi, Giorgio 1108
Gandhi, Rajiv 164
Gargano, Luisa 802
Gąsieniec, Leszek 81
Goemans, Michel X. 1138
Gorla, Daniele 119
Gröpl, Clemens 1095
Grossi, Roberto 316
Guha, Sudipto 189
Gurvich, Vladimir 543

- Gutiérrez, Francisco 956
 Hajiaghayi, Mohammad Taghi 829
 Hall, Alex 397
 Halperin, Eran 164
 Hammar, Mikael 802
 Hannay, Jo 903
 Havlicek, John 857
 Henzinger, Thomas A. 886, 1022
 Herbelin, Hugo 871
 Hippler, Steffen 397
 Hitchcock, John M. 278
 Holzer, Markus 490
 Høyer, Peter 291
 Hromkovič, Juraj 66, 439

 Ibarra, Oscar H. 668
 Ikeda, Satoshi 1054

 Jägersküpper, Jens 1068
 Jain, Rahul 300
 Jhala, Ranjit 886
 Johannsen, Jan 767

 Kärkkäinen, Juha 943
 Kang, Mihyun 1095
 Kanj, Iyad A. 845
 Kesselman, Alex 502
 Khachiyan, Leonid 543
 Khuller, Samir 164
 Kiayias, Aggelos 97
 Klaedtke, Felix 681
 Korman, Amos 369
 Kortsarz, Guy 164, 212
 Kubo, Izumi 1054
 Kupferman, Orna 697
 Kuske, Dietrich 426
 Kutrib, Martin 490

 Lange, Martin 767
 Lewenstein, Moshe 929
 Lücking, Thomas 514
 Lutz, Jack H. 278

 Majumdar, Rupak 886, 1022
 Makino, Kazuhisa 543
 Malhotra, Varun S. 527
 Mansour, Yishay 502
 Matias, Yossi 918
 Mayordomo, Elvira 278

 Mayr, Richard 570
 McIsaac, Anthony 857
 Merro, Massimo 584
 Meseguer, José 252
 Miltersen, Peter Bro 332
 Moggi, Eugenio 224
 Monien, Burkhard 514
 Moore, Christopher 200
 Mosca, Michele 291
 Munro, J. Ian 345
 Mutzel, Petra 34
 Mydlarz, Marcelo 410

 Nain, Sumit 109
 Naor, Joseph 189, 751, 1123
 Napoli, Margherita 776
 Nicosia, Gaia 1108

 Okhotin, Alexander 239
 Okumoto, Norihiro 1054

 Paepe, Willem E. de 624
 Parente, Mimmo 776
 Parlato, Gennaro 776
 Paulusma, Daniël 817
 Peled, Doron 47
 Peleg, David 369
 Penna, Paolo 1108
 Perković, Ljubomir 845
 Porat, Ely 918, 929
 Poulalhon, Dominique 1080
 Prelic, Amela 969
 Prencipe, Giuseppe 1181
 Pugliese, Rosario 119

 Rabinovich, Alexander 1008
 Radhakrishnan, Jaikumar 300
 Raman, Rajeev 345, 357
 Raman, Venkatesh 345
 Rao, S. Srinivasa 345, 357
 Riordan, Oliver 725
 Rode, Manuel 514
 Rueß, Harald 681
 Ruiz, Blas 956
 Rybina, Tatiana 714

 Sanders, Peter 943
 Santoro, Nicola 1181
 Sanwalani, Vishal 200
 Sassone, Vladimiro 969
 Schaefer, Gilles 1080

Scheideler, Christian 1153
 Schnitger, Georg 66, 439
 Schnoebelen, Phillipe 790
 Schönhage, Arnold 611
 Sedgwick, Eric 845
 Segal, Michael 1169
 Sen, Pranab 300
 Sen, Sandeep 384
 Sénizergues, Géraud 478
 Shachnai, Hadas 1123
 Shepherd, F. Bruce 410
 Sitters, René A. 624
 Skutella, Martin 397
 Srinivasan, Aravind 164
 Stee, Rob van 653
 Stoelinga, Mariëlle 464
 Stougie, Leen 624

 Tamir, Tami 1123
 Thérien, Denis 984
 Thilikos, Dimitrios M. 829

Torre, Salvatore La 776

 Unger, Walter 1108

 Vaandrager, Frits 464
 Vaccaro, Ugo 81
 Vardi, Moshe Y. 64, 697
 Voronkov, Andrei 714

 Wolf, Ronald de 291

 Xia, Ge 845
 Xie, Gaoyan 668

 Yamashita, Masafumi 1054
 Ye, Yinyu 145
 Yung, Moti 97

 Zappa Nardelli, Francesco 584
 Zavattaro, Gianluigi 133
 Zhang, Jiawei 145
 Zucca, Elena 224