Lecture Notes in Computer Science 2234

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

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Leszek Pacholski Peter Ružička (Eds.)

SOFSEM 2001: Theory and Practice of Informatics

28th Conference on Current Trends
in Theory and Practice of Informatics
Piešťany, Slovak Republic, November 24 – December 1, 2001
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Leszek Pacholski
University of Wroclaw, Institute of Computer Science
Przesmyckiego 20, 51-151 Wroclaw, Poland
E-mail: pacholsk@tcs.uni.wroc.pl

Peter Ružička
Comenius Univ., Fac. of Mathematics, Physics, and Informatics, Inst. of Informatics
Mlynská dolina, 84248 Bratislava, Slovakia
E-mail: ruzicka@dcs.fmph.uniba.sk

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Theory and practice of informatics : proceedings / SOFSEM 2001, 28th Conference on Current Trends in Theory and Practice of Informatics, Piešťany, Slovak Republic, November 24 - December 1, 2001. Leszek Pacholski ; Peter Ružička (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2234)
ISBN 3-540-42912-3

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

ISSN 0302-9743
ISBN 3-540-42912-3 Springer-Verlag Berlin Heidelberg New York

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

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN: 10845795 06/3142 5 4 3 2 1 0

Foreword

SOFSEM 2001, the International Conference on Current Trends in Theory and Practice of Informatics, was held on November 24 – December 1, 2001 in the well-known spa Piešťany, Slovak Republic. This was the 28th annual conference in the SOFSEM series organized either in the Slovak or the Czech Republic.

SOFSEM has a well-established tradition. Currently it is a broad, multidisciplinary conference, devoted to the theory and practice of software systems. Its aim is to foster cooperation among professionals from academia and industry working in various areas of informatics.

The scientific program of SOFSEM consists of invited talks, which determine the topics of the conference, and short contributed talks presenting original results. The topics of the invited talks are chosen so as to cover the whole range from theory to practice and to bring interesting research areas to the attention of conference participants. For the year 2001, the following three directions were chosen for presentation by the SOFSEM Steering Committee:

- Trends in Informatics
- Enabling Technologies for Global Computing
- Practical Systems Engineering and Applications

The above directions were covered through 12 invited talks presented by prominent researchers. There were 18 contributed talks, selected by the international Program Committee from among 46 submitted papers. The conference was also accompanied by workshops on Electronic Commerce Systems (coordinated by H. D. Zimmermann) and Soft Computing (coordinated by P. Hájek).

The present volume contains invited papers (including the keynote talk given by Vaughan R. Pratt), the Soft Computing workshop opening plenary talk (presented by Sándor Jenei), and all the contributed papers.

We are grateful to the members of both the SOFSEM Advisory Board and the SOFSEM Steering Committee for their proposals for the conference scientific program and for their cooperation in contacting the invited speakers. We also wish to thank everybody who submitted a paper for consideration, all Program Committee members for their meritorious work in evaluating the submitted papers, as well as all subreferees who assisted the Program Committee members in the evaluation process. We are deeply indebted to the authors of invited and contributed papers who prepared their manuscripts for presentation in this volume. Our special thanks goes to Miroslav Chladný who designed and managed electronic support for the Program Committee and who did most of the hard technical work in preparing this volume. We are also thankful to the Organizing Committee team lead by Igor Prívara, who made sure that the conference ran smoothly in a pleasant environment. Last but not least we would like to thank Springer-Verlag for their excellent cooperation during the publication of this volume.

September 2001 Leszek Pacholski, Peter Ružička

SOFSEM 2001

Organized by

Slovak Society for Computer Science
Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava

in cooperation with

Czech Society for Computer Science
Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Bratislava
Institute of Informatics and Statistics, Bratislava
Mathematical Institute, Slovak Academy of Sciences, Bratislava
SOFTEC, Ltd., Bratislava

Supported by

European Association for Theoretical Computer Science
Slovak Research Consortium for Informatics and Mathematics

Sponsored by

Compaq Computer Slovakia
European Research Consortium for Informatics and Mathematics
Oracle Slovensko
SAP Slovensko

Telenor Slovakia provided Internet connection to the conference site and hosted the SOFSEM 2001 web page.

Advisory Board

D. Bjørner (Technical University of Denmark, Lyngby, Denmark), M. Broy (Technical University Munich, Germany), M. Chytil (ANIMA Prague, Czech Republic), P. van Emde Boas (University of Amsterdam, The Netherlands), G. Gottlob (Vienna University of Technology, Austria), K. G. Jeffrey (CLRC RAL, Chilton, Didcot, Oxon, UK), M. Zemánková (NSF, Washington D.C., USA).

Steering Committee

M. Bartošek (Masaryk University, Brno, Czech Republic) (*secretary*), K. G. Jeffery (CLRC RAL, Chilton, Didcot, Oxon, UK), F. Plášil (Charles University, Prague, Czech Republic), I. Prívara (INFOSTAT, Bratislava, Slovak Republic), B. Rovan (Comenius University, Bratislava, Slovak Republic) (*chair*), J. Staudek (Masaryk University, Brno, Czech Republic), G. Tel (Utrecht University, The Netherlands), J. Wiedermann (Academy of Sciences, Prague, Czech Republic).

Program Committee

M. Bieliková (Bratislava), H. Bischof (Vienna), J. Csirik (Szeged), G. Engels (Paderborn), P. van Emde Boas (Amsterdam), V. Hlaváč (Prague), L. Kollar (Redmond), M. Křetínský (Brno), K. Matiaško (Žilina), L. Pacholski (Wrocław, *co-chair*), F. Plášil (Prague), J. Pokorný (Prague), P. Ružička (Bratislava, *chair*), J. Šafařík (Bratislava), A. Steger (Munich), G. Tel (Utrecht), P. Tůma (Prague), U. Ultes-Nitsche (Southampton), S. Zaks (Haifa).

Organizing Committee

M. Bečka, M. Chladný, I. Dostálová, R. Graus, V. Hambálková, R. Královič, Z. Kubincová, M. Nehéz, D. Pardubská (*vice-chair*), I. Prívara (*chair*), E. Ričányová, V. Solčány.

Table of Contents

Invited Talks

Contributed Papers

Keynote Talk

The Potential of Grid, Virtual Laboratories and Virtual Organizations for Bio-sciences

Hamideh Afsarmanesh, Ersin Kaletas, and Louis O. Hertzberger

University of Amsterdam, Informatics Institute,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{hamideh, kaletas, bob}@science.uva.nl

Abstract. VLAM-G, the Grid-based Virtual Laboratory AMsterdam, provides a science portal for distributed analysis in applied scientific research. It offers scientists the possibility to carry out their experiments in a familiar environment that provides seamless access to geographically distributed resources and devices. In this paper, the general design of the VLAM-G platform is introduced. Furthermore, the application of the VLAM-G and its extension with Virtual Organization concepts for specific scientific domains is presented, with focus on bio-sciences.

1 The Grid-Based Virtual Laboratory in Amsterdam

Due to the systematic growth of research efforts in experimental sciences, vast amounts of data/information are gathered at scattered data resources all around the world, that also need to be accessed by many geographically-distributed end-users. This scenario clearly requires shared access to resources available through global distributed computer facilities. In addition, advanced functionalities are required in order to allow researchers to conduct high-level scientific experimentation on top of such a distributed computer system. These advanced functionalities include for instance: distributed information management, data and information disclosure, visualization, etc. Modern advances in the IT area such as the Grid [1], [2] and new approaches such as Virtual Laboratories (VL) [3] can be properly applied here.

In this context, the ICES/KIS-II project VLAM-G [4] of the University of Amsterdam (UvA) aims at the design and development of an open, flexible, scalable, and configurable framework providing necessary Grid-based hardware and software enabling scientists and engineers in different areas of research to work on their problems via experimentation, while making optimum use of the modern Information Technology. The VLAM-G provides a distributed high-performance computing and communication infrastructure with advanced information management functionalities, addressing in specific the experimentation requirements in, among others, the scientific domains of biology, physics, and systems engineering. As such, access to physically distributed data and processes among many sites in the virtual laboratory, necessary for the achievement of complex experimentations, is totally transparent to the scientist, giving them the image of working in a single physical laboratory.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 1–9, 2001.

Considering these ideas, the global reference scenario for the VLAM-G is depicted in Fig. 1. Namely, the Virtual Laboratory is regarded as a Grid-based software layer on top which applications coming from different scientific domains can be developed. For instance, this figure depicts the following specific applications:

- "Materials Analysis of Complex Surfaces" (MACS) from the chemistry and physics domain. Here, the use of MACS methods within VLAM-G to study the properties of material surfaces has proven to be a powerful tool, not only for fundamental but also for applied research in fields as diverse as art conservation, cancer therapy and mass spectrometry [5].
- The DNA-Array application from the bio-informatics domain. In this case, the VLAM-G supports the integration of Micro-array technology to enable the study of the characteristics of thousands of genes in a single experiment [6].
- The Radiology application, for which advanced visualization and interactive simulation techniques are required. Namely, within the VLAM-G environment, a Virtual Radiology Explorer is demonstrated based on the virtual reality environment and database facilities that are integrated by the Virtual Laboratory layer [7].

As mentioned previously, all these applications are supported by the Virtual Laboratory layer based on the Grid infrastructure. Namely, considering the performance requirements of these application scenarios in terms of the amount of data being exchanged and the heavy computational processes that need to be executed, the use of a high-performance distributed resource management architecture, such as the Data

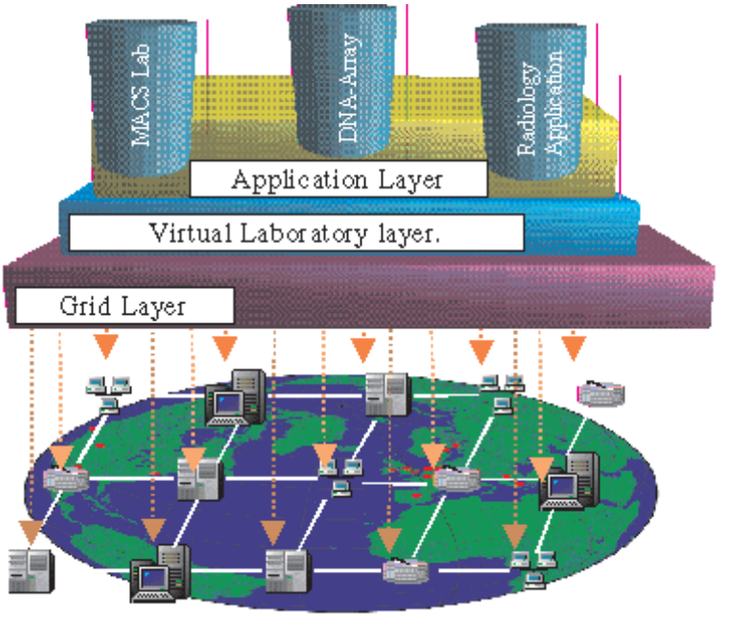


Fig. 1. The VLAM-G global reference scenario

Grid, is mandatory within the Virtual Laboratory infrastructure. The main objectives of the Data Grid are to integrate huge heterogeneous data archives into a distributed data management *Grid*, and to identify services for high-performance, distributed, data intensive computing. As such, this infrastructure offers a wide variety of high-performance services that are well exploited by the VLAM-G infrastructure components.

Furthermore, an architectural overview of the VLAM-G layer itself is given in Fig. 2. Thus, the VLAM-G represents a modular architecture composed of the following tiers. The application tier, in which the aforementioned scientific applications are provided. The application toolkits tier, including the web science portals and internal core components, such as: interactive visualization and simulation (VISE), communication and collaboration (COMCOL), and VL information management for cooperation (VIMCO) components. The Grid middleware tier, providing the Grid services and the Grid resources tier for the access to the underlying physical/logical distributed resources.

Within the application toolkit tier, the Web-based portal and workbench interface, together with the modular design of the VLAM-G architecture, provide a uniform environment for all experiments, and makes it possible to attach a wide range of software tools to the Virtual Laboratory. This ranges from basic tools such as simulation, visualization, data storage / manipulation to advanced facilities like: remote controlling of devices, visualization in a virtual reality environment and federated advanced information management. Modularity is the key to scalability and openness, and also to support the inter-disciplinary research. In this way, VLAM-G solves many technical problems that scientists face, hence enabling them to focus better on their experiments, and simultaneously it reduces the costs of experimentation by sharing the expensive resources among them.

The VIMCO component being developed by the CO-IM (Co-Operative Information Management) group of the UvA, supports advanced information management requirements of the VLAM-G applications, addressing many of the obstacles described earlier.

2 Data and Information Management in Biology

New generation of automated high-throughput experimental technologies has paved the way to sequence millions of base pairs a day, run thousands of assays for testing chemical compounds, and monitor the expression profiles of thousands of genes in a single experiment; leading to exponential growth in size of genome databases. At the present rate of growth, the SWISS-PROT database for instance, will double in size every 40 months and the nucleotide databases will double in size about every 14 months [8]. During the year 2000, every day over two million bases were deposited into GenBank, and the bio-informatics companies have already generated terabytes of genomic data [9].

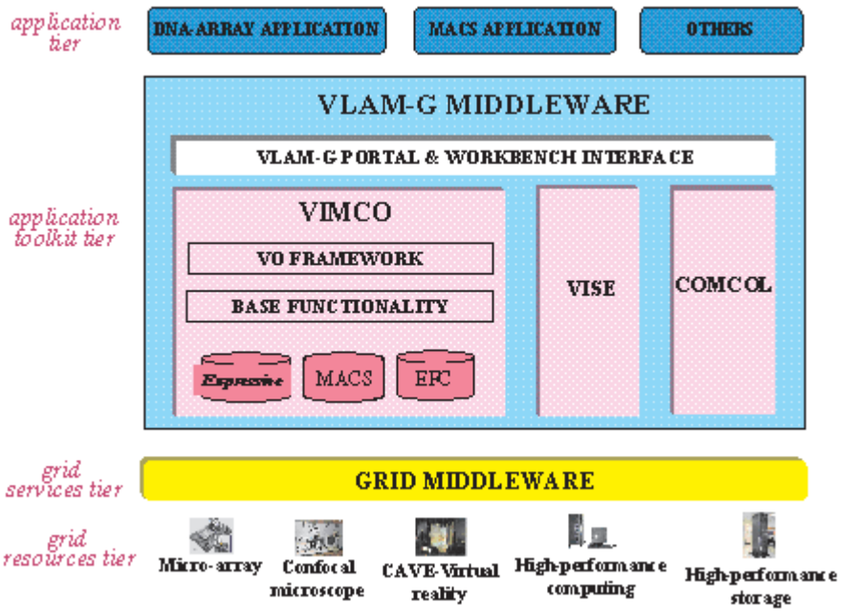


Fig. 2. VLAM-G architecture overview

At the same time, considering different areas within biosciences, research has resulted a large number and range of "heterogeneous" genome databases. The heterogeneity is partially due to: (1) the wide variety of types of genomic information, (2) various representations/formats designed for the stored data, and (3) different data storage organization used to store this information. In the latest compilation of key high-quality biological database resources of value, available all around the world [10], 281 databases are listed in 18 different categories of biological content. Every database is provided and supported by an independent and "autonomous" center. Out of these databases, 55 were added only since the previous compilation, reported in January 2000. These data resources contain information ranging from sequence to pathology, and gene expression to pathway aspects of the biosciences. Clearly, diverse information is stored using different data storage organizations by different centers. Many data resources store data in flat files, while each defining its own specific data structure and format to represent the data. Others use database management systems (DBMS) to store data, but even if the same DBMS is used by two centers, there are enormous differences in data representation (data models/data formats) designed by each center. Furthermore, querying capabilities and interfaces to retrieve data change from one resource to another. The heterogeneity problem grows further with the relatively new types of information, such as for instance the gene expression data, where even the consensus on defining the required minimum information set to be stored is still under study [11].

Data provided so far through the database resources reflect that: human genome is mostly sequenced, several metabolic pathways are identified, expression profiles have been generated for many genes, etc., that in turn opens the gates to the possibility of new discoveries and advanced researches, through the "extraction of knowledge" from the scattered collected data. Achieving advanced extraction of knowledge, performed either by individual scientists or in collaboration with other individuals or centers, primarily constitutes a "large experimentation process" with many steps. Namely, several distributed, heterogeneous, and autonomous sites need to be interlinked, and perhaps co-work towards the achievement of a common goal. Such an involved process both requires and benefits from some assisting "enhancement environments", that can facilitate the variety of needed functionalities and can provide models, mechanisms, and tools enabling scientists and centers with this complex and relatively long experimentation process.

From the information management point of view, such experimentation process involves many steps including for example: access to distributed data resources from different public and private (autonomous) centers, finding the relevant pieces of data, bringing these pieces of data together, understanding what each heterogeneous piece means and how it can relate to the other pieces, inter-link the relevant data, then after any processing/analyses store the results together with some description of the processes involved in the performed experiment for the sake of future reference to this experiment, and furthermore sharing some results with certain other collaborators. As a first base to support advanced experimentation, a comprehensive integrated/unified meta-data model (schema) must be created representing the variety of accessed information. This is a challenging task, especially when the experiment either covers several distinct areas within the biosciences (e.g. gene sequences, expressions, and pathways), or even more so when it is a multi-disciplinary scientific experiment (e.g. the proteomics research involving the fields of biology as well as physics and chemistry). To fully support advanced experiments, many information management problems need to be addressed, including for example handling: storage of very large scientific data sets in databases [12], incomplete and inconsistent data[13], heterogeneity in the used terminology in meta-data[14], semantic heterogeneity in the defined data models, syntactic heterogeneity in the data representation/formatting, preservation of site autonomy, and the interoperation and security issues among collaborating centers. Solutions addressing some of these challenges have been proposed [15, 16], however many important issues such as for instance, the secure sharing of some proprietary information with only certain authorized users while protecting it against others, or the design of flexible / dynamic mechanisms for autonomous sites to enable them to define information visibility levels and access rights for other sites, at fine granularity [17] are not yet properly addressed.

Motivated by these emerging requirements to support advanced experiments in Biosciences", enormous challenges are created for research in the area of "federated information management" [18], as well as for the creation of experimentation enhancement infrastructures supporting: (1) the research scientists with the systematic definition and execution of their experimentations, through the "Virtual Laboratory" environment [4] [19], and (2) the collaboration / interoperation of independent and

autonomous sites requiring authorized and secure exchange of information and coordination of their distributed but joint activities, through the "Virtual Organization" [20] environment.

A main target scientific domain studied by VIMCO, was the DNA-array experimentations from the bioscience domain. In this study, a large set of aspects/entities common to DNA-array scientific experiments were identified, and EEDM (Experimentation Environment Data Model), a base meta-data model for representation of experimental data [19], was designed. Later on the EEDM was further extended to become generalized representing the experimental data from other domains considered in the VLAM-G, namely the Material Analyses of Complex Surfaces (MACS) from the Physics domain and the Electronic Fee Collection (EFC) from the Engineering domain. The importance of such a base harmonized/integrated meta-data model, as described earlier in the paper, is not only for better support to complex experimentations in the area of biosciences, but the fact that it is invaluable in supporting interdisciplinary research. For instance, it supports the representation of inter-related biological, chemical, and physical properties of a certain biological element at once

Based on the EEDM data model defined for data modeling of scientific experimentations, several databases are developed in VLAM-G. In specific, EXPRESSIVE database is developed for gene expressions, in the context of the DNA-array application [21]. EXPESSIVE aims at both (1) definition and management of information to support the storage and retrieval of the *steps and annotations* involved in DNA micro-array experiments, for the purpose of investigation and reproducibility of experiments through the VLAM-G; and (2) storage and retrieval of the DNA *micro-array experiment results* (raw data and processed analysis results) through the VLAM-G, for the purpose of investigation, sharing, and scientific collaboration with other scientific centersThe VLAM-G EXPRESSIVE database model [21] complies with the MIAME specifications [11]; it is powerful enough to store any experimental information/annotations in the database, and thanks to the dynamic and flexible nature of the EEDM model, EXPRESSIVE is open for necessary future extensions.

In VIMCO, in addition to the base common data manipulation mechanisms, several libraries supporting Web based and platform independent database access are provided. Furthermore, distributed and multi-threaded manipulation of data for multi-user access to Virtual Laboratory environment, and XML-based export, import and data/information exchange facilities for interoperation among federated databases are developed, that are used by all VLAM-G databases.

3 A Biology Virtual Organization

Most emerging applications of the future require an enhancement environment supporting proper collaboration and interoperation among different autonomous and heterogeneous organizations. While advances in ICT and networking provide the base technologies, innovative approaches in several areas including: safe communication, process coordination and workflow management, and federated information management are necessary to provide the support infrastructure for the Virtual Organiza-

tions (VO) [20] paradigm. For the Bio-science applications, a Biology Virtual Organization is defined as a temporary (or permanent) alliance of enterprises and centers in the biology sector, that come together to share skills, core competencies and resources in order to achieve common goals, and whose cooperation is supported by computer networks. The concept of VO complements and extends the functionality and potential of VLAM-G in the sense that VO reinforces the secure collaboration and coordinated interoperation of several individual autonomous organizations as a single entity, towards the achievement of VO's common goals. Therefore, VO carefully addresses the secure exchange of proprietary information, through the communication network, among collaborating organizations, and tackles the coordination of distributed processes performed at different organizations. As such, several different VOs among different biology research groups may co-exist, while each has its own distinct goals, set of partners, and cooperation rules ensuring partners security. Furthermore, one organization may simultaneously be a partner in several VOs, while involved in different tasks and following different collaboration and information exchange rules in every VO.

Fig. 3 illustrates the concept of Biology Virtual Organization (BVO) with an example, and shows how it is used together with VLAM-G. The BVO depicted in this figure is composed of three partners collaborating in order to develop a new pharmaceutical drug: a pharmaceutical industry (Organization A), a biotechnology company which has an advanced DNA micro-array facility (Organization B) and a university based group providing a high-performance computing infrastructure (Organization C). All organizations have VLAM-G middleware installed. Organization A is in need of an advanced micro-array facility to produce the required arrays, and a high-performance computing facility to analyze the results, which are not locally available at Organization A. If the three organizations agree, then a BVO will be established, where organizations B and C will share their facilities with organization A, based on well-defined contracts.

Among the main considerations for development of BVO support infrastructure as extensions to the VLAM-G, we can mention the following: (1) incorporation of information modeling standards, (2) utilization of the existing VLAM-G with facilities such as reliability, robustness, security, and scalability, and (3) support for sharing and exchange of distributed information, maintaining the proper level of autonomy and security for each BVO partner, and provision of mechanisms for dynamic definition of information visibility levels and access rights for other BVO partners.

4 Conclusion

In conclusion, new discoveries and emerging advanced research in Biosciences constitutes complex and relatively long experimentation processes that requires advanced federated information management and can benefit from the Virtual Laboratory and Virtual Organization environments. The Grid-based Virtual Laboratory VLAM-G and virtual organization BVO described in this paper provide the base infrastructure,

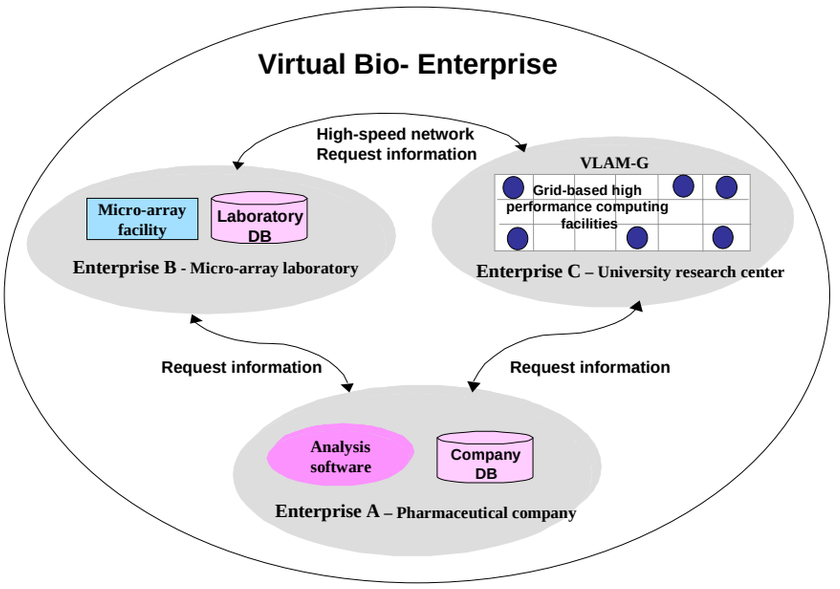


Fig. 3. Biology Virtual Organization (BVO)

where the federated information management of VIMCO with its EEDM meta-data model and the EXPRESSIVE database can be helpful to solve the obstacles a scientist faces when realizing his experiment, and assist him towards proper extraction of knowledge from the vast amounts of heterogeneous data from autonomous sources.

Acknowledgements

This work was supported by an ICES/KIS grant from the Dutch Ministry of Education and Sciences and the Ministry of Economic Affairs. The authors also wish to thank all members of the Virtual Laboratory team, and in particular Dr. A. Belloum and Mr. C. Garita for their valuable comments and contributions to the work presented in this paper.

References

1. I. Foster, C. Kesselman, and S. Tuecke, “The Anatomy of the Grid (to appear)”, International Journal of Supercomputer Applications (2001).
2. I. Foster and C. Kesselman, “A toolkit-based grid architecture”, in The Grid: Blueprints for a Future Computing Infrastructure: Morgan-Kauffman (1998).
3. DOE, "DOE2000 Collaboratories", http://www-unix.mcs.anl.gov/DOE2000/collabs.html, (2000).

4. H. Afsarmanesh, E. Kaletas, A. Benabdelkader, C. Garita, and L. O. Hertzberger, “A Reference Architecture for Scientific Virtual Laboratories”, Future Generation Computer Systems, vol. 17, (2001) 999-1008.
5. G. B. Eijkel, H. Afsarmanesh, D. Groep, A. Frenkel, and R. M. A. Heeren, “Mass spectrometry in the Amsterdam Virtual Laboratory: Development of a high-performance platform for meta-data analysis”, presented at 13th Sanibel Conference on Mass Spectometry: Informatics and Mass Spectometry, Florida, USA (2001).
6. P. Sterk, “Accessibility of micro-array data through the Virtual Laboratory at the University of Amsterdam”, presented at 3rd International Meeting on Microarray Data Standards, Annotations, Ontologies and Databases, Stanford CA, USA (2001).
7. Z.Zhao, R. G. Belleman, G. D. v. Albada, and P. M. A. Sloot, “System Integration for Interactive Simulation Systems using Intelligent Agents”, presented at ASCI Conference, Heijen, The Netherlands (2001).
8. G. Williams, “Nucleic acid and protein sequence databases”, in Genetics Databases, M. J. Bishop, Ed. Hinxton, Cambridge: Academic Press (1999) 11-37.
9. CELERA, "Celera Genomics", http://www.celera.com/,(2001).
10. A. D. Baxevanis, “The Molecular Biology Database Collection: an updated compilation of biological database resources”, Nucleic Acids Research, vol. 29, (2001).
11. "MIAME - Minimum Information about a Microarray Experiment", http://www.ebi.ac.uk/microarray/MGED/Annotations-wg/index.html, (2001).
12. M. J. Bishop, “Genetics Databases”. Hinxton, Cambridge: Academic Press (1999).
13. E. Pennisi, “Keeping genome databases clean and up to date”, Science, vol. 286, (1999) 447-449.
14. E. Pennisi, “Seeking common language in a tower of Babel”, Science, vol. 286, (1999) 449-450.
15. A. Siepel, A. Farmer, A. Tolopko, M. Zhuang, P. Mendes, W. Beavis, and B. Sobral, “ISYS:A decentralized, component-based approach to the integration of heterogeneous bioinformatics resources”, Bioinformatics, (2000).
16. L. M. Haas, P. M. Schwarz, P. Kodali, E. Kotlar, J. E. Rice, and W. C. Swope, “Discovery Link: A system for integrated access to life sciences data sources”, IBM Systems Journal, vol. 40, (2001) 489-511.
17. C. Garita, H. Afsarmanesh, and L. O. Hertzberger, “A Survey of Distributed Information Management Approaches for Virtual Enterprise Infrastructures (to appear)”, in Managing Virtual Web Organizations in the 21st Century: Issues and Challenges, U. J. Franke, Ed.: Idea Group Publishing, (2001).
18. C. Garita, Y. Ugur, A. Frenkel, H. Afsarmanesh, and L. O. Hertzberger, “DIMS: Implementation of a Federated Information Management System for PRODNET II”, presented at 11th International Conference and Workshop on Database and Expert Systems Applications - DEXA '2000, London, England (2000).
19. E. Kaletas, C. Garita, H. Afsarmanesh, and L. O. Hertzberger, “Implementation of a Federated Information Management System to Support VEs in the Tourism Sector”, Faculty of Science - University of Amsterdam, Technical Report, CS-01-05 (2001).
20. L. M. Camarinha-Matos and H. Afsarmanesh, “The Virtual Enterprise Concept”, in Infrastructures for Virtual Enterprises - Networking Industrial Enterprises, L. M. Camarinha-Matos and H. Afsarmanesh, Eds.: Kluwer Academic, (1999) 3-14.
21. E. C. Kaletas, H. Afsarmanesh, T. M. Breit, and L. O. Hertzberger, “EXPRESSIVE - A database for micro-array gene expression studies”, University of Amsterdam, Informatics Institute, Amsterdam, Technical report, CS-2001-02 (2001).

Agreement Problems in Fault-Tolerant Distributed Systems

Bernadette Charron-Bost

LIX, École Polytechnique, 91128 Palaiseau Cedex, France
charron@lix.polytechnique.fr

Abstract. Reaching agreement in a distributed system is a fundamental issue of both theoretical and practical importance. Consensus, Atomic Commitment, Atomic Broadcast, Group Membership which are different versions of this paradigm underly much of existing fault-tolerant distributed systems. We describe these problems, explain their relationships, and state some fundamental results on their solvability, depending on the system model. We then review and compare basic techniques to circumvent impossibility results in asynchronous systems: randomization, models of partial synchrony, unreliable failure detection.

1 Introduction

The design and verification of fault-tolerant distributed applications is notoriously quite challenging. In the last two decades, several paradigms have been identified for helping in this task. Key among these are consensus, atomic commitment, atomic broadcast, and group membership. In all these problems, processes ought to reach some form of agreement for achieving global consistency in the system. Roughly speaking, *consensus* and *atomic commitment* allow processes to reach a common decision, which depends on their initial values and/or on failures. *Atomic broadcast* is a convenient and efficient communication tool for delivering messages consistently, i.e., in the same order. A *group membership* manages the formation and the maintenance of a set of processes, called a group; each process has a local view of the group, and some form of process agreement is required to guarantee that local views are consistent. Since global consistency is what makes a collection of processes into a single system, distributed agreement algorithms are ubiquitous in distributed systems.

From a theoretical point of view, agreement problems are intensively studied because they have simple rigorously formulations and are surprisingly challenging. Impossibility results and lower bounds have been proved, demonstrating limitations on what problems can be solved, and with what costs. These results are fundamental in practice: they precisely establish the limits of what can be built, depending on the types of systems. Interestingly, they also provide a convenient way for comparing the power of models that make different assumptions about time and failures.

In this survey, our primary goal is to give precise specifications of the agreement problems mentioned above (namely, consensus, atomic commitment, atomic

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 10–32, 2001.

broadcast, and group membership), and to go over the most basic results known for each of them. Remarkably, all these agreement problems are showed to be unsolvable in fault-prone asynchronous systems. However, they are so fundamental that it is crucial to find ways around this limitation. We then review and compare the three classical approaches to circumvent these impossibility results, namely *partial synchrony*, *randomization*, and *failure detection*.

Given the vastness of the field, and the space limitations, we cannot attempt an exhaustive presentation of the material: we do not address some agreement problems (for instance, approximate agreement, k-agreement); we focus on the failures which are the most common ones in practice. Furthermore, many significant results are just mentioned in passing or not at all, and most proofs are omitted. Similarly, the bibliographic references are extensive, but incomplete. A comprehensive treatment can be found in the remarkable book by Lynch [37].

We rather strive to give a "global picture" of the field, and to put the main results in perspective. In particular, we point out that minor differences in assumptions about the model or in the problem specifications can result in quite important differences in the results on solvability and efficiency. This gives evidence that a rigorous treatment is needed in the area of fault-tolerant distributed computing to cope with the many subtle phenomena that arise.

This paper is organized as follows. In Section 2, we describe the models of computation commonly used. We successively examine the consensus, atomic commitment, atomic broadcast, and group membership problems in Section 3. Various ways of circumventing the impossibility results stated in this section are discussed in Section 4. In particular, some new results concerning atomic commitment are presented.

2 Models of Distributed Computing

Problems in fault-tolerant distributed computing have been studied in a large variety of computational models. Such models are classified in two categories according to the communication medium: *message-passing* and *shared-memory*. In the former, processes communicate by exchanging messages over communication channels; in the latter, they interact with each other via shared objects, such as registers, queues, etc. In this paper, we focus on message-passing models. In addition to the communication medium, the main features of a model are its degree of synchrony and the types of failures that are assumed to occur.

2.1 Degree of Synchrony

Synchrony is an attribute of both processes and communication channels. A system is said to be *synchronous* if it satisfies the following two properties:

1. There is a known upper bound on message delay, that is on the time it takes for a message to be delivered.
2. There is a known upper bound on the time that elapses between consecutive steps of a process.

A system is *asynchronous* if there is no such bounds. In other words, there is no timing assumptions in asynchronous systems.

Distributed applications are hard to design in the asynchronous model since no timing information can be used by algorithms. However, this model is attractive and has been extensively studied for several reasons: it has simple semantics, asynchronous algorithms are easier to port than those referring to some specific timing assumptions and are guaranteed to run correctly with arbitrary timing guarantees.

2.2 Failure Model

In an execution, a component (i.e., a process or a link) is *faulty* if its behavior deviates from the one prescribed by the algorithm it is running; otherwise, it is *correct*. A *failure model* specifies in what way a faulty component can deviate from its code.

Process Failure. One mainly considers two types of process failures:

1. *Crash:* a faulty process stops prematurely in the middle of its execution. Before stopping, it behaves correctly.
2. *Byzantine:*[1] a faulty process can exhibit any behavior whatsoever. For example, it can generate message or change state arbitrarily, without following its code.

In both of these failure models, one usually need to assume limitations on the number of process failures. In some works on analysis of systems with process failures, these limitations often take the form of probability distributions governing the frequency of failures. Instead of using probability, the works on distributed algorithms simply assume that the number of failures is bounded in advance by a fixed number t.[2]

Link Failure. It is commonly supposed that link failure can result only in lost messages. Models in which incorrect messages may be delivered are seldom studied because in practice, the use of checksums allows the system to detect garbled messages and discard them. A link failure may be transient and so yield the destruction of some messages, or it may cause all messages sent over the link to be lost. This latter type of link failure may lead to network partition, that is

[1] The term *Byzantine* was first used for this type of failure in a landmark paper by Lamport, Pease, and Shostak [36], in which the consensus problem is formulated in terms of *Byzantine generals*.

[2] At first sight, such an assumption is realistic in practice, in the sense that it is "unlikely" that more than t failures occur. However, in most practical situations, if the number of failures is already large, it is likely that more failures will occur. Assuming a bound on the number of failures implies that failures are *negatively correlated*, whereas in practice, failures are independent or positively correlated.

the communication graph becomes disconnected. Then, some pairs of processes cannot communicate, making most problems unsolvable.

In order to cope with link failures, there are basically two classical approaches. The first one consists in placing the responsibility of a message loss on the sender or the receiver. Link failures are so translated into process failures. This gives rise to a new type of process failures called *omission failures*, which are intermediate between crash and byzantine failures. Unfortunately, it turns out that this approach has bad side effects: it puts some artificial limits on the number and the localization of lossy links (instead of just limiting the number of message losses) and results in an undesirable weakening of the problem requirements if the specification refers only to the behavior of correct processes (as for the non uniform agreement problems introduced in Section 3).

The second approach uses *data link protocols* to mask message losses, i.e., to simulate reliable links. The problem of implementing reliable communications using unreliable links has been extensively studied. For example, the reader can refer to [7,48,1,8].

In the following, we consider process failures, but assume that links are reliable.

2.3 Formal Model

Now, we briefly present formal models for asynchronous and synchronous systems; notation and definitions are borrowed from [27,21] and [37].

We have tried to write this paper so that it can be mostly understood with minimal formal prerequisite. The reader not interested in the impossibility proofs of consensus and atomic commitment (presented in Section 3), can skip the following formal definitions, save the one of round in the synchronous model.

We consider distributed systems consisting of a set of n processes $\Pi = \{p_1, \cdots, p_n\}$. Every pair of processes is connected by a reliable channel.

Asynchronous Systems. Each process p_i has a buffer, $buffer_i$, that represents the set of messages that have been sent to p_i but that are not yet received. An *algorithm* A is a collection of n deterministic automata, one for each process. The automaton which runs on p_i is denoted by A_i. A *configuration* C of A consists of:

- n process states $state_1(C), \cdots, state_n(C)$ of $A_1, \cdots, A_n$, respectively;
- n sets of messages $buffer_1(C), \cdots, buffer_n(C)$, representing the messages presently in $buffer_1, \cdots, buffer_n$.

Configuration C is an *initial configuration* if every state $state_i(C)$ is an initial state of A_i and $buffer_i(C)$ is empty. Computations proceed in *steps* of A. In each step, a unique process p_i atomically (1) receives a single message from the message buffer $buffer_i$ or a "null" message meaning that no message is received by during this step, (2) changes its state, and (3) may send messages to other processes, depending on its state at the beginning of the step and on the message

received in the step. The message received during the receive phase of a step of p_i is chosen nondeterministically among the messages in $buffer_i$, and the null message. In particular, the null message may be received even if though $buffer_i$ is not empty. A step executed by process p_i is *applicable to* a configuration C if the message received in this step is present in $buffer_i(C)$.

A *schedule* of A is a finite or infinite sequence of A's steps. A schedule S is *applicable to* a configuration C if the steps of S are applicable in turn, starting from C. If S is finite, $S(C)$ denotes the resulting configuration, which is said to be *reachable* from C.

In the asynchronous model, we will only consider crash failures (cf. Section 3). A process p_i is *correct* in an infinite schedule S provided it takes infinitely many steps in S, and it is *faulty* otherwise. A *run of algorithm* A in the *asynchronous model* is a pair $<C_I, S>$, where C_I is an initial configuration of A, S is an infinite schedule of A applicable to C_I, and every message sent to a correct process is eventually received in S.

Synchronous Systems. In the case of synchronous systems, computations can be organized in *synchronized rounds*. This gives rise to a simple computational model that is quite convenient to describe, prove, and assess distributed algorithms in the synchronous case.

As for asynchronous systems, each process p_i has a buffer denoted $buffer_i$. An algorithm A of the synchronous model consists for each process $p_i \in \Pi$ in the following components: a set of states denoted by $states_i$, a nonempty subset $init_i$ of $states_i$ representing the possible initial states of p_i , a message-generation function $msgs_i$ mapping each pair in $states_i \times \Pi$ to a unique (possibly $null$) message, and a state-transition function $trans_i$ mapping $state_i$ and vectors (indexed by Π) of message to $states_i$. In any execution of A, each process p_i repeatedly performs the following two stages:

1. Apply $msgs_i$ to the current state to generate the messages to be sent to each process. Put these messages in the appropriate buffers.
2. Apply $trans_i$ to the current state and the messages present in $buffer_i$ to obtain the new state. Remove all messages from the $buffer_i$.

The combination of these two actions is called a *(synchronized) round* of A.[3]

A process can exhibit a crash failure by stopping anywhere in the middle of its execution. In terms of the model, the process may fail before, after, or in the middle of performing Stage 1 or Stage 2. This means that the process may succeed in sending only a subset of the messages it is supposed to send, thus creating inconsistency in the system. A process can also exhibit byzantine failure by generating its next messages or next state in an arbitrary way, without following its code, i.e., the rules specified by its message-generation and state-transition functions.

[3] A classical way for emulating synchronized rounds in a synchronous system uses a simple time-out mechanism. Time-out periods are then determined by the timing bounds available in the synchronous system.

A *schedule* of an algorithm is defined as a finite or infinite sequence of successive rounds. As for the asynchronous model, we define the notions configuration, initial configuration, and run. In this model, the time complexity of an algorithm is measured in terms of the number of rounds until all the required outputs are produced.

3 Agreement Problems in Distributed Systems

3.1 Consensus

The *consensus* problem is a simplified version of a problem that originally arose in the development of on-board aircraft systems. In this problem, some components in a redundant system ought to settle on a value, given slightly different readings from different sensors (eg., altimeters). Consensus algorithms are thus incorporated into the hardware of fault-tolerant systems to guarantee that a collection of processors carry out identical computations, agreeing on the results of some critical steps. This redundancy allows the processors to tolerate the failure of some processors.

In the consensus problem, each process starts with an initial value from a fixed set V, and must eventually reach a common and irrevocable decision from V. More formally, the consensus problem is specified as follows:

Agreement: No two correct processes decide differently.
Validity: If all processes processes start with the same value v, then v is the only possible decision value.
Termination: All correct processes eventually decide.

An alternative validity condition is as follows:

Strong validity: If a process decides v, then v is the initial value of some process.

Obviously, this condition implies the validity condition that we have stated first, and these two conditions are equivalent for the binary consensus, that is when $V = \{0, 1\}$.

The agreement condition of consensus may sound odd because it allows two processes to disagree even if one of them fails a very long time after deciding. Clearly, such disagreements are undesirable in many applications since they may lead the system to inconsistent states. This is why one introduces a strengthening of the agreement condition, called the *uniform agreement* condition, which precludes any disagreement even due to faulty processes:

Uniform agreement: No two processes (whether correct or not) decide differently.

The problem that results from substituting agreement for uniform agreement is called the *uniform consensus* problem. The uniform agreement condition is

clearly not achievable if processes may commit byzantine failures since this failure model imposes no limitation on the possible behaviors, and so on the possible decisions of faulty processes. Basically, this is the reason why consensus, which was first studied in the byzantine failure model, has been originally stated with non uniform conditions [42,36]. Afterwards, the problem specifications that have been studied were often non uniform specifications, even in the setting of benign failures: numerous results have been stated for consensus [27,21,23,22,25,13], and only a few are about uniform consensus [24,41,37].

Consensus in Synchronous Systems. We now give a brief survey of the main results on consensus in synchronous systems.

Various algorithms have been devised, which tolerate crash failures. The most basic one is the *FloodSet* algorithm [37]: each process repeatedly broadcasts the set of values it has ever seen. If at most t processes may crash, the synchronized round model guarantees that all the alive processes have seen exactly the same values throughout the first $t + 1$ rounds. At the end of round $t + 1$, processes agree on the set of values they have seen, and so can use a common decision rule based on this set to decide safely.

As mentioned above, uniform consensus is clearly not solvable in the byzantine failure model no matter the number of faulty processes is. On the other hand, consensus has been shown to be solvable if less than one third of processes are faulty [42,36]. The $n > 3t$ restriction is not accidental: there is no consensus algorithm in a synchronous system with n processes which tolerates t byzantine failures, if $2 \leq n \leq 3t$. Interestingly, the impossibility proof given in [36] is based on a reduction to the case of $n = 3$ and $t = 1$.

Like *FloodSet*, the consensus algorithms in [42,36] which tolerate byzantine failures use $t + 1$ rounds. Indeed, these algorithms are optimal with respect to the number of rounds required for deciding: there is no consensus algorithm, for either type of failure, in which all the alive processes decide by the end of t rounds. This $t + 1$ lower bound has been originally stated by Fischer and Lynch [28] in the case of byzantine failures. The result was then extended to the case of crash failures, first for uniform consensus by Dwork and Moses [25], and subsequently for consensus by Lynch [37]. With respect to this worst case time complexity, consensus and uniform consensus are therefore two equivalent problems in the crash failure model. A simpler proof of this $t+1$ lower bound for crash failures based was presented by Aguilera and Toueg [2]. This latter proof combines a bivalency argument borrowed from [27] and a reduction to systems in which at most one process crashes in each round.

We can refine this analysis by discriminating runs according to the number of failures that *actually* occur: we consider the number of rounds required to decide not over all the runs of an algorithm that tolerates t crash failures, but over all the runs of the algorithm in which at most f processes crash for any $0 \leq f \leq t$. Charron-Bost and Schiper [16] prove that uniform consensus requires at least $f + 2$ rounds whereas consensus requires only $f + 1$ rounds if f is less than $t - 1$. For $f = t - 1$ or $f = t$, the discrepancy between consensus and its

uniform version does not hold anymore: both consensus and uniform consensus require $f+1$ rounds. Dolev, Reischuck, and Strong [22] develop "*early deciding*" consensus algorithms that achieves the general $f+1$ lower bound for consensus: they design an early deciding algorithm such that in each run with at most f failures, all processes have decided by the end of round $f+1$. Consequently, uniform consensus is harder than consensus in the context of the synchronous model with crash failures. For uniform consensus, Charron-Bost and Schiper [16] show that their lower bound is also tight. Note that early deciding algorithms are quite interesting in practice since they are much more efficient in the failure-free case – the most frequent case since failures are casual.

Consensus in Asynchronous Systems. Although consensus can be solvable in synchronous systems in the presence of failures, either benign (crash) or severe (byzantine), this problem is unsolvable in an asynchronous system that is subject to even a *single* crash failure, as established by Fischer, Lynch, and Paterson in their seminal paper [27]. This impossibility result is clearly fundamental by itself, and also by the new concepts introduced for its proof.

Essentially, the impossibility of consensus in the asynchronous case stems from the inherent difficulty for determining whether a process has actually crashed or is only very slow. The proof in [27] is based on this fundamental feature of asynchronous systems, but the implementation of this intuitive argument in a rigorous manner is quite subtle. Fischer, Lynch, and Paterson introduced new ideas that subsequently, have been extensively used for proving other results: the notion of the *valency* of a configuration and the *round-robin* process for constructing failure-free runs.

More precisely, they consider the binary consensus problem, and for each configuration C, they define the set of decision values of the configurations reachable from C, denoted $Val(C)$. If $Val(C) = \{0, 1\}$, then C is said to be *bivalent*; otherwise, C is *univalent*.

Roughly speaking, their proof is structured as follows. For the sake of contradiction, they suppose that there exists a binary consensus algorithm A that tolerates one crash, and then prove the following assertions:

1. Because of the asynchrony of the system, A has an initial bivalent configuration (Lemma 2).
2. If a step s is applicable to a bivalent configuration C, then s can be delayed to yield a new bivalent configuration. More precisely, there is a finite schedules ending with the step s, applicable to C, and leading to a configuration that is still bivalent (Lemma 3).
3. Using a round-robin argument and thanks to Lemma 3, one can construct a failure-free run starting from an initial bivalent configuration provided by Lemma 2, and such that all the configurations in this run are bivalent. In other words, processes remain forever indecisive in this run of A, and so A violates the termination condition of the consensus specification.

The delicate point of this proof is the construction of an "indecisive" run that is admissible, namely a run in which communications are reliable and at most one process crashes.

Interestingly, the impossibility of consensus is quite easier to prove if a majority of processes may be faulty ($n \leq 2t$), using a standard "partioning" argument. We now recall this direct proof.

Proof (of the impossibility of consensus if $n \leq 2t$). The proof is by contradiction. Suppose algorithm A solves consensus in asynchronous systems with $t \geq \lceil n/2 \rceil$. Partition the processes into two sets Π_0 and Π_1 such that Π_0 contains $\lceil n/2 \rceil$ processes, and Π_1 contains the remaining $\lfloor n/2 \rfloor$ processes. Consider the following two runs of A:

- ρ^0 starts from the initial configuration C_0 in which all the initial values are 0. All processes in Π_0 are correct, while those in Π_1 crash at the beginning of the run.
- ρ^1 starts from the initial configuration C_1 in which all the initial values are 1. All processes in Π_1 are correct, while those in Π_0 crash at the beginning of the run.

By validity and termination, all correct processes decide 0 in ρ^0, and 1 in ρ^1. Let σ^0 (resp. σ^1) be the shortest prefix of the schedule associated to ρ^0 (resp. ρ^1) such that all the processes in Π_0 (resp. Π_1) decides 0 (resp. 1) in $\sigma^0(C_0)$ (resp. $\sigma^1(C_1)$). We now consider the initial configuration C where all the initial values of the processes in Π_0 are 0 and those of the processes in Π_1 are 1. The concatenation $\sigma = \sigma^0; \sigma^1$ is a possible schedule of A, applicable to C. By a simple round robin argument,[4] we can extend σ to get a failure free run ρ starting from C which violates the agreement condition – a contradiction.

The impossibility of consensus in asynchronous systems is extremely robust: it still holds when weakening many assumptions of the [27] model. In particular, the proof works even when considering non-deterministic processes, with any type of broadcast communications except atomic broadcast, or if receiving and sending are split into two separate steps. Moreover, Fisher, Lynch, and Paterson prove their impossibility result for a very weak validity condition, that we call the *0-1 validity* condition, and which only stipulates that 0 and 1 are two possible decision value. More formally, this condition is as follows:

0-1 Validity: There exist two runs in which the decision values are 0 and 1, respectively.

[4] More precisely, the run ρ is constructed step by step, starting from $\sigma(C)$: processes of Π are maintained in a queue in an arbitrary order. Each step is executed by the first process in the queue. In every step taken by process p_i, this process receives the earliest sent message in $buffer_i$ or the null message if $buffer_i$ is empty. Every process takes infinitely many steps in ρ and receives every message sent to it. Therefore, ρ is a failure-free run.

3.2 Atomic Commitment

In a distributed database system, ensuring that transactions terminate consistently is a critical task: the sites whose databases where updated by the transaction must agree on whether to *commit* it (i.e., its results will take effect at all the sites) or *abort* it (i.e., its results will be discarded).

More specifically, we consider a collection of processes which participate in the processing of a database transaction. After this processing, each process arrives at an initial "opinion" about whether the transaction ought to be committed or aborted, and *votes* **Yes** or **No**, accordingly. A process votes for committing the transaction if its local computation on behalf of that transaction has been successfully completed, and otherwise will vote for aborting the transaction. The processes are supposed to eventually output a decision, by setting an output variable *decision* to **Commit** or **Abort**. For this problem, the correctness conditions are

Agreement: No two processes decide on different values.
Validity:

1. If any process initially votes **No**, then **Abort** is the only possible decision.
2. If all processes vote **Yes** and there is no failure, then **Commit** is the only possible decision.

with a termination condition which comes into two flavors

Weak termination: If there is no failure, then all processes eventually decide.
Non-blocking termination: All correct processes eventually decide.

The problems specified by these conditions are called *atomic commitment (AC)* for the weak termination and *non-blocking atomic commitment (NB-AC)* for the non-blocking termination. Generally, both AC and NB-AC are studied in the context of the failure model where links are reliable and only processes may fail by crashing.

Clearly, the NB-AC and uniform consensus problems are very similar: when identifying **Yes** and **Commit** with 1, and **No** and **Abort** with 0, they only differ in the validity condition. At this point, it is relevant to consider the *weak validity condition* introduced by Hadzilacos in [31]:

Weak validity: If there is no failure, then any decision value is the initial value of some process.

Obviously, the weak validity condition is weaker than the validity conditions of both consensus and atomic commitment, but stronger than the 0-1 validity condition. Therefore, the impossibility result of [27] immediately implies that NB-AC cannot be solved in asynchronous systems with crash failures, even if only one process may fail. Actually, the impossibility result of NB-AC can be showed by a simple argument which does not require the subtle Lemma 3 of [27]. We now give this direct proof.

Proof (of impossibility of NB-AC). Suppose, to obtain a contradiction, that there is an algorithm A which solves NB-AC and tolerates one failure. We first state a general technical lemma for the asynchronous model.[5] For that, we define the *failure-free valency* of an initial configuration C as the set of decisions which are reachable from C in failure-free runs. This set is denoted by $Val^0(C)$.

Lemma 1. *For any initial configuration C, valency and failure-free valency of C are equal:*

$$Val^0(C) = Val(C).$$

Proof. Obviously, $Val^0(C) \subseteq Val(C)$. Conversely, let $d \in Val(C)$, and ρ be a run of A starting from C in which the decision value is d. By the termination condition, there is a finite prefix σ of ρ such that some process has decided d in the configuration $\sigma(C)$. By a round-robin argument, we construct an extension ρ^0 of σ that is a failure-free run of A. Since decisions are irrevocable and ρ^0 extends σ, the decision value in ρ^0 is d, and so $d \in Val^0(C)$.

Lemma 2. *The initial configuration where all processes vote* **Yes** *is bivalent.*

Proof. Let C_1 be the initial configuration where all processes vote **Yes**. By the second part of the validity condition, **Commit** $\in Val(C_1)$.

Consider a run ρ of A starting from C_1 in which only one process, say p, is faulty and crashes from the beginning. The infinite schedule of events corresponding to ρ is applicable to the initial configuration where all processes vote **Yes** except process p which votes **No**. We so construct a run ρ' of A which is indistinguishable from ρ to any process different than p. In particular, any process different than p decides the same value in ρ and ρ'. By the first part of the validity condition, the decision value in ρ' is **Abort**, and so processes decide **Abort** in ρ. This shows that **Abort** $\in Val(C_1)$, so $Val(C_1) = \{$**Abort**,**Commit**$\}$, as needed.

Combining Lemmas 1 and 2 immediately yields a contradiction with the second part of the validity condition.

Though the impossibility of NB-AC is much easier to prove than the one of uniform consensus, this latter problem cannot be reduced to NB-AC in general. Charron-Bost and Toueg [18] actually show that (1) uniform consensus is not reducible to to NB-AC except if $t = 1$; (2) conversely, NB-AC is never reducible to uniform consensus. In other words, uniform consensus and NB-AC are two agreement problems with quite similar specifications, but which are not comparable.

Contrary to its non-blocking version, AC is attainable in asynchronous systems with crash failures. The simplest and best-known AC algorithm is the *two*

[5] Actually, this lemma holds in any computational model that *displays no failure* [40], i.e., such that any finite schedule applicable to some configuration C has an infinite extension which is a failure-free run. Note that this lemma has been already stated in [21] to prove an impossibility result for consensus.

phase commit (2PC) algorithm [29]. This algorithm and various variations of it are discussed in [10]. Unfortunately, the *2PC* algorithm may cause blocking even when running in synchronous systems. Skeen [47] devised the *three phase commit (3PC)* algorithm – an embellishment of the *2PC* algorithm – which guarantees non-blocking termination in synchronous systems. In the event of timing failures, the *3PC* algorithm may lead to inconsistent decisions, which is probably the main reason why it is not used in practice.

The *3PC* algorithm requires $3n$ rounds. This is much higher than the $t+1$ lower bound for consensus and its uniform version in synchronous systems. As mentioned above, uniform consensus and NB-AC are not comparable, and so the validity condition of NB-AC may yield a different lower bound on the number of rounds required for deciding. Actually, we can observe that the proof of the $t+1$ lower bound for consensus presented by Aguilera and Toueg [2] still works when considering the weak validity condition[6] – a weaker proviso than the validity condition of AC. Therefore, the $t+1$ lower bound also holds for the NB-AC problem. Moreover, the *FloodSet* algorithm can be easily modified to design a version that achieves NB-AC in $t+1$ rounds.

In the *3PC* algorithm, $3t$ rounds may be required to decide. So why is *3PC* an interesting algorithm? The main reason is that in the failure-free case – the most frequent case – , *3PC* requires only 3 rounds to decide. But it is not clear whether 3 rounds are necessary to decide in failure-free runs. More generally, it would be worthy to determine a lower bound for early deciding NB-AC algorithms, that is the number of rounds required to decide in a run of a NB-AC algorithm with at most f $(0 \leq f \leq t)$ failures.

3.3 Atomic Broadcast

One way to implement a fault-tolerant service is by using multiple servers that may fail independently. The state of the service is replicated at these servers, and updates are coordinated so that even when a subset of servers fail, the service remains available. One approach for replication management is the so-called "state-machine approach" or "active replication", which has no centralized control. In this approach, replica coordination and consistency are ensured by enforcing all replicas to receive and process the same sequence of requests. This condition can be achieved by a broadcast primitive, called *atomic broadcast*, which guarantees that all correct processes deliver the same messages in the same order.

Formally, atomic broadcast is a broadcast which satisfies the following four properties:

Validity: If a correct process p broadcasts a message m, then p eventually delivers m.

[6] On the other hand, their proof does not work with the 0-1 validity condition. Indeed, Dwork and Moses [25] devised an algorithm which guarantees agreement, termination, and 0-1 validity in two rounds.

Agreement: If a correct process delivers a message m, then all correct processes eventually deliver m.

Integrity: For any message m, every process delivers m at most once, and only if was previously broadcast.

Total order: If correct processes p and q both deliver messages m and m', then p delivers m before m' if and only if q delivers m before m'.

Atomic broadcast is a problem that involves the achievement of some sort of agreement – namely, agreement on the sequence of messages delivered by correct processes – in a fault-tolerant manner, and so has a common flavor with the consensus problem. Indeed, consensus and atomic broadcast are equivalent in asynchronous systems with crash failures. First, consensus can be easily reduced to atomic broadcast as follows [21]: To propose a value, a process atomically broadcasts it. Every process then decides on the value of the first message that it delivers. Note that this reduction makes no assumption on the system synchrony or topology, and tolerates any number of crash failures. Using the impossibility result for consensus [27], this reduction shows that atomic broadcast cannot be solved in asynchronous systems, even if we assume that at most one process crash.

Conversely, Chandra and Toueg [13] show how to transform any consensus algorithm into an atomic broadcast algorithm. Their transformation uses repeated executions of consensus. Messages that must be delivered are partitioned into batches, and the k-th execution of consensus is used to decide on the k-th batch of messages to be atomically delivered. A precise description of the reduction is given in [30,13]. This reduction shows that atomic broadcast can be solved using randomization, partially synchronous models, or some failure detectors since consensus is solvable in such models (cf. Section 4). Note that the reduction of atomic broadcast into consensus also applies to any asynchronous system and tolerates any number of crash failures. However, it requires that *reliable broadcast* – the weaker broadcast which only guarantees that correct processes deliver the same set of messages – be solvable in the system, and so requires infinite storage [15].

3.4 Group Membership

The problem of *group membership* has been the focus of much theoretical and experimental work on fault-tolerant distributed systems. A group membership protocol manages the formation and maintenance of a set of processes called a *group*. For example, a group may be a set of processes that are cooperating towards a common task (e.g., the primary and backup servers of a database), a set of processes that share a common interest (e.g., clients that subscribe to a particular newsgroup), or the set of all processes in the system that are currently deemed to be operational. In general, a process may *leave* a group because it failed, it voluntarily requested to leave, or it is forcibly expelled by other members of the group. Similarly, a process may *join* a group; for example, it may have been selected to replace a process that has recently left the group. A group membership protocol must manage such dynamic changes in some coherent way: each

process has a *local view* of the current membership of the group, and processes maintain some form of agreement on these local views.

Two types of group membership services have emerged: *primary-partition*, e.g. [45,34,39,38,32], and *partitionable*, e.g. [3,33,49,5,26]. Roughly speaking, a primary-partition group membership service maintains a *single* agreed view of the group (i.e., processes agree on their local views of the group). Such services are intended for systems with no network partitions, or for systems that allow the group membership to change in at most one network partition, the *primary partition*. In contrast, a partitionable group membership service allows *multiple* views of the group to co-exist and evolve concurrently: there may be several disjoint subsets of processes such that processes in each subset agree that they are the current members of the group. In other words, such group membership services allow group splitting (e.g., when the network partitions) and group merging (e.g., when communication between partitions is restored).

The group membership problem was first defined for synchronous systems by [20]. Since then, the group membership problem for asynchronous systems has also been the subject of intense investigation. Yet, despite the wide interest that it has attracted and the numerous publications on this subject, the group membership problem for asynchronous systems is far from being understood. In particular, there is no commonly agreed definition for this problem, and some of the most referenced formal definitions are unsatisfactory [4].

Despite their differences, all versions of the group membership problem require some form of process agreement in systems with failures. The potential for running into an impossibility result as for consensus or atomic commitment is therefore obvious. On the other hand, group membership is different from consensus in at least two ways:

- In group membership, a process that is suspected to have crashed can be *removed* from the group, or even *killed*, even if this suspicion is actually incorrect (e.g., the suspected process was only very slow). The [27] model does not speak about process removals, and it does not directly model process killing (i.e., program-controlled crashes).
- Up to this point, all the specifications of the agreement problems have a termination condition, i.e., require progress in *all* runs. Group membership does allow runs that "do nothing" (for instance, "doing nothing" is desirable when no process wishes to join or leave the group, and no process crashes).

These differences have been widely cited as reasons why group membership is solvable in asynchronous systems while other classical agreement problems are not.

For primary-partition group membership services, Chandra, Charron-Bost, Hadzilacos, and Toueg [14] actually prove that these reasons are not sufficient to make group membership solvable: they define a problem called *WGM* (for *Weak Group Membership*) that allows the removal of erroneously suspected processes from the group, and is subsumed by any reasonable definition of group membership, and show that WGM cannot be solved in asynchronous systems with

failures, even in systems that allow program-controlled process crashes. When looking closer at the implementations of primary-partition group membership, it turns out that they do not satisfy the very weak liveness requirement of WGM. Indeed, they have runs that "block" forever, or remove or kill all processes.[7]

In contrast to primary-partition group membership services, partitionable ones allow processes to disagree on the current membership of the group, i.e., several different views of the membership of the group may evolve concurrently and independently from each other. In particular, there may be several disjoint subsets of processes such that processes in each subset agree that they are the current members of the group. In other words, such group membership services allow *group splitting* (e.g., when the network partitions) and *group merging* (e.g., when communication between partitions is restored).

By allowing disagreement, such group membership services escape from the impossibility result of [14]. However, they run into another fundamental problem: their specification must be strong enough to rule out useless group membership protocols (in particular, protocols that can *capriciously* split groups into singleton sets) and yet it should be weak enough to remain solvable. The design of such a specification is still an open problem.

4 Circumventing Impossibility Results

As explained in Section 3, agreement problems are not solvable in asynchronous systems, even for a single crash failure. However the agreement problems are so fundamental in distributed computing that it is quite important to find ways to cope with this limitation. In order to make agreement problems solvable, we can strengthen the model, weaken the correctness requirements, or both. Three approaches have been thus investigated to circumvent the impossibility of agreement problems: *partial synchrony*, *randomization*, and *unreliable failure detection*.

4.1 Partial Synchrony

This approach is based on the observation that between the synchronous model and the asynchronous model there lie a variety of intermediate models that are called *partially synchronous*. In a partially synchronous system, processes have informations about time, although this information might be partial or inexact. For example, processes in a partially synchronous system might have access to synchronized clocks, or might know bounds on message delivery time or relative processes speeds.

Dolev, Dwork, and Stockmeyer [21] consider the following five synchrony parameters:

1. *synchronous processes:* processes know a bound on process step time;

[7] For example, the implementation of *S-GMP* [45] can crash *all* processes in the system before any new view is installed.

2. *synchronous communication:* processes know a bound on message delivery time;
3. *synchronous message order:* messages are delivered in order of sending;
4. *broadcast transmission:* in an atomic step, a process can broadcast messages to all processes;
5. *atomic receive/send:* receiving and sending are part of the same atomic step.

The third parameter is non classical: actually, synchronous message order is similar to the *causal order* condition [35,46] in which the causality relation is replaced by real-time order. Varying these five parameters yields $2^5 (= 32)$ partially synchronous models. Within the space of these models, [21] precisely delineates the boundary between solvability and unsolvability of consensus: they identify four "minimal" cases in which a consensus algorithm that tolerates $n-1$ crashes exists, but the weakening of any synchrony parameter would yield a model of partial synchrony where consensus is unsolvable. These four minimal cases are:

- synchronous processes and synchronous communication (corresponding to the completely synchronous model);
- synchronous processes and synchronous message order;
- broadcast transmission and synchronous message order;
- synchronous communication, broadcast transmission, and atomic receive /send.

In particular, contrary to synchronous processes, synchronous communication by its own makes consensus solvable in the [27] model (i.e., broadcast transmission and atomic receive/send). In any weaker model with point-to-point communication or separate receive and send, synchronous processes and synchronous communication are both required for the consensus problem to be solvable in the presence of two failures.

Two other partially synchronous models are considered in [23]. The first model assumes that there are bounds on process step time and on message delivery time, but these bounds are not known. The second model assumes that these bounds are known (system is synchronous) but they hold only after some unknown time called *global stabilization time*. In both of these partially synchronous models, consensus is shown to be solvable if the ratio of faulty processes is less than 1/2 for crash failures, and 1/3 for byzantine failures.

The algorithms devised in [23] use the *rotating coordinator* paradigm and proceed in "non-synchronized rounds". The coordinator of each round tries to get other processes to change to some value v that it thinks "acceptable"; it decides v if it receives sufficiently many acknowledgments from the processes that have changed their value to v, so that any value different from v will never be found acceptable in subsequent rounds. This scheme already appears in [44] and has been used many times to design algorithms that may not terminate but never permits disagreement when the system malfunctions (i.e., when synchrony assumptions are not met).

4.2 Randomization

As explained in Section 3.1, there are two fundamental limitations for the consensus problem: the $t + 1$ round lower bound in synchronous systems and the impossibility result of [27] for completely asynchronous systems. Contrary to the partially synchrony approach, randomization helps us to cope with both limitations.

In this approach, we both strengthen the model and weaken the correctness requirements of consensus. On one hand, we augment the system model by including randomization: each process has access to an oracle (usually called a *coin*) that provides random bits, and so processes can make random choices during the computation. On the other hand, the correctness conditions are slightly weaker than previously: validity and agreement are still required but termination must be guaranteed only with probability 1. The precise meaning of this probabilistic termination involves the notion of *adversary*. A randomized distributed algorithm has a high degree of uncertainty, and so a large set of possible executions because of both the nondeterministic choices – that is, which process takes the next step and which message is received – and the probabilistic choices. The nondeterministic choices must be resolved in order to obtain a purely probabilistic system. It is convenient to imagine that the nondeterministic choices are under the control of an adversary.[8] Given a randomized distributed algorithm and the strategy employed by an adversary $\mathcal{A}$, the probabilities of random choices induce a well-defined probability distribution on the set $\mathcal{E}_{\mathcal{A}}$ of executions that are under the control of $\mathcal{A}$. The probabilistic termination condition means that for every adversary $\mathcal{A}$, termination is satisfied with probability 1 in $\mathcal{E}_{\mathcal{A}}$.

The first two randomized consensus algorithms appeared in 1983, and were devised by Ben-Or [9] and Rabin [43] respectively. The two algorithms are similar in flavor, but the principal difference between the two algorithms is in the random oracles that are used. In Rabin's algorithm, the coin is shared by processes which thus see the same random bit. Implicitly, this induces some kind of agreement among processes. Ben-Or was the first to give a truly distributed algorithm where processes flip private coins. In both Rabin's and Ben-Or's work, the original goal was to achieve consensus in a completely asynchronous system, but the resulting algorithms can be run in the synchronous environment, often with high resiliency. Moreover, the expected running time of the synchronous versions of these algorithms are considerably less than the lower bound of $t + 1$ rounds for nonrandomized algorithms.[9] Later on, other randomized algorithms for consensus have been proposed; they differ in various respects: type of adversaries to which they are tolerant, resiliency, expected running time, communication costs (for a very complete survey of randomized consensus algorithms see [19]).

[8] Various types of adversaries can be considered, which are more or less powerful (cf. [19]): adversaries may have limitations on their computing power and on the information that they can obtain from the system. In particular, a *malicious adversary* has knowledge of the past execution, including information about past random choices.

[9] More precisely, this holds for Rabin's algorithm, and for Ben-Or's algorithm with sufficiently small t.

These results about consensus show that contrary to the classical (centralized) algorithms , the randomized distributed model is computationally more powerful. However, randomization does not completely eliminate the lower bounds of consensus (the $t + 1$ lower bound in synchronous systems and the impossibility result for the asynchronous model), but it puts the limitations farther. Actually, by a refinement of the partionning argument of Section 3.1 Bracha and Toueg [11] show that there is no randomized consensus algorithm if more than half of the processes may be faulty ($n \leq 2t$). Furthermore, Bar-Joseph and Ben-Or [6] prove a tight lower bound of $\Theta(t/\sqrt{n \log(2 + t/\sqrt{n})}$ on the expected number of rounds needed for randomized consensus algorithms for the crash failure model.

Until now, we have focused on consensus. But what about the other classical agreement problems? Since atomic broadcast can be reduced to consensus in asynchronous systems with crash failures [13], it can be solved using randomization in such environments. To the best of my knowledge, the randomization approach has not been investigated yet for the atomic commitment and group membership problems.

4.3 Unreliable Failure Detectors

An alternative approach to solve agreement problems in fault-prone asynchronous systems is to strengthen the model by adding a new type of component called a *failure detector*. Since impossibility results for asynchronous systems stem from the inherent difficulty to determine whether a process has actually crashed or is only very slow, the idea is to augment the asynchronous computational model with a failure detection mechanism that can possibly make errors.

More specifically, a failure detector is a distributed oracle that gives some (possibly incorrect) hints about which processes may have crashed so far.[10] Each process has access to a failure detector module that it consults in each step. The notion of failure detector is defined and developed by Chandra and Toueg [13] and by Chandra, Hadzilacos, and Toueg [12]. The precise definitions of a failure detector $\mathcal{D}$ and the computational model of an asynchronous system equipped with $\mathcal{D}$ are given in both papers.

Given two failure detectors $\mathcal{D}$ and $\mathcal{D}'$, Chandra and Toueg [13] formally define what it means for $\mathcal{D}$ to provide at least as much information as $\mathcal{D}'$ does. To do so, they introduce the notion of *reducibility* among failure detectors. Roughly speaking, a failure detector $\mathcal{D}'$ *is reducible to* $\mathcal{D}$ if there is a distributed algorithm that can transform $\mathcal{D}$ into $\mathcal{D}'$. We also say that $\mathcal{D}'$ *is weaker than* $\mathcal{D}$: any problem that can be solved using $\mathcal{D}'$ can also be solved using $\mathcal{D}$ instead. We note $\mathcal{D}' \preceq \mathcal{D}$. The reducibility relation $\preceq$ is transitive, and so define a preorder.[11] Two failure detectors are equivalent if they are reducible to each other.

[10] As an example, a failure detector module can maintain a list of processes that it currently considers to be correct. Such an instance of failure detectors is very close to a group membership service (cf. Section 3.4).

[11] Note that the reducibility relation $\preceq$ is not antisymmetric, in general.

Chandra and Toueg [13] and Chandra, Hadzilacos, and Toueg [12] focus on the consensus problem. They define a failure detector denoted Ω such that the output of the failure detector module of Ω at process p is a single process. Intuitively, when q is the output of Ω at p the failure detector module of Ω at process p currently considers q to be correct; then, we say that p *trusts* q. The Ω failure detector satisfies the following property:

> There is a time after which all the correct processes always trust the same correct process.

In Chandra and Toueg [13], it is shown that consensus can be solved in an asynchronous system equipped with Ω if a majority of processes are correct ($n > 2t$). As for the consensus algorithm devised by Dwork, Lynch, and Stockmeyer [23] for the eventual synchronous model – the partially synchronous model which is synchronous after some unknown time –, their algorithm proceeds in asynchronous rounds and uses the rotating coordinator paradigm. Conversely, Chandra, Hadzilacos, and Toueg [12] show that if a failure detector $\mathcal{D}$ can be used to solve consensus, then Ω is reducible to $\mathcal{D}$, i.e., $\Omega \preceq \mathcal{D}$. Thus Ω is the weakest failure detector for solving consensus in asynchronous systems with a majority of correct processes. On the other hand, Chandra and Toueg [13] show that the consensus problem cannot be solved using Ω if $n \leq 2t$.

Note that the above positive result – namely, consensus is solvable using Ω – is less surprising than it may seem at first sight: indeed, the Ω failure detector ensures that processes eventually agree on the name of a correct process,[12] and consensus is easily solvable if processes trust the same correct process. From this standpoint, the asynchronous model augmented with Ω appears as the natural counterpart of the eventual synchronous model in the failure detector approach.

From the equivalence between atomic broadcast and consensus, we deduce that the same positive and negative results hold for atomic broadcast. But what about the atomic commitment problem? In a recent paper [17], I address this question and point out the impact of the validity condition on the solvability issues of agreement problems in the asynchronous environment. This paper introduces the *binary flag* failure detector, denoted $\mathcal{BF}$, the output of which is the flag **NF** (for No Failure) or **F** (for Failures). This failure detector satisfies the following three properties:

1. If all processes are correct, then no flag is ever **F**.
2. If the flag of some process is **F**, then it remains **F** forever.
3. If some process crashes, then there is a time after which all the flags are **F**.

By comparing the consensus and atomic commitment problems in asynchronous systems equipped with the $\mathcal{BF}$ failure detector, I show that atomic commitment can be solved using $\mathcal{BF}$ if $t = 1$.[13] Conversely, if a failure detector

[12] In fact, Chandra and Toueg [13] solve consensus using another failure detector, denoted $\diamond\mathcal{W}$. The eventual agreement ensured by Ω is far less evident in $\diamond\mathcal{W}$. As a result of their main theorem, Chandra, Hadzilacos, and Toueg [12] show that Ω and $\diamond\mathcal{W}$ are actually equivalent.

[13] Note that if at most one process is faulty ($t = 1$), then Ω is easily reducible to $\mathcal{BF}$.

$\mathcal{D}$ can be used to solve atomic commitment, then $\mathcal{BF}$ is reducible to $\mathcal{D}$, i.e., $\mathcal{D} \preceq \mathcal{SF}$. Thus, $\mathcal{BF}$ is indeed the weakest failure detector for solving atomic commitment in asynchronous systems with $t = 1$.

Note that this latter impossibility result for atomic commitment is far less technical and difficult to prove than the analogous result stated by Chandra, Hadzilacos, and Toueg [12] for consensus. Basically, the proof relies on one simple idea: if all processes initially vote **Yes**, then any atomic commitment algorithm decides **Commit** if and only if there is no failure, and so can be used to detect whether failures occur.

Finally, I prove in [17] that the atomic commitment problem cannot be solved using $\mathcal{BF}$ if $t > 1$. It thus turns out that the $t = n/2$ boundary of consensus is replaced by the $t = 1$ boundary for atomic commitment. Intuitively, this can be explained by the form of the validity conditions: validity of consensus is a symmetric condition which does not refer to the failure pattern; on the other hand, the validity condition of atomic commitment enforces some decision values according to the fact that some failures occur or not.

References

1. Y. Afek, H. Attiya, A. D. Fekete, M. Fischer, N. Lynch, Y. Mansour, D. Wang, and L. Zuck. Reliable communication over unreliable channels. *Journal of the ACM*, 41(6):1267–1297, 1994.
2. Markos Aguilera and Sam Toueg. A simple bivalency-based proof that t-resilient consensus requires $t + 1$ rounds. *Information Processing Letters*, 71(4):155–158, 1999.
3. Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. Membership algorithms for multicast communication groups. In *Proceedings of the Sixth International Workshop on Distributed Algorithms*, volume 647 of *Lecture Notes on Computer Science*, pages 292–312. Springer-Verlag, November 1992.
4. Emmanuelle Anceaume, Bernadette Charron-Bost, Pascale Minet, and Sam Toueg. On the formal specification of group membership services. Technical report, INRIA, Rocquencourt, July 1995.
5. Özalp Babaoğlu, Renzo Davoli, Luigi-Alberto Giachini, and Mary Gray Baker. *RELACS: a communications infrastructure for constructing reliable applications in large-scale distributed systems*. BROADCAST Project deliverable report, 1994. Department of Computing Science, University of Newcastle upon Tyne, UK.
6. Z. Bar-Joseh and Michael Ben-Or. A tight lower bound for randomized synchronous consensus. In *Proceedings of the Seventeenth ACM Symposium on Principles of Distributed Computing*, pages 193–199, August 1998.
7. K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communication of the ACM*, 12(5):260–261, 1969.
8. A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In Ö. Babaoğlu and K. Marzullo, editors, *Proceedings of the Tenth International Workshop on Distributed Algorithms*, volume 1151 of *Lecture Notes on Computer Science*, pages 105–122. Springer-Verlag, October 1996.

9. Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the Second ACM Symposium on Principles of Distributed Computing*, pages 27–30, August 1983.
10. P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
11. Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, October 1985.
12. T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.
13. T. D. Chandra and S. Toueg. Unreliable failure detectors for asynchronous systems. *Journal of the ACM*, 43(2):225–267, March 1996.
14. Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernadette Charron-Bost. On the impossibility of group membership. In *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing*, pages 322–330, Philadelphia, Pennsylvania, USA, May 1996.
15. B. Charron-Bost and A. Schiper. Reliable broadcast is not so easy. Unpublished manuscript., July 2000.
16. B. Charron-Bost and A. Schiper. Uniform consensus is harder than consensus. Technical Report DSC/2000/028, Département Systèmes de Communication, EPFL, May 2000.
17. Bernadette Charron-Bost. The weakest failure detector for solving atomic commitment. In preparation, July 2001.
18. Bernadette Charron-Bost and Sam Toueg. Comparing the atomic commitment and consensus problems. In preparation, January 2001.
19. Benny Chor and Cynthia Dwork. Randomization in byzantine agreement. *Advances in Computer Research*, 5:443–497, 1989.
20. Flaviu Cristian. Reaching agreement on processor group membership in synchronous distributed systems. *Distributed Computing*, 4(4):175–187, April 1991.
21. D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.
22. Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in Byzantine agreement. *Journal of the ACM*, 37(4):720–741, October 1990.
23. C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.
24. C. Dwork and D. Skeen. Patterns of communication in consensus protocols. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 143–153, August 1984.
25. Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, October 1990.
26. Paul D. Ezhilchelvan, Raimundo A. Macêdo, and Santosh K. Shrivastava. Newtop: a fault-tolerant group communication protocol. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, Vancouver, BC, Canada, June 1995.
27. M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
28. Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14:183–186, 1982.
29. James N. Gray. Notes on database operating systems. In R. Bayer, R. M. Graham, and G. Seegmuller, editors, *Operating Systems: An Advanced Course*, volume 66 of

Lecture Notes on Computer Science. Springer-Verlag, 1978. Also appears as IBM Research Laboratory Technical report RJ2188.
30. V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR 94-1425, Cornell University, Dept. of Computer Science, May 1994.
31. Vassos Hadzilacos. On the relationship between the atomic commitment and consensus problems. Workshop on Fault-Tolerant Distributed Computing, March 17-19, 1986, Pacific Grove, CA. Lecture Notes in Computer Science, Vol. 448. Springer-Verlag., 1986.
32. Matti A. Hiltunen and Richard D. Schlichting. Properties of membership services. In *Proceedings of the Second International Symposium on Autonomous Decentralized Systems*, Phoenix, AZ, April 1995.
33. Farnam Jahanian, Sameh Fakhouri, and Ragunathan Rajkumar. Processor group membership protocols: specification, design and implementation. In *Proceeding of the Twelfth IEEE Symposium on Reliable Distributed Systems*, pages 2–11, Princeton, October 1993.
34. M. Frans Kaashoek and Andrew S. Tanenbaum. Group communication in the amoeba distributed operating system. In *Proceedings of the Eleventh International Conference on Distributed Computer Systems*, pages 222–230, Arlington, TX, May 1991.
35. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
36. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.
37. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
38. P.M. Melliar-Smith, Louise Moser, and Vivek Agrawala. Processor membership in asynchronous distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):459–473, May 1994.
39. Shivakant Mishra, Larry L. Peterson, and Richard D. Schlichting. A membership protocol based on partial order. In *Proceedings of the IEEE International Working Conference on Dependable Computing For Critical Applications*, pages 137–145, Tucson, AZ, February 1991.
40. Yoram Moses and Sergio Rajsbaum. The unified structure of consensus: a layered analysis approach. In *Proceedings of the Seventeenth ACM Symposium on Principles of Distributed Computing*, pages 123–132, August 1998.
41. G. Neiger and S. Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, 1990.
42. Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.
43. Michael Rabin. Randomized Byzantine generals. In *Proceedings of the Twenty-Fourth Symposium on Foundations of Computer Science*, pages 403–409. IEEE Computer Society Press, November 1983.
44. Rüdiger Reischuk. A new solution for the Byzantine general's problem. Technical Report RJ 3673, IBM Research Laboratory, November 1982.
45. Aleta Ricciardi and Ken Birman. Using process groups to implement failure detection in asynchronous environments. In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 341–351. ACM Press, August 1991.
46. Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

47. Dale Skeen. Nonblocking commit protocols. In *Proceedings of the ACM SIGMOD Conf. on Management of Data*, pages 133–147. ACM, June 1982.
48. N. V. Stenning. A data transfer protocol. *Computer Networks*, 1(2):99–110, 1976.
49. Robbert van Renesse, Kenneth P. Birman, Robert Cooper, Bradford Glade, and Patrick Stephenson. The horus system. In Kenneth P. Birman and Robbert van Renesse, editors, *Reliable Distributed Computing with the Isis Toolkit*, pages 133–147. IEEE Computer Society Press, Los Alamitos, CA, 1993.

Negotiating the Semantic Gap: From Feature Maps to Semantic Landscapes

William I. Grosky[1] and Rong Zhao[2]

[1] Computer and Information Science Department, University of Michigan-Dearborn, Dearborn, Michigan 48128
wgrosky@umich.edu

[2] Computer Science Department, State University of New York at Stony Brook, Stony Brook, New York 11794
roz@cs.sunysb.edu

Abstract. In this paper, we present the results of our work that seeks to negotiate the gap between low-level features and high-level concepts in the domain of web document retrieval. This work concerns a technique, latent semantic indexing (LSI), which has been used for textual information retrieval for many years. In this environment, LSI determines clusters of co-occurring keywords, sometimes, called concepts, so that a query which uses a particular keyword can then retrieve documents perhaps not containing this keyword, but containing other keywords from the same cluster. In this paper, we examine the use of this technique for content-based web document retrieval, using both keywords and image features to represent the documents.

1 Introduction

The emergence of multimedia technology and the rapidly expanding image and video collections on the internet have attracted significant research efforts in providing tools for effective retrieval and management of visual data. Image retrieval is based on the availability of a representation scheme of image content. Image content descriptors may be visual features such as color, texture, shape, and spatial relationships, or semantic primitives.

Conventional information retrieval was based solely on text, and those approaches to textual information retrieval have been transplanted into image retrieval in a variety of ways. However, “a picture is worth a thousand words”. Image contents are much more versatile compared with texts, and the amount of visual data is already enormous and still expanding very rapidly. Hoping to cope with these special characteristics of visual data, content-based image retrieval methods have been introduced. It has been widely recognized that the family of image retrieval

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 33–52, 2001.

techniques should become an integration of both low-level visual features addressing the more detailed perceptual aspects and high-level semantic features underlying the more general conceptual aspects of visual data. Neither of these two types of features is sufficient to retrieve or manage visual data in an effective or efficient way [1]. Although efforts have been devoted to combining these two aspects of visual data, the gap between them is still a huge barrier in front of researchers. Intuitive and heuristic approaches do not provide us with satisfactory performance. Therefore, there is an urgent need of finding the latent correlation between low-level features and high-level concepts and merging them from a different perspective. How to find this new perspective and bridge the gap between visual features and semantic features has been a major challenge in this research field.

1.1 Image Retrieval

Image retrieval is an extension to traditional information retrieval. Its purpose is to retrieve images, from a data source (usually a database or the entire internet), that are relevant to a piece of query data. Approaches to image retrieval are somehow derived from conventional information retrieval and are designed to manage the more versatile and enormous amount of visual data which exists.

The different types of information items that are normally associated with images are as follows:

- Content-independent metadata: data that is not directly concerned with image content, but related to it. Examples are image format, author's name, date and location.
- Content metadata:
  - Content-dependent metadata: data referring to low-level or intermediate-level features, such as color, texture, shape, spatial relationships, and their various combinations.
  - Content-descriptive metadata: data referring to content semantics, concerned with relationships of image entities to real-world entities.

Low-level visual features such as color, texture, shape and spatial relationships are directly related to perceptual aspects of image content. Since it is usually easy to extract and represent these features and fairly convenient to design similarity measures by using the statistical properties of these features, a variety of content-based image retrieval techniques have been proposed in the past few years. High-level concepts, however, are not extracted directly from visual contents, but they represent the relatively more important meanings of objects and scenes in the images that are perceived by human beings. These conceptual aspects are more closely related to users' preferences and subjectivity. Concepts may vary significantly in different circumstances. Subtle changes in the semantics may lead to dramatic conceptual differences. Needless to say, it is a very challenging task to extract and manage meaningful semantics and to make use of them to achieve more intelligent and user-friendly retrieval. The next section analyzes these challenges in more detail.

1.2 Challenges

High-level conceptual information is normally represented by using text descriptors. Traditional indexing for image retrieval is text-based. In certain content-based retrieval techniques, text descriptors are also used to model perceptual aspects. However, the inadequacy of text description is very obvious:

- It is difficult for text to capture the perceptual saliency of visual features.
- It is rather difficult to characterize certain entities, attributes, roles or events by means of text only.
- Text is not well suited for modeling the correlation between perceptual and conceptual features.
- Text descriptions reflect the subjectivity of the annotator and the annotation process is prone to be inconsistent, incomplete, ambiguous, and very difficult to be automated.

Although it is an obvious fact that image contents are much more complicated than textual data stored in traditional databases, there is an even greater demand for retrieval and management tools for visual data, since visual information is a more capable medium of conveying ideas and is more closely related to human perception of the real world. Image retrieval techniques should provide support for user queries in an effective and efficient way, just as conventional information retrieval does for textual retrieval [2]. In general, image retrieval can be categorized into the following two types:

- Exact Matching – This category is applicable only to static environments or environments in which features of the image do not evolve over an extended period of time. Databases containing industrial and architectural drawings, or electronics schematics are examples of such environments.
- Similarity-Based Searching – In most cases, it is not quite obvious to know which images best satisfy the query. Different users may have different ideas. Even the same user may have different preferences under different circumstances. Thus, it is desirable to return the top several similar images based on the similarity measure, so as to give users a good sampling. User interaction plays an important role in this type of retrieval. Databases containing natural scenes or human faces are examples of such environments.

For either type of retrieval, the dynamic and versatile characteristics of image content require expensive computations and sophisticated methodologies in the areas of computer vision, image processing, data visualization, indexing, and similarity measurement. In order to manage image data effectively and efficiently, many schemes for data modeling and image representation have been proposed. Typically, each of these schemes builds a symbolic image for each given physical image to provide logical and physical data independence. Symbolic images are then used in conjunction with various index structures as proxies for image comparisons to reduce the searching scope. The high-dimensional visual data is usually reduced into a lower-dimensional subspace so that it is easier to index and manage the visual contents. Once the similarity measure has been determined, indexes of corresponding images are located in the image space and those images are retrieved from the

database. Due to the lack of any unified framework for image representation and retrieval, certain methods may perform better than others under certain query situations. Therefore, these schemes and retrieval techniques have to be somehow integrated and adjusted on the fly to facilitate effective and efficient image data management.

1.3 Research Goals

The work presented in this chapter aims to improve several aspects of content-based image retrieval by finding the latent correlation between low-level visual features and high-level semantics and integrating them into a unified vector space model. To be more specific, the significance of this approach is to design and implement an effective and efficient framework of image retrieval techniques, using a variety of visual features such as color, texture, shape and spatial relationships. Latent semantic indexing, an information retrieval technique, is incorporated with content-based image retrieval. By using this technique, we hope to extract the underlying semantic structure of image content and hence to bridge the gap between low-level visual features and high-level conceptual information. Improved retrieval performance and more efficient indexing structure can also be achieved. We have investigated the following issues in our preliminary research and the experimental results are very promising. Our goals are as follows:

- We aim to present a novel approach based on latent semantic indexing to image retrieval and explore how it helps to reveal the latent correlation between feature sets and semantic clusters.
- We aim to experiment with a special feature extraction method, namely, the *anglogram*, which captures the spatial distribution of feature points. This technique is based on the extraction of information from a Delauney triangulation of these feature points.
- We aim to present a unified framework to integrate multiple visual features including color histograms, shape anglograms and color anglograms with latent semantic indexing and demonstrate the efficacy of our image indexing scheme by comparing it with relevant image indexing techniques.
- We aim to incorporate textual annotation with visual features in the proposed framework of image retrieval and indexing to further our efforts of negotiating the semantic gap. Relevance feedback and other techniques will also be integrated.

The remaining part of this chapter is organized as follows. Section 2 introduces the feature extraction techniques applied in our approach. The theoretical background of latent semantic indexing and its application in textual information retrieval are detailed in Section 3. In Section 4, we present the preliminary results of our study of finding the latent correlation between features and semantics. Finally, Section 5 summarizes the chapter and highlights some proposed future work.

2 Features

In this section we present the feature extraction techniques that are applied in this research work. We propose to integrate a variety of visual features with the latent semantic indexing technique for image retrieval. These visual features include global and subimage color histograms, as well as anglograms. Anglograms can be used for shape-based and color-based representations, as well as for the spatial-relationships of image objects. Thus, a unified framework of image retrieval techniques is going to be generated in our proposed study.

2.1 Color Histogram

Color is a visual feature that is immediately perceived when looking at an image. Retrieval by color similarity requires that models of color stimuli are used, such that distances in the color space correspond to human perceptual distances between different colors. Moreover, color patterns must be represented in such a way that salient chromatic properties are captured.

A variety of color models have been introduced, such as *RGB, HSV, CIE, LUV and MTM*. Humans perceive colors through hue, saturation, and brightness. *Hue* describes the actual wavelength of the color. *Saturation* indicates the amount of white light that is present in a color. Highly saturated colors, also known as pure colors, have no white light component. *Brightness*, which is also called intensity, value, or lightness, represents the intensity of color. Since the combination of hue, saturation and value reflects human perception of color, the *HSV* color model has been selected to be the basis for our color-based extraction approach.

The *color histogram* is the most traditional and the most widely used way to represent color patterns in an image. It is a relatively efficient representation of color content and it is fairly insensitive to variations originated by camera rotation or zooming [1]. Also, it is fairly insensitive to changes in image resolution when images have quite large homogeneous regions, and insensitive to partial occlusions as well.

In our study, the *HSV* color histogram is generated for each image on either the whole image level or the subimage level. On whole image level, a two-dimensional global histogram of both the hue component and saturation component is computed. Since the human perception of color depends mostly on hue and saturation, we ignore the intensity value component in our preliminary research, in order to simplify the computation. Each image is first converted from the *RGB* color space to the *HSV* color space. For each pixel of the resulting image, hue and saturation are extracted and each quantized into a 10-bin histogram. Then, the two histograms h and s are combined into one $h \times s$ histogram with 100 bins, which is taken to be the representing feature vector of each image. This is a vector of 100 elements, $\mathbf{V} = [f_1, f_2, f_3, \ldots f_{100}]^T$, where each element corresponds to one of the bins in the hue-saturation histogram.

On the subimage level, each image is decomposed into 5 subimages, which is illustrated by the sample image in Figure 1. Such an approach was used in [3], and is a step toward identifying the *semcons* [4] appearing in an image. Considering that it is very common to have the major object located in central position in the image, we have one subimage to capture the central region in each image, and the other four subimages cover the upper-left, upper-right, lower-left, and lower-right areas in the image. For each pixel of the resulting subimage, hue and saturation are extracted and each quantized into a 10-bin histogram. Then the two histograms h and s are again combined into one $h \times s$ histogram with 100 bins, which is taken to be the representing feature vector of each image. This is a vector of 100 elements, $\mathbf{V} = [f_1, f_2, f_3, \ldots f_{100}]^T$, where each element again corresponds to one of the bins in the hue-saturation histogram.

Fig. 1. Subimage Decomposition

Since both global and subimage color histograms are formulated as a feature vector, it is very easy to use them as the input for latent semantic indexing.

2.2 Anglogram

In this section, we first provide some background concepts for Delaunay triangulation in computational geometry, and then present the geometric triangulation-based *anglogram* for encoding spatial correlation, which is invariant to translation, scale, and rotation.

Let $P = \{ p_1, p_2, \ldots, p_n \}$ be a set of points in the two-dimensional Euclidean plane, namely the *sites*. Partition the plane by labeling each point in the plane to its nearest site. All those points labeled as p_i form the *Voronoi region* $V(p_i)$. $V(p_i)$ consists of all the points x, which are at least as close to p_i as to any other site:

$$V(p_i) = \{ x: | p_i - x| \leq | p_j - x|, \forall j \neq i \}.$$

Some of the points do not have a unique nearest site, however. The set of all points that have more than one nearest site form the *Voronoi diagram* $V(P)$ for the set of sites.

Construct the *dual* graph G for a Voronoi Diagram $V(P)$ as follows: the nodes of G are the sites of $V(P)$, and two nodes are connected by an arc if their corresponding Voronoi polygons share a Voronoi edge. In 1934, Delaunay proved that when the

dual graph is drawn with straight lines, it produces a planar triangulation of the Voronoi sites *P*, so called the *Delaunay triangulation D(P)*. Each face of *D(P)* is a triangle, so called the *Delaunay triangle*.

The spatial layout of a set of points can be coded through such an *anglogram*. One discretizes and counts the angles produced by the Delaunay triangulation of a set of unique feature points in some context, given the selection criteria of what the bin size will be and of which angles will contribute to the final angle histogram. An important property of our proposed anglogram for encoding spatial correlation is its invariance to translation, scale, and rotation. An O(*max(N, #bins)*) algorithm is necessary to compute the anglogram corresponding to the Delaunay triangulation of a set of *N* points.

The *shape anglogram* approach can be used for image object indexing, while the *color anglogram* can be used as a spatial color representation technique.

In the *shape anglogram* approach, those objects that will be used to index the image are identified, and then a set of high-curvature points along the object boundary are obtained as the feature points. The Delaunay triangulation is performed on these feature points and thus the feature point histogram is computed by discretizing and counting the number of either the two largest angles or the two smallest angles in the Delaunay triangles.

To apply the *color anglogram* approach, color features and their spatial relationship are extracted and then coded into the Delaunay triangulation. Each image is decomposed into a number of non-overlapping blocks. Each individual block is abstracted as a unique feature point labeled with its spatial location and feature values. The feature values in our experiment are dominant or average hue and saturation in the corresponding block. Then, all the normalized feature points form a point feature map for the corresponding image. For each set of feature points labeled with a particular feature value, the Delaunay triangulation is constructed and then the feature point histogram is computed by discretizing and counting the number of either the two largest angles or the two smallest angles in the Delaunay triangles. Finally, the image will be indexed by using the concatenated feature point histogram for each feature value.

3 Latent Semantic Indexing

In this section we describe an approach to automatic information indexing and retrieval, namely, *latent semantic indexing (LSI)*. It is introduced to overcome a fundamental problem that plagues existing retrieval techniques that try to match words of queries with words of documents. The problem is that users want to retrieve on the basis of conceptual content, while individual words provide unreliable evidence about the conceptual meaning of a document. There are usually many ways to express a given concept. Therefore, the literal terms used in a user's query may not match those of a relevant document. In addition, most words have multiple meanings and are used in different contexts. Hence, the terms in a user's query may literally match the terms in documents that are not of any interest to the user at all.

In information retrieval these two problems are addressed as *synonymy* and *polysemy*. The concept *synonymy* is used in a very general sense to describe the fact that there are many ways to refer to the same object. Users in different contexts, or with different needs, knowledge, or linguistic habits will describe the same concept using different terms. The prevalence of synonyms tends to decrease the *recall* performance of the retrieval. By *polysemy* we refer to the general fact that most words have more than one distinct meaning. In different contexts or when used by different people the same term takes on varying referential significance. Thus the use of a term in a query may not necessarily mean that a document containing the same term is relevant at all. Polysemy is one factor underlying poor *precision* performance of the retrieval [5].

Latent semantic indexing tries to overcome the deficiencies of term-matching retrieval by treating the unreliability of observed term-document association data as a statistical problem. It is assumed that there exists some underlying latent semantic structure in the data that is partially obscured by the randomness of word choice with respect to retrieval. Statistical techniques are used to estimate this latent semantic structure, and to get rid of the obscuring noise. By semantic structure we mean the correlation structure in which individual words appear in documents; semantic implies only the fact that terms in a document may be taken as referents to the document itself or to its topic.

The latent semantic indexing technique makes use of the singular value decomposition (SVD). We take a large matrix of term-document association data and construct a semantic space wherein terms and documents that are closely associated are placed near to each other. Singular value decomposition allows the arrangement of the space to reflect the major associative patterns in the data, and ignore the smaller, less important influences. As a result, terms that did not actually appear in a document may still end up close to the document, if that is consistent with the major patterns of association in the data. Position in the space then serves as a new kind of semantic indexing. Retrieval proceeds by using the terms in a query to identify a point in the semantic space, and documents in its neighborhood are returned as relevant results to the query.

Latent semantic indexing is based on the fact that the term-document association can be formulated by using the vector space model, in which each document is encoded as a vector, where each vector component reflects the importance of a particular term in representing the semantics of that document. The vectors for all the documents in a database are stored as the columns of a single matrix. Latent semantic indexing is a variant of the vector space model in which a low-rank approximation to the vector space representation of the database is employed. That is, we replace the original matrix by another matrix that is as close as possible to the original matrix but whose column space is only a subspace of the column space of the original matrix. Reducing the rank of the matrix is a means of removing extraneous information or noise from the database it represents. Rank reduction is used in various applications of linear algebra and statistics as well as in image processing, data compression, cryptography, and seismic tomography. According to [6], latent semantic indexing

has achieved average or above average performance in several experiments with the TREC collections.

3.1 The Vector-Space Model

In the vector space model, a vector is used to represent each item or *document* in a collection. Each component of the vector reflects a particular concept, keyword, or term associated with the given document. The value assigned to that component reflects the importance of the term in representing the semantics of the document. Typically, the value is a function of the frequency with which the term occurs in the document or in the document collection as a whole [7].

A database containing a total of d documents described by t terms is represented as a $t \times d$ *term-by-document matrix A*. The d vectors representing the d documents form the columns of the matrix. Thus, the matrix element a_{ij} is the weighted frequency at which term i occurs in document j. The columns of A are called the *document vectors*, and the rows of A are the *term vectors*. The semantic content of the database is contained in the column space of A, meaning that the document vectors span that content. We can exploit geometric relationships between document vectors to model similarity and differences in content. Meanwhile, we can also compare term vectors geometrically in order to identify similarity and differences in term usage.

A variety of schemes are available for weighting the matrix elements. The element a_{ij} of the term-by-document matrix A is often assigned values as $a_{ij} = l_{ij}g_i$. The factor g_i is called the *global weight*, reflecting the overall value of term i as an indexing term for the entire collection. As one example, consider a very common term like *image* within a collection of articles on image retrieval. It is not important to include that term in the description of a document as all of the documents are known to be about image so a small value of the global weight g_i is appropriate. Global weighting schemes range from simple normalization to advanced statistics-based approaches [7]. The factor l_{ij} is a local weight that reflects the importance of term i within document j itself. Local weights range in complexity from simple binary values to functions involving logarithms of term frequencies. The latter functions have a smoothing effect in that high-frequency terms having limited discriminatory value are assigned low weights.

3.2 Singular-Value Decomposition

The singular value decomposition (SVD) is a dimension reduction technique which gives us reduced-rank approximations to both the column space and row space of the vector space model. The SVD also allows us to find a rank-k approximation to a matrix A with minimal change to that matrix for a given value of k [6]. The decomposition is defined as $A = U \Sigma V^T$, where U is the $t \times t$ orthogonal matrix having the left singular vectors of A as its columns, V is the $d \times d$ orthogonal matrix having the right singular vectors of A as its columns, and Σ is the $t \times d$ diagonal matrix

having the singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r$ of the matrix A in order along its diagonal, where $r = min(t, d)$. This decomposition exists for any given matrix A [8].

The rank r_A of the matrix A is equal to the number of nonzero singular values. It follows directly from the orthogonal invariance of the *Frobenius* norm that $\| A \|_F$ is defined in terms of those values,

$$\|A\|_F = \|U\Sigma V^T\|_F = \|\Sigma V^T\|_F = \|\Sigma\|_F = \sqrt{\sum_{j=1}^{r_A} \sigma_j^2}$$

The first r_A columns of matrix U are a basis for the column space of matrix A, while the first r_A rows of matrix V^T are a basis for the row space of matrix A. To create a rank-k approximation A_k to the matrix A, where $k \leq r_A$, we can set all but the k largest singular values of A to be zero. A classic theorem about the singular value decomposition states that the distance between the original matrix A and its rank-k approximation is minimized by the approximation A_k. The theorem further shows how the norm of that distance is related to singular values of matrix A. It is described as

$$\|A - A_k\|_F = \min_{rank(X) \leq k} \|A - X\|_F = \sqrt{\sigma_{k+1}^2 + \ldots + \sigma_{r_A}^2}$$

Here $A_k = U_k \Sigma_k V_k^T$, where U_k is the $t \times k$ matrix whose columns are the first k columns of matrix U, V_k is the $d \times k$ matrix whose columns are the first k columns of matrix V, and Σ_k is the $k \times k$ diagonal matrix whose diagonal elements are the k largest singular values of matrix A.

How to choose the rank that provides optimal performance of latent semantic indexing for any given database remains an open question and is normally decided via empirical testing. For very large databases, the number of dimensions used usually ranges between 100 and 300. Normally, it is a choice made for computational feasibility as opposed to accuracy. Using the SVD to find the approximation A_k, however, guarantees that the approximation is the best that can be achieved for any given choice of k.

3.3 Similarity Measure

In the vector space model, a user queries the database to find relevant documents, using the vector space representation of those documents. The query is also a set of terms, with or without weights, represented by using a vector just like the documents. It is likely that many of the terms in the database do not appear in the query, meaning that many of the query vector components are zero. Meanwhile, even though some of the terms in the query and in the documents are common, they may be used to refer to different concepts. Considering the general problems of *synonymy* and *polysemy*, we are trying to reveal the underlying semantic structure of the database and thus improve the query performance by using the latent semantic indexing technique. A query can be issued after the SVD has been performed on the database and an appropriate lower rank approximation has been generated. The matching process is to find the documents most similar to the query in the use and weighting of terms. In the

vector space model, the documents selected are those geometrically closest to the query in the transformed semantic space.

One common measure of similarity is the cosine of the angle between the query and document vectors. If the term-by-document matrix A has columns a_j, $j = 1, 2, \ldots, d$, those d cosines are computed according to the following formula

$$\cos\theta_j = \frac{a_j^T q}{\|a_j\|_2 \|q\|_2} = \frac{\sum_{i=1}^{t} a_{ij} q_i}{\sqrt{\sum_{i=1}^{t} a_{ij}^2} \sqrt{\sum_{i=1}^{t} q_i^2}}$$

for $j = 1, 2, \ldots, d$, where the Euclidean vector norm $\| x \|_2$ is defined by

$$\|x\|_2 = \sqrt{x^T x} = \sqrt{\sum_{i=1}^{t} x_i^2}$$

for any t-dimensional vector x. Because the query and document vectors are typically sparse, the dot product and norms are generally inexpensive to compute. Furthermore, the document vector norms $\|a_j\|_2$ need to be computed only once for any given term-by-document matrix. Note that multiplying either a_j or q by a constant does not change the cosine value, thus, we may scale the document vectors or the queries by any convenient factor.

With any given document database and user's query, we can always generate the term-by-document matrix and then apply the singular value decomposition to this matrix. We hope to choose a good lower-ranked approximation after the SVD and use this transformed matrix to construct the semantic space of the database. Then, the query process will be to locate those documents geometrically closest to the query vector in the semantic space.

The latent semantic indexing technique has been successfully applied to information retrieval, in which it shows distinctive power of finding the latent correlation between terms and documents. This inspires us to attempt to borrow this technique from traditional information retrieval and apply it to visual information retrieval. We hope to make use of the power of latent semantic indexing to reveal the underlying semantic nature of visual contents, and thus to find the correlation between visual features and semantics of visual documents or objects. We will explore this approach further by correlating low-level feature groups and high-level semantic clusters, hoping to figure out the semantic nature behind those visual features. Some preliminary experiments have been conducted and the results show that integrating latent semantic indexing with content-based retrieval is a promising approach. Details of these experiments are presented in the next section.

4 Finding Latent Correlation between Visual Features and Semantics

Existing management systems for image collections and their users are typically at cross-purposes. While these systems normally retrieve images based on low-level features, users usually have a more abstract notion of what will satisfy them. Using low-level features to correspond to high-level abstractions is one aspect of the *semantic gap* [9] between content-based system organization and the concept-based user. Sometimes, the user has in mind a concept so abstract that he himself doesn't know what he wants until he sees it. At that point, he may want images similar to what he has just seen or can envision. Again, however, the notion of similarity is typically based on high-level abstractions, such as activities taking place in the image or evoked emotions. Standard definitions of similarity using low-level features generally will not produce good results.

In reality, the correspondence between user-based semantic concepts and system-based low-level features is many-to-many. That is, the same semantic concept will usually be associated with different sets of image features. Also, for the same set of image features, different users could easily find dissimilar images relevant to their needs, such as when their relevance depends directly on an evoked emotion.

In this section, we present the results of a series of experiments that seeks to transform low-level features to a higher level of meaning. This study concerns a technique, latent semantic analysis, which has been used for information retrieval for many years. In this environment, this technique determines clusters of co-occurring keywords, sometimes, called *concepts*, so that a query which uses a particular keyword can then retrieve documents perhaps not containing this keyword, but containing other keywords from the same cluster. In this preliminary study, we examine the use of this technique for content-based image retrieval to find the correlation between visual features and semantics.

4.1 The Effects of Latent Semantic Indexing, Normalization, and Weighting for Global and Subimage Color Histograms

In this and the next section, we show the improvement that latent semantic analysis, normalization, and weighting can give to two simple and straightforward image retrieval techniques, both of which use standard color histograms. For our experiments, we use a database of 50 JPEG images, each of size 192×128. This image collection consists of ten semantic categories of five images each. The categories consist of: ancient towers, ancient columns, birds, horses, pyramids, rhinos, sailing scenes, skiing scenes, sphinxes, and sunsets.

Our first approach uses global color histograms. Each image is first converted from the RGB color space to the HSV color space. For each pixel of the resulting image, hue and saturation are extracted and each quantized into a 10-bin histogram. Then the two histograms h and s are combined into one $h \times s$ histogram with 100 bins, which is the representing feature vector of each image. This is a vector of 100 elements, $\mathbf{V} =$

$[f_1, f_2, f_3, \ldots f_{100}]^T$, where each element corresponds to one of the bins in the hue-saturation histogram.

We then generate the feature-image-matrix, $\mathbf{A} = [\mathbf{V}_1,\ldots,\mathbf{V}_{50}]$, which is 100×50. Each row corresponds to one of the elements in list of features and each column is the entire feature vector of the corresponding image. This matrix is written into a file so the computation is done only once. The matrix will be retrieved from the file during the query process.

A singular value decomposition is then performed on the feature-image-matrix. The result comprises three matrices, **U, S** and **V**, where $\mathbf{A} = \mathbf{USV}^T$. The dimensions of **U** are 100×100, **S** is 100×50, and **V** is 50×50. The rank of matrix **S**, and thus the rank of matrix **A**, in our case is 50. Therefore, the first 50 columns of **U** spans the column space of **A** and all the 50 rows in $\mathbf{V}^T$ spans the row space of **A. S** is a diagonal matrix of which the diagonal elements are the singular values of **A**. To reduce the dimensionality of the transformed latent semantic space, we use a rank-*k* approximation, $\mathbf{A}_k$, of the matrix **A**, for k = 34, which worked better than other values tried. This is defined by $\mathbf{A}_k = \mathbf{U}_k\mathbf{S}_k\mathbf{V}_k^T$. The dimension of $\mathbf{A}_k$ is the same as **A**, 100 by 50. The dimensions of $\mathbf{U}_k$, $\mathbf{S}_k$, and $\mathbf{V}_k$ are 100×34, 34×34, and 50×34, respectively.

The query process in this approach is to compute the distance between the transformed feature vector of the query image, **q**, and that of each of the 50 images in the database, **d**. This distance is defined as $dist(\mathbf{q}, \mathbf{d}) = \mathbf{q}^T\mathbf{d} / \|\mathbf{q}\| \|\mathbf{d}\|$, where $\|\mathbf{q}\|$ and $\|\mathbf{d}\|$ are the norms of those vectors. The computation of $\|\mathbf{d}\|$ for each of the 50 images is done only once and then written into a file. Using each image as a query, in turn, we find the average sum of the positions of all of the five correct answers. Note that in the best case, where the five correct matches occupy the first five positions, this average sum would be 15, whereas in the worst case, where the five correct matches occupy the last five positions, this average sum would be 240. A measure that we use of how good a particular method is defined as,

$$measure - of - goodness = \frac{48 - average - sum/5}{45}.$$

We note that in the best case, this measure is equal to 1, whereas in the worst case, it is equal to 0.

This approach was then compared to one without using latent semantic analysis. We also wanted to see whether the standard techniques of normalization and term weighting from text retrieval would work in this environment.

The following *normalization* process will assign equal emphasis to each component of the feature vector. Different components within the vector may be of totally different physical quantities. Therefore, their magnitudes may vary drastically and thus bias the similarity measurement significantly. One component may overshadow the others just because its magnitude is relatively too large. For the feature image matrix $\mathbf{A}=[\mathbf{V}_1,\mathbf{V}_2, \ldots, \mathbf{V}_{50}]$, we have $\mathbf{A}_{i,j}$ which is the i^{th} component in vector $\mathbf{V}_j$. Assuming a Gaussian distribution, we can obtain the mean μ_i and standard deviation σ_i for the i^{th} component of the feature vector across the whole image

database. Then we normalize the original feature image matrix into the range of [-1,1] as follows,

$$A_{i,\,j} = \frac{A_{i,\,j} - \mu_i}{\sigma_i}.$$

It can easily be shown that the probability of an entry falling into the range of [-1, 1] is 68%. In practice, we map all the entries into the range of [-1, 1] by forcing the out-of-range values to be either –1 or 1. We then shift the entries into the range of [0, 1] by using the following formula

$$A_{i,\,j} = \frac{A_{i,\,j} + 1}{2}.$$

After this normalization process, each component of the feature image matrix is a value between 0 and 1, and thus will not bias the importance of any component in the computation of similarity.

One of the common and effective methods for improving full-text retrieval performance is to apply different weights to different components [7]. We apply these techniques to our image environment. The raw frequency in each component of the feature image matrix, with or without normalization, can be weighted in a variety of ways. Both global weight and local weight are considered in our approach. A *global weight* indicates the overall importance of that component in the feature vector across the whole image collection. Therefore, the same global weighting is applied to an entire row of the matrix. A *local weight* is applied to each element indicating the relevant importance of the component with its vector. The value for any component $\mathbf{A}_{i,\,j}$ is thus *L(i, j)G(i),* where *L(i, j)* is the local weighting for feature component *i* in image *j*, and *G(i)* is the global weighting for that component.

Common local weighting techniques include term frequency, binary, and log of term frequency, whereas common global weighting methods include *Normal*, *GfIdf*, *Idf*, and *Entropy*. Based on previous research it has been found that log of term frequency helps to dampen effects of large differences in frequency and thus has the best performance as a local weight, whereas Entropy is the appropriate method for global weighting [7].

The entropy method is defined by having a component global weight of,

$$1 + \sum_j \frac{p_{ij} \log(p_{ij})}{\log(number\_of\_images)}$$

where pij = tf_{ij} / gf_i is the probability of that component, tf_{ij} is the raw frequency of component $\mathbf{A}_{i,j}$, and gf_i is the global frequency, i.e., the total number of times that component *i* occurs in the whole collection.

The global weights give less emphasis to those components that occur frequently or in many images. Theoretically, the entropy method is the most sophisticated weighting scheme and it takes the distribution property of feature components over the image collection into account.

We conducted similar experiments for these four cases:

1. Global color histograms, no normalization, no term weighting, no latent-semantic indexing (raw data)

2. Global color histograms, normalized and term-weighted, no latent semantic indexing
3. Global color histograms, no normalization, no term-weighting, with latent semantic indexing
4. Global color histograms, normalized and term-weighted, with latent semantic indexing

The results are shown in Table 1, where each table entry is a measure-of-goodness of the corresponding technique. We note that the improvements under LSI don't seem very large. This is an artifact of the small size and nature of our database and the fact that any of the techniques mentioned work well. It is, however, an indication that LSI is a technique worthy of further study in this environment.

Table 1. Results for Global Color Histogram and Color Anglogram Representations

	Global Color Histogram	**Color Anglogram**
Raw Data	0.9257	0.9508
Raw Data with LSI	0.9377	0.9556
Normalized and Weighted Data	0.9419	0.9272
Normalized and Weighted Data with LSI	0.9446	0.9284

Thus, for the global histogram approach, using normalized and weighted data or using latent semantic indexing with the raw data improves performance, while using both techniques is even better.

Our next approach uses sub-image matching in conjunction with color histograms. Each image is first converted from the RGB color space to the HSV color space. Each image is decomposed into 5 overlapping subimages, as shown in Figure 1. For the 50 images in our case, 250 subimages will be used in the following feature extraction process. For each pixel of the resulting image, hue and saturation are extracted and each quantized into a 10-bin histogram. Then the two histograms h and s are combined into one $h \times s$ histogram with 100 bins, which is the representing feature vector of each image. This is a vector of 100 elements, $\mathbf{V} = [f_1, f_2, f_3, \ldots f_{100}]^T$, where each element corresponds to one of the bins in the hue-saturation histogram.

We then generate the feature-subimage-matrix, $\mathbf{A} = [\mathbf{V}_1,\ldots,\mathbf{V}_{250}]$, which is 100×250. Each row corresponds to one of the elements in the feature vector and each column is the whole feature vector of the corresponding subimage. This matrix is written into a file so the computation is done only once. The matrix will be retrieved from the file during the query process.

A singular value decomposition is then performed on the feature-subimage-matrix. The result comprises three matrices, **U, S** and **V**, where $\mathbf{A} = \mathbf{USV}^T$. The dimensions of **U** are 100×100, **S** is 100×250, and **V** is 250×250. The rank of matrix **S**, and thus the rank of matrix **A**, in our case is 100. Therefore, the first 100 columns of **U** spans the column space of **A** and all the 100 rows in $\mathbf{V}^T$ spans the row space of **A. S** is a diagonal matrix of which the diagonal elements are the singular values of **A**. To

reduce the dimensionality of the transformed latent semantic space, we use a rank-*k* approximation, $\mathbf{A}_k$, of the matrix **A**, for k = 55. This is defined by $\mathbf{A_k} = \mathbf{U_k S_k V_k}^T$. The dimension of $\mathbf{A}_k$ is the same as **A**, 100 by 50. The dimensions of $\mathbf{U}_k$, $\mathbf{S}_k$, and $\mathbf{V}_k$ are 100×55, 55×55, and 250×55, respectively.

The first step of the query process in this approach is to compute the distance between the transformed feature vector of each subimage of the query image, **q**, and that of each of the 250 images in the database, **d**. This distance is defined as $dist(\mathbf{q}, \mathbf{d}) = \mathbf{q}^T\mathbf{d} / ||\mathbf{q}||\ ||\mathbf{d}||$, where $||\mathbf{q}||$ and $||\mathbf{d}||$ are the norms of those vectors. The computation of $||\mathbf{d}||$ for each of the 250 subimages is done only once and then written into a file.

With respect to the query image and each of the 50 database images, we now have the distances between each pair of subimages by the previous step. These distance values $dist(\mathbf{q}_i, \mathbf{d}_j)$ are then combined into one distance value between these two images in an approach similar to the computation of Euclidean distance. Given a query image **q**, with corresponding subimages $q_1, \ldots, q_5$, and a candidate database image **d**, with corresponding subimages $d_1, \ldots, d_5$, we define,

$$dist(q, d) = \frac{1}{5}\sqrt{\sum_{i=1}^{5}[dist(q_i, d_i)]^2}$$

This approach was again compared to one without using latent semantic analysis. Each image is decomposed into five subimages which are then represented by their hue-saturation histograms ***V***. Then the cosine measure between corresponding subimages is computed and used as the similarity metric. We thus have the distance between the query image and each of the 50 database images. These similarity values are then combined into one similarity measure between these two images. Given a query image **q** and a candidate image **d** in the database, we define,

$$dist(q, d) = \frac{1}{5}\sum_{i=1}^{5} sim(q_i, d_i)$$

Using each image as a query, we again find the average sum of the positions of all of the five correct answers. Now, without using latent semantic analysis, using the measure previously introduced, the result is 0.9452, while the use of latent semantic analysis brings this measure to 0.9502.

We also did a similar experiment where $dist(q, d)$ weighted the center subimage twice as much as the peripheral subimages. Using the same measure, the results of these experiments are 0.9475 for the experiment without using latent semantic analysis and 0.9505 for that using latent semantic analysis. Therefore, latent semantic indexing does improve the retrieval performance for both global and subimage color histogram based retrieval. Comparison of global and subimage results shows that subimage provides better performance than global color histogram either with or without latent semantic indexing.

4.2 The Effects of Latent Semantic Indexing, Normalization, and Weighting for Color Anglograms

Our next approach performs similar experiments utilizing our previously formulated approach of color anglograms [10]. This is a novel spatial color indexing scheme based on the point feature map obtained by dividing an image evenly into a number of *M***N* non-overlapping blocks with each individual block abstracted as a unique feature point labeled with its spatial location, dominant hue, and dominant saturation.

For our experiments, we divide the images into 8*8 blocks, have 10 quantized hue values and 10 quantized saturation values, count the two largest angles for each Delauney triangle, and have an anglogram bin of 5°. Our vector representation of an image thus has 720 elements: 36 hue bins for each of 10 hue ranges and 36 saturation bins for each of 10 saturation ranges. We use the same approach to querying as in the previous section.

We conducted similar experiments for these four cases:

1. Color anglograms, no normalization, no term weighting, no latent-semantic indexing (raw data)
2. Color anglograms, normalized and term-weighted, no latent semantic indexing
3. Color anglograms, no normalization, no term-weighting, with latent semantic indexing
4. Color anglograms, normalized and term-weighted, with latent semantic indexing

with the results shown in Table 1.

From these results, one notices that our anglogram method is better than the standard global color histogram, which is consistent with our previous results [10,11]. One also notices that latent semantic indexing improves the performance of this method. However, it seems that normalization and weighting has a negative impact on query performance. We more thoroughly examined the impact of these techniques and derived the data shown in Table 2.

Table 2. More Detailed Results for Color Anglogram Representation

	Color Anglogram
Raw Data with LSI	0.9556
Normalized Data with LSI	0.9476
Weighted Data with LSI	0.9529
Normalized and Weighted Data with LSI	0.9284

The impact of normalization is worse than that of weighting. Normalization is a compacting process which transforms the original feature image matrix (the anglogram elements) to the range [0, 1]. Now, the feature image matrix in this case is a sparse matrix with many 0's, some small integers, and a relatively small number of large integers. We believe that these large integers represent the discriminatory power of the anglogram and that the compacting effect of normalization weakens their significance. Local log-weighting also has a compacting effect. Since both the local and global weighting factors lie between 0 and 1, the transformed matrix always has

smaller values than the original one, even though no normalization is applied. Thus, normalization and weighting don't help improve the performance, but actually makes it worse.

4.3 Utilizing Image Annotations

We conducted various experiments to determine whether image annotations could improve the query results of our various techniques. The results indicate that they can.

For both the global color histogram and color anglogram representation, we appended an extra 15 elements to each of these vectors (called *category bits*) to accommodate the following 15 keywords associated with these images: *sky, sun, land, water, boat, grass, horse, rhino, bird, human, pyramid, column, tower, sphinx, and snow*. Thus, the feature vector for the global histogram representation now has 115 elements (100 visual elements and 15 textual elements), while the feature vector for the color anglogram representation now has 735 elements (720 visual elements and 15 textual elements). Each image is annotated with appropriate keywords and the area coverage of each of these keywords. For instance, one of the images is annotated with sky(0.55), sun(0.15), and water(0.30). This is a very simple model for incorporating annotation keywords. One of the strengths of the LSA technique is that it is a vector-based method that helps us to integrate easily different features into one feature vector and to treat them just as similar components. Hence, ostensibly, we can apply the normalization and weighting mechanisms introduced in the previous sections to the expanded feature image matrix without any concern.

For the global color histogram representation, we start with an image feature matrix of size 115×50. Then, using the SVD, we again compute the rank 34 approximation to this matrix, which is also 115×50. For each query image, we fill bits 101 through 115 with 0's. We also fill the last 15 rows of the transformed image feature matrix with all 0's. Thus, for the querying, *we do not use any annotation information*. We also note, that as before, we apply normalization and weighting, as this improves the results, which are shown in Table 3. The first two results are from Table 1, while the last result shows how our technique of incorporating annotation information improves the querying process.

Table 3. Global Color Histograms with Annotation Information

	Global Color Histogram
Normalized and Weighted Data	0.9419
Normalized and Weighted Data with LSI	0.9446
Normalized and Weighted Data with LSI and Annotation Information	0.9465

For the color anglogram representation, we start with an image feature matrix of size 735×50. Then, using the SVD, we again compute the rank 34 approximation to

this matrix, which is also 735 × 50. For each query image, we fill bits 721 through 735 with 0's. We also fill the last 15 rows of the transformed image feature matrix with all 0's. Thus, for the querying, *we do not use any annotation information*. We also note that as before, we do not apply normalization and weighting, as this improves the results, which are shown in Table 4. The first two results are from Table 1, while the last result shows how our technique of incorporating annotation information improves the querying process.

Table 4. Global Color Histograms with Annotation Information

	Color Anglogram
Raw Data	0.9508
Raw Data with LSI	0.9556
Raw Data with LSI and Annotation Information	0.9590

Note that annotations improve the query process for color anglograms, even though we do not normalize the various vector components, nor weight them. This is quite surprising, given that the feature image vector consists of 720 visual elements, which are relatively large integers, and only 15 annotation elements, which are in the range [0,1].

5 Conclusion and Future Work

In this chapter we proposed image retrieval schemes that incorporate multiple visual feature extraction represented by color histograms and color anglograms. Features are extracted on both whole image level and subimage level to better capture salient object descriptions. To negotiate the gap between low-level visual features and high-level concepts, latent semantic indexing is applied and integrated with these content-based retrieval techniques in a vector space model. Correlation between visual features and semantics are explored. Annotations are also fused into the feature vectors to improve the efficiency and effectiveness of the retrieval process.

The results presented in the previous section are quite interesting and are certainly worthy of further study. Our hope is that latent semantic analysis will find that different image features co-occur with similar annotation keywords, and consequently lead to improved techniques of semantic image retrieval. We are currently experimenting with the integration of shape anglograms, color anglograms, and structural features with latent semantic indexing and developing a unified framework to accommodate multiple features and their representation. We will further test and benchmark this integrated image retrieval framework over various large image databases, along with tuning the latent semantic indexing scheme to achieve optimal performance with highly reduced dimensionality. We will further our study of image semantics and incorporation of textual annotations and explore the correlation between visual feature groups and semantic clusters. We also consider applying various clustering techniques and use the cluster identifier in place of annotation

information. Analyzing the patterns of user interaction, either in the query process or in the browsing process, is another interesting research topic. Making use of relevance feedback to infer user preference should also be incorporated to elevate the retrieval performance. Finally, considering that the image archives on the internet are normally associated with other sources of information such as captions, titles, labels, and surrounding texts, we also propose to extend the application of the latent semantic indexing technique to analyze the structure of different types of visual and hypermedia documents.

References

1. A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, Content-Based Image Retrieval at the End of the Early Years, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 12, 2000.
2. R. Zhao and W. I. Grosky, From Features to Semantics: Some Preliminary Results, *Proceedings of the IEEE International Conference on Multimedia & Expo*, New York, New York, 2000.
3. M. Stricker and A. Dimai. Color Indexing with Weak Spatial Constraints. *Proceedings of SPIE Storage and Retrieval for Image and Video Databases*, Vol. 2670, 1996, pp. 29-40.
4. W. I. Grosky, F. Fotouhi, and Z. Jiang. Using Metadata for the Intelligent Browsing of Structured Media Objects, *Managing Multimedia Data: Using Metadata to Integrate and Apply Digital Data*, A. Sheth and W. Klas (Eds.), McGraw Hill Publishing Company, New York, 1998, pp. 67-92.
5. S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman, Indexing by Latent Semantic Analysis, *Journal of the American Society for Information Science*, Volume 41, Number 6 (1990), pp. 391-407.
6. M. Berry, Z. Drmac, and E. Jessup, Matrices, Vector Spaces, and Information Retrieval, *SIAM Review*, Vol. 41, No. 2, 1999, pp. 335-362.
7. S. Dumais, Improving the Retrieval of Information from External Sources, *Behavior Research Methods, Instruments, and Computers*, Vol. 23, Number 2 (1991), pp. 229-236.
8. G. H. Golub and C. Van Loan, *Matrix Computation*, Johns Hopkins Univ. Press, Baltimore, MD, 1996.
9. V. N. Gudivada and V. Raghavan. Design and Evaluation of Algorithms for Image Retrieval by Spatial Similarity. *ACM Transactions on Information Systems*, Vol. 13, No. 1 (April 1995), pp. 115-144.
10. Y. Tao and W. I. Grosky, Spatial Color Indexing Using Rotation, Translation, and Scale Invariant Anglograms, *Multimedia Tools and Applications*, To Appear.
11. Y. Tao and W. I. Grosky, Object-Based Image Retrieval Using Point Feature Maps, *Proceedings of The 8th IFIP 2.6 Working Conference on Database Semantics (DS8)*, Rotozur, New Zealand, January 5-8, 1999.

Inference in Rule-Based Systems by Interpolation and Extrapolation Revisited

Sándor Jenei*

Institute of Mathematics and Informatics, University of Pécs
Ifjúság u. 6, H–7624 Pécs, Hungary
jenei@ttk.pte.hu

Abstract. We deal with the problem of rule interpolation and rule extrapolation for fuzzy and possibilistic systems. Such systems are used for representing and processing vague linguistic **If-Then**-rules, and they have been increasingly applied in the field of control engineering, pattern recognition and expert systems. The methodology of rule interpolation is required for deducing plausible conclusions from sparse (incomplete) rule bases. For this purpose the well-known fuzzy inference mechanisms have to be extended or replaced by more general ones. The methods proposed so far in the literature for rule interpolation are mainly conceived for the application to fuzzy control and miss certain logical characteristics of an inference. This serves as a motivation for looking for a more flexible method that is superior to the proposed ones with respect to its general applicability to fuzzy as well as to possibilistic systems.
First, a set of axioms is proposed. With this, a definition is given for the notion of interpolation, extrapolation, linear interpolation and linear extrapolation of fuzzy rules. The axioms include all the conditions that have been of interest in the previous attempts and others which either have logical characteristics or try to capture the linearity of the interpolation. A new method for linear interpolation and extrapolation of compact fuzzy quantities of the real line is suggested and analyzed in the spirit of the given definition. The method is extended to non-linear interpolation and extrapolation. Finally, the method is extended to the general case, where the input space is n-dimensional, by using the concept of aggregation operators.

Keywords: Knowledge-based systems, Fuzzy sets, Sparse rule base, Inference, Interpolation/extrapolation of fuzzy rules, Approximate reasoning, Expert system, Fuzzy control

1 Summary of the Topic

Fuzzy set theory is a formal framework for modeling input-output relations, for which only vague/linguistic information is available to describe them. The fact

* Supported by the Fonds zur Förderung der wissenschaftlichen Forschung (FWF, Project P12900-TEC) and by the National Scientific Research Fund Hungary (OTKA F/032782).

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 53–57, 2001.

that especially complex systems can often be described in a more transparent and efficient way than by physical-mathematical models, has led to a broad spectrum of applications of fuzzy set based methods. Expert and data base systems, operations research, image processing and control engineering are examples, where fuzzy systems have successfully been applied.

Historically, fuzzy systems were constructed from linguistic **If-Then**-rules of a human expert. Recently, learning techniques have increasingly been developed and applied to the construction of fuzzy **If-Then**-rules from numerical sample data. The major advantages of fuzzy models are their modularity (each rule can be designed separately) and the semantics they rely on, by which numerical as well as subjective, linguistic information can be integrated within one system. For both methods of constructing rule bases it can happen that for special input parameters no rule is specified. In the case of learning techniques it may happen that the sample data do not sufficiently represent input parameters which only occur infrequently. In the case of asking a human expert, an incomplete rule base can be the consequence of missing experience for particular system configurations. Another reason for an incomplete rule base may be the following: Sometimes human experts are only able (or willing) to state explicit rules for prototypical system configurations. In this case rules for different configurations have to be derived from the prototypical ones by *analogue reasoning*. It should be pointed out that knowledge can be represented by stating only prototypical rules and applying analogue reasoning in an especially transparent and efficient way.

In any case rule interpolation may be considered as an "inference technique" for fuzzy rule bases for which the premises do not cover the whole input space. Of course, any inference mechanism has to satisfy certain logical criteria. It may be said that the methods proposed so far in the literature are mainly conceived for the application to fuzzy control and that they are not appropriate for applications where the logical aspects of approximate reasoning are intrinsically important. All this serves as a motivation for looking for a more flexible method in this paper which is superior to the proposed ones with respect to its general applicability to fuzzy as well as to possibilistic systems.

After an axiomatic treatment of linear (and non-linear) interpolation and extrapolation, we introduce a new method for linear interpolation of compact fuzzy quantities of $\mathbb{R}$. We extend it to extrapolation, to non-linear (e.g., quadratic) interpolation and extrapolation and a corresponding defuzzification method is proposed. Finally, we extend the method to the general n-dimensional input space case. The details can be found in [4] and [5]. Only the axiomatic basis is presented below:

1.1 Conditions on Rule Interpolation/Extrapolation

We postulate a set of axioms and a definition for interpolation, extrapolation, linear interpolation and linear extrapolation of fuzzy sets.

In the following let $\mathcal{X}$ denote an input space and $\mathcal{Y}$ an output space. Denote the set of *valid* fuzzy subsets of $\mathcal{X}$ and $\mathcal{Y}$ by $\mathcal{F}_{\mathcal{V}}(\mathcal{X})$ and $\mathcal{F}_{\mathcal{V}}(\mathcal{Y})$, respectively. Further, let us consider a rule-base $\mathcal{R}$ of m rules of the form

$$\textbf{If } X = A_i \textbf{ Then } Y = B_i$$

where $A_i \in \mathcal{F}(\mathcal{X})$ and $B_i \in \mathcal{F}(\mathcal{Y})$ for all $1 \leq i \leq m$. Formally a rule interpolation $\mathcal{I} : \mathcal{F}_{\mathcal{V}}(\mathcal{X}) \to \mathcal{F}_{\mathcal{V}}(\mathcal{Y})$ is a mapping, which assigns to an observation $A \in \mathcal{F}_{\mathcal{V}}(\mathcal{X})$ an interpolating (plausible) conclusion $\mathcal{I}(A) \in \mathcal{F}_{\mathcal{V}}(\mathcal{Y})$. If the rule interpolation $\mathcal{I}$ is expected to behave as an inference mechanism in the sense of approximate reasoning, at least the following conditions have to be satisfied:

[I0] Validity of the conclusion
The conclusion should be a fuzzy subset of the universe $\mathcal{Y}$ with a valid membership function. This means that "membership functions" like in Figure 1 are not allowed. We felt necessary to postulate this, since some of the existing methods *do not* satisfy even this very elementary condition. Usually, further conditions for validity are required too: For example, when a method is restricted to the use of e.g. trapezoidal membership functions, then the result may be expected to be trapezoidal too. The normality of the quantities is frequently supposed too.

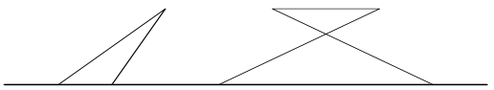


Fig. 1. Invalid "membership functions"

[I1] Compatibility with the rule-base
For all $i \in \{1, \ldots, m\}$ and all $A \in \mathcal{F}_{\mathcal{V}}(\mathcal{X})$ it follows from $A = A_i$ that $\mathcal{I}(A) = B_i$. This condition is the *modus ponens* in logic.

The following condition is essential for any logical inference.

[I2] Monotonicity condition
If $A^* \in \mathcal{F}_{\mathcal{V}}(\mathcal{X})$ is more specific than $A \in \mathcal{F}(\mathcal{X})$, then $\mathcal{I}(A^*)$ is more specific than $\mathcal{I}(A)$, i.e., for all $A, A^* \in \mathcal{F}_{\mathcal{V}}(\mathcal{X})$ the inequality $A^* \sqsubseteq A$ implies the inequality $\mathcal{I}(A^*) \sqsubseteq \mathcal{I}(A)$.

In addition to the basic properties **[I0]**, **[I1]** and **[I2]** also "smoothness" conditions on the mapping $\mathcal{I}$ are of interest, because in many applications "similar" observations are expected to induce "similar" conclusions. In order to make this property more precise, adequate concepts like "continuity" for mappings which map fuzzy subsets to fuzzy subsets, are required. Let $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ be metrics on $\mathcal{F}_{\mathcal{V}}(\mathcal{X})$ and $\mathcal{F}_{\mathcal{V}}(\mathcal{Y})$ respectively.

[I3] Continuity condition
For $\varepsilon > 0$ there exists $\delta > 0$ such that if $A, A^* \in \mathcal{F}_{\mathcal{V}}(\mathcal{X})$, and $d_{\mathcal{X}}(A, A^*) \leq \delta$ then for the corresponding conclusions we have $d_{\mathcal{Y}}(\mathcal{I}(A), \mathcal{I}(A^*)) \leq \varepsilon$.

If $\cup$ (resp. $\cap$) denotes union (resp. intersection) of fuzzy subsets (with appropriate subscripts if other operations than the pointwise maximum (resp. minimum) of the membership functions are considered) then it is natural to require the following two conditions. **[I4]** says that it is worth putting together all the available information first and drawing the conclusion afterwards rather than concluding from each piece of information separately and then putting together the conclusions. **[I5]** is just the dual of **[I4]**.

[I4] $\mathcal{I}(A \cap_{\mathcal{X}} A^*) \sqsubseteq \mathcal{I}(A) \cap_{\mathcal{Y}} \mathcal{I}(A^*)$, whenever $A \cap_{\mathcal{X}} A^*$ has valid membership function.

[I5] $\mathcal{I}(A \cup_{\mathcal{X}} A^*) \sqsupseteq \mathcal{I}(A) \cup_{\mathcal{Y}} \mathcal{I}(A^*)$ whenever $A \cup_{\mathcal{X}} A^*$ has valid membership function.

In the case of *linear* interpolation/extrapolation in a fully-ordered space it is widely accepted to choose *two* rules for the observation to be the basis of the interpolation/extrapolation. For a *linear* interpolation/extrapolation it is quite natural to postulate the following axioms: (For any fuzzy quantity A and real number c, denote by $A + c$ the "shifted" fuzzy quantity defined by $\mu_{A+c}(x) = \mu_A(x - c)$.)

[I6] Linearity principle
If $\mathcal{X} = \mathcal{Y} = \mathbb{R}$, $\mathcal{I}(A_1) = A_1 + c$ and $\mathcal{I}(A_2) = A_2 + c$ then for any observation A for which the antecedents of the basis of interpolation/extrapolation are A_1 and A_2 we should have $\mathcal{I}(A) = A + c$.

For $c = 0$ the linearity principle reduces to the

[I6*] Identity principle
If $\mathcal{X} = \mathcal{Y} = \mathbb{R}$, $\mathcal{I}(A_1) = A_1$ and $\mathcal{I}(A_2) = A_2$ then for any observation A for which the antecedents of the basis of interpolation/extrapolation are A_1 and A_2 we should have $\mathcal{I}(A) = A$.

Note that axioms **[I6]** and **[I6*]** are meaningful only if $\mathcal{X} = \mathcal{Y} = \mathbb{R}$.

In the case of *linear* interpolation (resp. extrapolation) in a fully-ordered space it is widely accepted to choose the basis of the interpolation (resp. extrapolation) in such a way that the observation lies "in between" the antecedents of the chosen rules (resp. the second antecedent lies "in between" the first antecedent and the observation). Then, of course, the corresponding conclusion is expected to lie "in between" the consequences of the antecedents (resp. the consequence of the second antecedent is expected to lie "in between" the consequence of the first antecedent and the conclusion). This leads to axiom

[I7] Preserving "in between"
If the observation A is in between A_i and A_j (resp. A_j is in between A and A_i) then $\mathcal{I}(A)$ should be in between $\mathcal{I}(A_i)$ and $\mathcal{I}(A_j)$ (resp. $\mathcal{I}(A_j)$ should be in between $\mathcal{I}(A)$ and $\mathcal{I}(A_i)$).

Since the choice of the relation "in between" may not be natural, this axiom is not included in the following definition.

Definition 1. We call a mapping $\mathcal{I} : \mathcal{F}_{\mathcal{V}}(\mathcal{X}) \to \mathcal{F}_{\mathcal{V}}(\mathcal{Y})$ *interpolation/extrapolation* if it satisfies axioms **[I0]**–**[I5]**, and in the case $\mathcal{X} = \mathcal{Y} = \mathbb{R}$ we call it *linear interpolation/extrapolation* if in addition it satisfies **[I6]** (and hence **[I6*]**). We call $\mathcal{I}$ $\cap$-*decomposable* (resp. $\cup$-*decomposable*) if axiom **[I4]** (resp. **[I5]**) holds with equality instead of subsethood. An interpolation/extrapolation which is both $\cap$-decomposable and $\cup$-decomposable is called *decomposable*.

References

1. D. Dubois, H. Prade and M. Grabisch, Gradual rules and the approximation of control laws, In: H.T. Nguyen, M. Sugeno, R. Tong and R. R. Yager (Eds.), *Theoretical Aspects of Fuzzy Control*, John Wiley and Sons, New York, 1995, 147–181.
2. S. Gottwald, *A Treatise on Many-Valued Logics.* Studies in Logic and Computation, vol. 9, Research Studies Press: Baldock, Hertfordshire, England, 2001.
3. P. Hájek, *Metamathematics of fuzzy logic*, Kluwer, Dordrecht, (1998).
4. S. Jenei, Interpolation and Extrapolation of Fuzzy Quantities Revisited — an Axiomatic Approach, *Soft Computing* (to appear)
5. S. Jenei, E.P. Klement, R. Konzel, Interpolation and extrapolation of fuzzy quantities — the multiple-dimensional case, *Soft Computing* (to appear).
6. L. A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, *IEEE SMC* **3** (1973) 28–44.

Recent Advances in Wavelength Routing*

Christos Kaklamanis

Computer Technology Institute and
Dept. of Computer Engineering and Informatics,
University of Patras, 26500 Rio, Greece.
kakl@cti.gr

Abstract. We study the problem of allocating optical bandwidth to sets of communication requests in all–optical networks that utilize Wavelength Division Multiplexing (WDM). WDM technology establishes communication between pairs of network nodes by establishing transmitter–receiver paths and assigning wavelengths to each path so that no two paths going through the same fiber link use the same wavelength. Optical bandwidth is the number of distinct wavelengths. Since state–of–the–art technology allows for a limited number of wavelengths, the engineering problem to be solved is to establish communication between pairs of nodes so that the total number of wavelengths used is minimized; this is known as the *wavelength routing problem*.
In this paper we survey recent advances in bandwidth allocation in tree–shaped WDM all–optical networks. We present hardness results and lower bounds for the general problem and the special case of symmetric communication. We also survey various techniques that have been developed recently, and explain how they can be used to attack the problem. First, we give the main ideas of deterministic greedy algorithms and study their limitations. Then, we show how to use various ways and models of wavelength conversion in order to achieve almost optimal bandwidth utilization. Finally, we show that randomization can help to improve the deterministic upper bounds.

1 Introduction

Optical fiber is rapidly becoming the standard transmission medium for backbone networks, since it can provide the required data rate, error rate and delay performance necessary for high speed networks of next generation[18,37]. However, data rates are limited in opto–electronic networks by the need to convert the optical signals on the fiber to electronic signals in order to process them at the network nodes. Although electronic parallel processing techniques are capable, in principle, to meet future high data rate requirements, the opto–electronic conversion is itself expensive. Thus, it appears likely that, as optical technology improves, simple optical processing will remove the need for opto–electronic conversion. Networks using optical transmission and maintaining optical data paths through the nodes are called *all–optical networks*.

* This work was partially funded by the European Union under IST FET Project ALCOM–FT and Improving RTN Project ARACNE.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 58–72, 2001.

Optical technology is not yet as mature as conventional technology. There are limits as to how sophisticated optical processing at each node can be done.

Multiwavelength communication [18,37] is the most popular communication technology used on optical networks. Roughly speaking, it allows to send different streams of data on different wavelengths along an optical fiber. Multiwavelength communication is implemented through *Wavelength Division Multiplexing* (WDM). WDM takes all data streams traveling on an incoming link and route each of them to the right outgoing link, provided that each data stream travels on the same wavelength on both links.

In a WDM all–optical network, once the data stream has been transmitted as light, it continues without conversion to electronic form until it reaches its destination. For a packet transmission to occur, a transmitter at the source must be tuned to the same wavelength as the receiver at the destination for the duration of the packet transmission and no data stream collision may occur at any fiber.

We model the underlying fiber network as a directed graph, where vertices are the nodes of the network and links are optical fibers connecting nodes. Communication requests are ordered pairs of nodes, which are to be thought of as transmitter–receiver pairs. WDM technology establishes connectivity by finding transmitter–receiver directed paths and assigning a wavelength (color) to each path, so that no two paths going through the same link use the same wavelength.

Optical bandwidth is the number of available wavelengths. Optical bandwidth is a scarce resource. State–of–the–art technology allows some hundreds wavelengths per fiber in the laboratory, even less in manufacturing, and there is no anticipation for dramatic progress in the near future. At the state of the art there is no WDM all–optical network that uses the optical bandwidth in an efficient way. However, for a realistic use of WDM all–optical networks for long distance communication networks, it seems necessary a significant progress in the protocols for allocation of the available bandwidth. Thus, the important engineering problem to be solved is to establish communication between pairs of nodes so that the total number of wavelengths used is minimized; this is known as the *wavelength routing problem* [1,30].

Given a pattern of communication requests and a corresponding path for each request, we define the *load* of the pattern as the maximum number of requests that traverse any fiber of the network. For tree networks, the load of a pattern of communication requests is well defined, since transmitter–receiver paths are unique. Clearly, for any pattern of requests, its load is a lower bound on the number of necessary wavelengths.

Theoretical work on optical networks mainly focuses on the performance of wavelength routing algorithms on regular networks using oblivious (predefined) routing schemes. We point out the pioneering work of Pankaj [30] who considered shuffle exchange, De Bruijn, and hypercubic networks. Aggarwal et al. [1] consider oblivious wavelength routing schemes for several networks. Raghavan and Upfal in [33] consider mesh–like networks. Aumann and Rabani [7] improve the bounds of Raghavan and Upfal for mesh networks and also give tight re-

sults for hypercubic networks. Rabani in [31] gives almost optimal results for the wavelength routing problem on meshes.

These topologies reflect architectures of optical computers rather than wide–area networks. For fundamental practical reasons, the telecommunication industry does not deploy massive regular architectures: backbone networks need to reflect irregularity of geography, non–uniform clustering of users and traffic, hierarchy of services, dynamic growth, etc. In this direction, Raghavan and Upfal [33], Aumann and Rabani [7], and Bermond et al. [10], among other results, focus on bounded–degree networks and give upper and lower bounds in terms of the network expansion.

However, wide–area multiwavelength technology is expected to grow arround the evolution of current networking principles and existing fiber networks. These are mainly SONET (Synchronous Optical Networking Technology) rings and trees [18,37]. In this sense, even asymptotic results for expander graphs do not address the above telecommunications scenario.

In this work we consider tree topologies, with each edge of the tree consisting of two opposite directed fiber links. Raghavan and Upfal [33] considered trees with single undirected fibers carrying undirected paths. However, it has since becomes apparent that optical amplifiers placed on fiber will be directed devices. Thus, directed graphs are essential to model state of the art technology.

In particular, we survey recent methods and algorithms for efficient use of bandwidth in trees considering arbitrary patterns of communication requests. All the results are given in terms of the load of the pattern of requests that have to be routed. Surveys on bandwidth allocation for more specific communication patterns like *broadcasting*, *gossiping*, and *permutation routing* can be found in [9,22].

The rest of this paper is structured as follows. In Section 2 we formalize the wavelength routing problem and present hardness results and lower bounds for the general case and the special case of symmetric communication. In Section 3 we describe deterministic greedy algorithms and present the best known results on them. In Section 4 we relax the model and introduce devices called *converters* which can improve bandwidth allocation with some sacrifice in network cost. In Section 5 we briefly describe the first randomized algorithm for the problem. We conclude in Section 6 with a list of open problems.

2 Hardness Results and Lower Bounds

As we mentioned in the previous section, the load of the communication pattern is a lower bound on the number of necessary colors (wavelengths). The following question now arises. Given any communication pattern of load L, can we hope for a wavelength routing with no more than L wavelengths? The answer is negative for two reasons. The former is that this problem is NP–hard. The latter is that there are patterns that require more than L wavelengths.

We formalize the problem as follows.

WAVELENGTH ROUTING IN TREES
INSTANCE: A directed tree T and a pattern of communication requests (i.e., a set of directed paths) P of load L.
QUESTION: Is it possible to assign wavelengths (colors) from $\{1, 2, ..., L\}$ to requests of P in such a way that requests that share the same directed link are assigned different wavelength?

Intuitively, we may think of the wavelengths as colors and the wavelength routing problem as a coloring problem of directed paths. In the rest of the paper we use the terms wavelength (wavelength routing) and color (coloring), interchangeably.

Erlebach and Jansen [14] have proved the following hardness result.

Theorem 1 ([14]). WAVELENGTH ROUTING IN TREES *is NP–complete.*

Note that the above statement is true even if we restrict instances to arbitrary trees and communication patterns of load 3. The following statement, which is due to Erlebach and Jansen [15] as well, applies to binary trees and communication patterns of arbitrary load.

Theorem 2 ([15]). WAVELENGTH ROUTING ON BINARY TREES *is NP–complete.*

Thus, the corresponding optimization problems (minimizing the number of wavelengths) are NP–hard, in general.

Now, we give the second reason why, given a communication pattern of load L, a wavelength routing with not much more than L wavelengths is infeasible.

Theorem 3 ([25]). *For any integer $l > 0$, there exists a communication pattern of load $L = 4l$ on a binary tree T that requires at least $5L/4$ wavelengths.*

Theorems 1 and 2 hold even in the special case of patterns of symmetric communication requests [11], i.e., for any transmitter–receiver pair of nodes (v_1, v_2) in the communication pattern, its symmetric pair (v_2, v_1) also belongs to the pattern. Furthermore, there exist patterns of symmetric communication requests of load L on binary trees which require arbitrarily close to $5L/4$ wavelengths [11]. The construction of these patterns is much more complicated than that of [25]. In the same work [11], interesting inherent differencies between the general problem and the symmetric case are also shown.

3 Greedy Algorithms

All known wavelength routing algorithms [29,20,25,21,19,13] belong to a special class of algorithms, the class of *greedy* algorithms. We devote this section to their study. Given a tree network T and a pattern of requests P, we call greedy a wavelength routing algorithm that works as follows:

> Starting from a node, the algorithm computes a breadth–first (BFS) numbering of the nodes of the tree. The algorithm proceeds in phases, one per each node u of the tree. The nodes are considered following their BFS numbering. The phase associated with node u assumes that we already have a proper coloring where all requests that touch (i.e. start, end, or go through) nodes with numbers strictly smaller than u's have been colored and no other request has been colored. During this phase, the partial proper coloring is extended to one that assigns proper colors to requests that touch node u but have not been colored yet. During each phase, the algorithm does not recolor requests that have been colored in previous phases.

Thus, various greedy algorithms differ among themselves in the strategies followed to extend the partial proper coloring during a phase. The algorithms in [29,20,25,21] make use of complicated subroutines in order to extend the partial coloring during a phase; in particular, their subroutines include a reduction of the problem to an edge coloring problem on a bipartite graph. On the other hand, the algorithms in [19,13] use much simpler methods to solve the wavelength routing problem on binary trees. The common characteristic for all these algorithms is that they are deterministic.

The algorithms in [29,20,25,21] reduces the coloring of a phase associated with node u to an edge coloring problem on a bipartite graph. In the following we describe this reduction.

Let v_0 be u's parent and let $v_1, \cdots, v_k$ be the children of u. The algorithm constructs the *bipartite graph associated with* u in the following way. For each node v_i, the bipartite graph has four vertices W_i, X_i, Y_i, Z_i and the left and right partitions are $\{W_i, Z_i | i = 0, \cdots k\}$ and $\{X_i, Y_i | i = 0, \cdots k\}$. For each request of the tree directed out of some v_i into some v_j, we have an edge in the bipartite graph from W_i to X_j. For each request directed out of some v_i and terminating on u, we have an edge from W_i to Y_i. Finally, for each request directed out of u into some v_i, we have an edge from Z_i to X_i. See Figure 1. The above edges are called *real*. Notice that all edges that are adjacent to either X_0 or W_0 have already been colored, as they correspond to requests touching a node with BFS number smaller than u's (in this specific case, the requests touch u's parent) that have been colored at some previous phase. We call the edges incident to either X_0 or W_0 *color–forced edges*.

Notice that no real edge extends across opposite vertices Z_i and Y_i or W_i and X_i. Indeed vertex Z_i has edges only to vertices of type X_i; on the other hand, an edge from W_i to X_i would correspond to a request in the tree going from c_i to itself. We call a pair of opposite vertices a *line*. Notice also that all vertices of the bipartite graph have degree at most L and, thus, it is possible to add *fictitious edges* to the bipartite graph so that all vertices have degree exactly L. The following claim holds.

Claim ([29]). Let P be a pattern of communication requests on a tree T. Consider a specific BFS numbering of the nodes of T, a node u and a partial coloring χ of the requests of P that touch nodes with BFS number smaller than u's. Then,

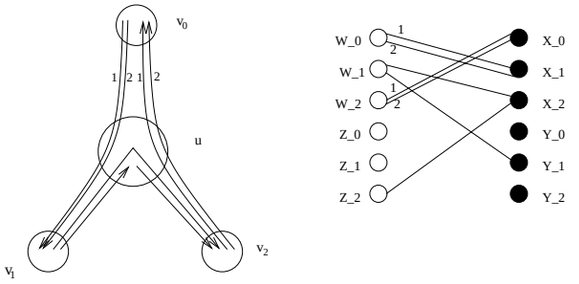


Fig. 1. Requests touching node u and the relative bipartite graph (only real edges are shown)

any coloring of the edges of the bipartite graph associated with u corresponds to a legal coloring of the requests of P that touch node u.

Thus, the problem of coloring requests is reduced to the problem of coloring the edges of an L–regular bipartite graph, under the constraints that some colors have already been assigned to edges adjacent to W_0 and X_0. We call this problem an *α–constrained bipartite edge coloring problem* on an L–regular bipartite graph. The parameter α denotes that edges incident to nodes W_0 and X_0 have been colored with αL colors. The objective is to extend the coloring to all the edges of the bipartite graph.

The works [29,20,25,21] give efficient solutions to this problem. They either consider matchings in pairs and color them in sophisticated ways using detailed potential and averaging arguments for the analysis [29,25] or partition matchings into groups which can be colored and accounted for indepedently [20,21]. In particular, Kaklamanis *et al.* [21] solve the problem proving the following theorem.

Theorem 4 ([21]). *For any $\alpha \in [1, 4/3]$ and integer $L > 0$, there exists a polynomial time algorithm for the α–constrained bipartite edge coloring problem on an L–regular bipartite graph which uses at most $\left(1 + \frac{\alpha}{2}\right) L$ total colors and at most $4L/3$ colors per line.*

The interested reader may refer to the papers [29,20,25,21] for detailed description of the techniques. Note that one might think bipartite edge coloring problems with different constraints. Tight bounds on the number of colors for more generalized constrained bipartite edge coloring problems can be found in [12].

Using as a subroutine the coloring algorithm presented in [21] for $\alpha = 4/3$, the wavelength routing algorithm at each phase maintains the following two invariants:

I. Each phase uses a total number of colors no greater than $5L/3$.
II. The number of colors seen by two opposite directed links is at most $4L/3$.

In this way the following statement can be proved.

Theorem 5 ([21], see also [16]). *There exists a polynomial time greedy algorithm which routes any pattern of communication requests of load L on a tree using at most $5L/3$ wavelengths.*

In the same work [21] (see also [16]), it was proved that, in general, greedy deterministic algorithms cannot route patterns of communication requests of load L in trees using less than $5L/3$ wavelengths. Thus, the greedy algorithm presented in [21,16] is best possible within the class of deterministic greedy algorithms. In Section 5 we demonstrate how randomization can be used to beat the barrier of 5/3 at least on binary trees.

4 Wavelength Conversion

In the previous section we saw that (deterministic) greedy wavelength routing algorithms in tree networks cannot use, in the worst case, less than $5L/3$ wavelengths to route sets of communication requests of load L, resulting to 60% utilization of available bandwidth. Furthermore, even if a better (non-greedy or randomized) algorithm is discovered, we know by Theorem 3 that there exist communication patterns of load L that require $5L/4$ wavelengths, meaning that 20% of the available bandwidth across the fiber links will remain unutilized.

The inefficiency of greedy algorithms in the allocation of the bandwidth is due to the fact that greedy algorithms color requests going through node u without "knowing" which requests go through its children.

Thus, if we are seeking better utilization of optical bandwidth we have to relax some of the constraints of the problem. In particular, *wavelength converters* allow to relax the restriction that a request has to use the same wavelength along the whole request from the transmitter to the receiver.

A possibility would be to convert the optical signal into electronic form and to retransmit it at a different wavelength. If there is no restriction on the wavelengths on which the message can be retransmitted, then it is possible to route all patterns of communication requests of load L with L wavelengths. In fact, in this case the assignment of wavelengths to requests on a link of the network is independent from the wavelengths assigned to the same requests on the other links.

However, converting optical signals to electronic signals has the drawback of wasting the benefits of using optical communication. Recently, a new technology has been proposed that allows to change the wavelength of an optical signal without converting it into electronic form. Wavelength converters have been designed and constructed [43]. A wavelength converter, placed at a node of the network, can be used to change the wavelengths assigned to requests traversing that node. The effects of wavelength conversion have been extensively studied in different models and it has been proved that it can dramatically improve the efficiency in the allocation of the optical bandwidth [8,24,34,35,36,39,42]. However, wavelength conversion is a very expensive technology and it is not

realistic to assume to have in all the nodes of the network the capability of changing wavelengths to all requests. This motivates the study of all-optical networks that allow for some form of restricted wavelength conversion.

In the literature, two prevalent approaches are used to address WDM optical networks with wavelength conversion: *sparse conversion* and *limited conversion*. In a network with sparse wavelength conversion, only a fraction of the nodes are equipped with wavelength converters that are able to perform an arbitrary number of simultaneous conversions; in a network with limited conversion, instead, each node hosts wavelength converters, but these devices can perform a limited number of conversions.

Sparse conversion optical networks have been considered in [35,39,41,23]. Wilfong and Winkler [41] consider the problem of minimizing the number of nodes of a network that support wavelength conversion in order to route any communication pattern using a number of wavelengths equal to the optimal load. They prove that the problem is NP–hard. Kleinberg and Kumar [23] present a 2–approximation to this problem for general networks, exploiting its relation to the problem of computing the minimum vertex cover of a graph. Ramaswami and Sasaki [35] show that, in a ring network, a simple converter is sufficient to guarantee that any pattern of requests of load L can be routed with L wavelengths. Subramaniam *et al.* [39] give heuristics to allocate wavelengths, based on probabilistic models of communication traffic.

Variants of the limited conversion model have been considered by Ramaswami and Sasaki [35], Yates *et al.* [42], and Lee and Li [26]. Ramaswami and Sasaki [35] propose ring and star networks with limited wavelength conversion to support communication patterns efficiently. Although they address the undirected case, all their results for rings translate to the directed case as well. Furthermore, they propose algorithms for bandwidth allocation in undirected stars, trees, and networks with arbitrary topologies for the case where the length of requests is at most two.

4.1 The Network Model

In our network model some nodes of the network host wavelength converters. A wavelength converter can be modeled as a bipartite graph $G = (X, Y, E)$. Each one of the sets of vertices X and Y have one vertex for each wavelength and there is an edge between vertices $x \in X$ and $y \in Y$ if and only if the converter is capable of converting the wavelength corresponding to x to the wavelength corresponding to y. For example, a full converter corresponds to a complete bipartite graph and a fixed conversion converter corresponds to a bipartite graph where the vertices of U have degree one. Some examples of wavelength converters are depicted in Figure 2.

In order to increase network performance without tremendous increase in cost, we mainly use converters of limited functionality. The term "limited" reflects the fact that the converters are simple according to two measures: their *degree* and their *size*. We proceed, now, to define these two measures.

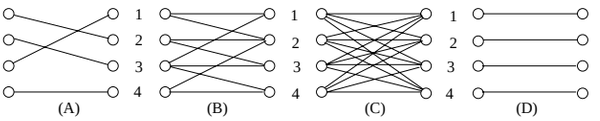


Fig. 2. Wavelength conversion bipartite graphs: (A) fixed conversion, (B) partial conversion, (C) full conversion, and (D) no conversion

Definition 6 *The degree of a converter is the maximum degree of the vertices of its corresponding bipartite graph.*

Definition 7 *The size of a converter is the number of edges in the corresponding bipartite graph.*

Let u be a node that hosts converters: each converter located at u is assigned to a pair of incoming and outgoing directed links adjacent to u. This converter will be used only to change colors assigned to requests containing the two directed links, while traversing u. Thus, a request may have one color on the incoming link and a different color on the outgoing link. In other words, the connection request corresponding to the request may travel on a wavelength on the segment ending at u and on a different wavelength on the segment starting from u.

In the discussion that follows, we consider three models of limited conversion networks, differing in the number of converters placed at nodes and in the placement pattern. Denote by d the degree of u and by p the parent of u. The models we consider are the following:

all-pairs There is one converter for each pair of incoming and outgoing links adjacent to u. Thus, the number of converters at u is $d(d-1)/2$ and we can change color to all requests traversing u.

top-down For each child v of u, there is a converter between links (p, u) and (u, v) and another converter between links (v, u) and (u, p). The number of converters at u is $2(d-1)$ and we can change color only to requests coming from or going to p.

down For each child v of u, there is a converter between links (p, u) and (u, v) and a converter between links (w, u) and (u, v), for each child w of u different from v. The number of converters at u is $(d-1)^2$ and we can change color only to requests going from p to a descendant of u or to requests traversing two distinct children of u.

In Figure 3 it is shown how converters are positioned at a node of a limited conversion binary tree.

Clearly, a wavelength routing algorithm for down and top–down limited conversion networks also work for all–pairs limited conversion network. However, the converse does not necessarily hold.

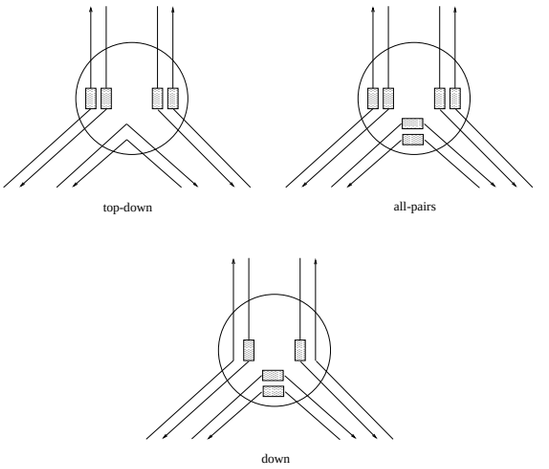


Fig. 3. The position of wavelength converters at a node of a limited conversion binary tree network

4.2 Recent Results

Recently, in a series of papers [4,5,6,17] (see also [2]), significant progress has been made in bandwidth allocation in sparse and limited conversion trees. In the following, we briefly discuss these results.

Auletta et al. [4] prove that converter in at least $\rfloor \frac{n}{4} - \frac{1}{2} \lfloor$ of the n nodes of a tree may be required if we seek for optimal bandwidth utilization. They furthermore demonstrate how to locate full converters in at most $\rfloor \frac{n}{4} - \frac{1}{2} \lfloor$ nodes so that optimal bandwidth utilization is always feasible.

In the case of limited conversion, Auletta et al. [4] show how to construct converters of degree $2\sqrt{L} - 1$ which allow optimal bandwidth utilization in top–down binary trees, i.e., all sets of requests of load L can be routed with L wavelengths. In [6], using properties of Ramanujan graphs and dispersers (classes of graphs which have been explicitly constructed in [27,28,40]), they construct converters of almost linear size which also allow optimal bandwidth utilization in top–down binary trees.

Auletta et al. [6] and independently Gargano [17] show how to achieve optimal bandwidth utilization in all–pairs conversion binary trees using converters of degree 15. The construction of these converters is based on properties of Ramanujan graphs. Furthermore, Gargano in [17] show how to achieve nearly–optimal bandwidth utilization in down–conversion arbitrary trees with low–degree converters. The construction of converters is based on properties of Ramanujan graphs as well. The interested reader may see [2] for formal proofs of these results.

5 Randomized Algorithms

In an attempt to beat the 5/3 lower bound for deterministic greedy algorithms, Auletta et al. [3] define the class of randomized greedy wavelength routing algorithms. Randomized greedy algorithms have the same structure as deterministic ones, i.e., starting from a node, they consider the nodes of the tree in a BFS manner. Their main difference is that a randomized greedy algorithm $\mathcal{A}$ uses a palette of colors and at each phase associated with a node, $\mathcal{A}$ picks a random proper coloring of the uncolored requests using colors of the palette according to some probability distribution.

The results presented in the following were originally obtained in [3]. The interested reader may see [3] for further details.

As far as lower bounds are concerned, Auletta et al. [3] prove that with very high probability, routing requests of load L with less than $3L/2$ colors in trees of depth $\Omega(L)$ and routing requests of load L with less than $1.293L - o(L)$ colors in trees of constant depth is infeasible using greedy algorithms. These statements are proved by considering randomized adversaries for the construction of the pattern of communication requests.

In the following we give the main ideas of a randomized wavelength routing algorithm presented in [3]. The algorithm has a greedy structure but allows for limited recoloring at the phases associated with each node.

At each phase, the wavelength routing algorithm maintains the following two invariants:

I. Each phase uses a total number of colors no greater than $7L/5$.
II. The number of colors seen by two opposite directed links is exactly $6L/5$.

At a phase associated with a node u, a *coloring procedure* is executed which extends the coloring of requests that touch u and its parent node to the requests that touch u and are still uncolored. The coloring procedure is randomized (selects the coloring of requests being uncolored according to a specific probability distribution). In this way, the algorithm can complete the coloring at the phase associated with node u using at most $7L/5$ colors in total, keeping the number of colors seen by the opposite directed links between u and its children to $6L/5$.

At each phase associated with a node u, the algorithm is enhanced by a *recoloring procedure* which recolors a small subset of requests in order to maintain some specific properties on the (probability distribution of the) coloring of requests touching u and its parent. This procedure is randomized as well.

The recoloring procedure at each phase of the algorithm works will very high probability. The coloring procedure at each phase always works correctly maintaining the two invariants. As a result, if the depth of the tree is not very large (no more than $O(L^{1/3})$), the algorithm executes the phases associated with all nodes, with high probability.

After the execution of all phases, the set of requests being recolored by the executions of the recoloring procedure are colored using the simple deterministic greedy algorithm with at most $o(L)$ extra colors due to the fact that as far as

the depth of the tree is not very large (no more than $O(L^{1/3})$), the load of the set of requests being recolored is at most $o(L)$, with high probability.

In this way, the following theorem is proved. The interested reader may look at [3] for a detailed description of the algorithm and formal proofs.

Theorem 8 ([3]). *Let $0 < \delta < 1/3$ be a constant. There exists a randomized wavelength routing algorithm that routes any pattern of communication requests of load L on a binary tree of depth at most $L^{\delta}/8$ using at most $7L/5 + o(L)$ colors, with probability at least $1 - \exp\left(-\Omega(L^{\delta})\right)$.*

6 Open Problems

Recent work on wavelength routing in trees has revealed many open problems; some of them are listed below.

- The main open problem is to close the gap between $5L/4$ and $5L/3$ for the number of wavelengths sufficient for routing communication patterns of load L on arbitrary trees (see Theorems 3 and 5). Closing the gap between $5L/4$ and $7L/5 + o(L)$ for binary trees also reserves some attention.
- Furthermore, although for deterministic greedy algorithms we know tight bounds on the number of wavelengths, this is not true for randomized greedy algorithms. Exploring the power of randomized greedy algorithms in more depth is interesting as well.
- Many improvements can be made in the work on wavelength conversion. The main open problem here is whether we can achieve optimal bandwidth utilization in arbitrary trees using converters of constant degree (or even linear size). We conjecture that this may be feasible in all–pairs conversion trees with deterministic greedy algorithms.

References

1. A. Aggarwal, A. Bar-Noy, D. Coppersmith, R. Ramaswami, B. Schieber, and M. Sudan, Efficient Routing and Scheduling Algorithms for Optical Networks, *Journal of the ACM*, Vol. 43, No. 6, 1996, pp. 973–1001.
2. V. Auletta, I. Caragiannis, L. Gargano, C. Kaklamanis, P. Persiano. Sparse and Limited Wavelength Conversion in All–Optical Tree Networks. *Theoretical Computer Science*, to appear.
3. V. Auletta, I. Caragiannis, C. Kaklamanis, and P. Persiano, Randomized Path Coloring on Binary Trees. In *Proc. of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2000)*, LNCS 1913, Springer, pp. 60–71, 2000.
4. V. Auletta, I. Caragiannis, C. Kaklamanis, and P. Persiano, Bandwidth Allocation Algorithms on tree–Shaped All-Optical Networks with Wavelength Converters. In *Proc. of the 4th International Colloquium on Structural Information and Communication Complexity (SIROCCO '97)*, 1997, pp. 13–27.

5. V. Auletta, I. Caragiannis, C. Kaklamanis, and P. Persiano, Efficient Wavelength Routing in Trees with Low–Degree Converters. *Multichannel Optical Networks: Theory and Practice*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 46, AMS, pp. 1–13, 1998.
6. V. Auletta, I. Caragiannis, C. Kaklamanis, and P. Persiano, On the Complexity of Wavelength Converters. In *Proc. of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS '98)*, LNCS 1450, Springer, 1998, pp. 771–779.
7. Y. Aumann and Y. Rabani, Improved Bounds for All Optical Routing. In *Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '95)*, 1995, pp. 567–576.
8. R. A. Barry, and P. A. Humblet, Models of Blocking Probability in All–Optical Networks with and without Wavelength Changers. IEEE Journal on Selected Areas of Communication, vol. 14, 1996, pp. 858–867.
9. B. Beauquier, J.-C. Bermond, L. Gargano, P. Hell, S. Perennes, and U. Vaccaro, Graph Problems Arising from Wavelength–Routing in All–Optical Networks. *2nd Workshop on Optics and Computer Science (WOCS '97)*, 1997.
10. J.–C. Bermond, L. Gargano, S. Perennes, A.A. Rescigno, and U. Vaccaro, Efficient Collective Communication in Optical Networks. *Theoretical Compute Science*, 233 (1-2), 2000, pp. 165-189.
11. I. Caragiannis, C. Kaklamanis, and P. Persiano. Wavelength Routing of Symmetric Communication Requests in Directed Fiber Trees. In *Proc. of the 5th International Colloquium on Structural Information and Communication Complexity (SIROCCO 98)*, pp. 10–19, 1998.
12. I. Caragiannis, C. Kaklamanis, and P. Persiano. Edge Coloring of Bipartite Graphs with Constraints. In *Proc. of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS 99)*, LNCS 1450, Springer, pp. 771–779, 1999. Also, *Theoretical Computer Science*, to appear.
13. I. Caragiannis, C. Kaklamanis, P. Persiano. Bounds on Optical Bandwidth Allocation in Directed Fiber Tree Topologies. *2nd Workshop on Optics and Computer Science*, 1997.
14. T. Erlebach and K. Jansen. Scheduling of Virtual Connections in Fast Networks. In *Proc. of 4th Workshop on Parallel Systems and Algorithms (PASA '96)*, World Scientific, 1997, pp. 13–32.
15. T. Erlebach, K. Jansen. Call Scheduling in Trees, Rings and Meshes. In *Proc. of the 30th Hawaii International Conference on System Sciences*, 1997.
16. T. Erlebach, K. Jansen, C. Kaklamanis, M. Mihail and P. Persiano Optimal Wavelength Routing on Directed Fiber Trees *Theoretical Computer Science* 221, 1999, pp. 119–137.
17. L. Gargano, Limited Wavelength Conversion in All–Optical Tree Networks. In *Proc. of the 25th International Colloquium on Automata, Languages, and Programming (ICALP '98)*, LNCS 1443, Springer, pp. 544–555.
18. P. E. Green, Fiber-Optic Communication Networks. *Prentice–Hall*, 1992.
19. K. Jansen. Approximation Results for Wavelength Routing in Directed Binary Trees. *2nd Workshop on Optics and Computer Science*, 1997.
20. C. Kaklamanis and P. Persiano, Efficient Wavelength Rouiting in Directed Fiber Trees. In *Proc. of the 4th European Symposium on Algorithms (ESA '96)*, LNCS 1136, Springer, pp. 460–470.
21. C. Kaklamanis, P. Persiano, T. Erlebach, K. Jansen Constrained Bipartite Edge Coloring with Applications to Wavelength Routing. In *Proc. of the 24th Inter-*

national Colloquium on Automata, Languages, and Programming (ICALP '97), LNCS 1256, Springer, pp. 493–504, 1997.
22. R. Klasing, Methods and Problems of Wavelength–Routing in All–Optical Networks. In *Proc. of MFCS 98 Workshop on Communication*, pp. 1–9, 1998.
23. J. Kleinberg and A. Kumar, Wavelength Conversion in Optical Networks. In *Proc. of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '99)*, pp. 566–575, 1999.
24. M. Kovacevic and S. Acampora, Benefits of wavelength Translation in All–Optical Clear–Channel Networks. IEEE Journal on Selected Areas of Communication, vol. 14, 1996, pp. 868–880.
25. E. Kumar and E. Schwabe, Improved Access to Optical Bandwidth in Trees. In *Proc. of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, 1997, pp. 437–444.
26. K. -C. Lee and V.O.K. Li, Routing and Switching in a wavelength convertible lightwave network. In *Proc. of IEEE INFOCOM 1993*, 1993, pp. 578–585.
27. A. Lubotzky, R. Phillips, and P. Sarnak, Ramanujan Graphs. *Combinatorica*, vol. 8, 1988, pp. 261–278.
28. G.A. Margulis, Explicit Group–Theoretic Constructions of Combinatorial Schemes and their Applications for the Construction of Expanders and Concentrators. *Problemy Peredaci Informacii*, 1988.
29. M. Mihail, K. Kaklamanis, and S. Rao, Efficient Access to Optical Bandwidth. In *Proc. of 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS '95)*, 1995, pp. 548–557.
30. R. Pankaj, Architectures for Linear Lightwave Networks. PhD. Thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge MA, 1992.
31. Y. Rabani, Path Coloring on the Mesh, In *Proc. of the 37th IEEE Symposium on Foundations of Computer Science (FOCS 96)*, pp. 400–409, 1996.
32. J. Radhakrishnan and A. Ta-Shma, Tight Bounds for Depth-Two Super concentrators. In *Proc. of the 38th Annual IEEE Symposium on Foundations of Computer Science*, 1997, pp. 585–594.
33. P. Raghavan and E. Upfal, Efficient Routing in All–Optical Networks. In *Proc. of the 26th Annual ACM Symposium on the Theory of Computing (STOC '94)*, 1994, pp. 133–143.
34. B. Ramamurthy and B. Mukherjee, Wavelength Conversion in WDM Networking. IEEE J. on Selected Areas of Communication, vol. 16, 1998, pp. 1061–1073.
35. R. Ramaswami and G.H. Sasaki, Multiwavelength Optical Networks with Limited Wavelength Conversion. In *ACM/IEEE Trans. Networking*, vol. 6, n. 6, 1998, pp. 744–754.
36. R. Ramaswami and K. Sivarajan, Routing and wavelength Assignment in All–Optical Networks. In *ACM/IEEE Trans. Networking*, vol. 3, n. 5, 1995, pp. 489–500.
37. R. Ramaswami, K. Sivarajan, Optical Networks. *Morgan Kauffman Publishers*, 1998.
38. M. Sipser, Expanders, Randomness, or Time versus Space. *Journal of Computer and System Sciences*, vol. 36, n. 3, 1988, pp. 379–383.
39. S. Subramaniam, M. Azizoglu, and A.K. Somani, All–Optical Networks with Sparse wavelength Conversion. *ACM/IEEE Trans. on Networking*, vol. 4, n. 4, 1996, pp. 544–557.
40. A. Ta–Shma, Almost Optimal Dispersers. In *Proc. of the 30th Annual ACM Symposium on the Theory of Computing (STOC '98)*, 1998, pp. 196–202.

41. G. Wilfong and P. Winkler, Ring Routing and Wavelength Translation. In *Proc. of the 9th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA '98)*, 1998, pp. 333–341.
42. J. Yates, J. Lacey, D. Everitt, and M. Summerfield, Limited Range Wavelength Translation in All–Optical Networks. In *Proc. of IEEE INFOCOM '96*, 1996, pp. 954-961.
43. J. Zhou, N. Park, K. Vahala, M. Newkirk, and B. Miller, Four–Wave Mixing Wavelength Conversion Efficiency in Semiconductor Travelling–Wave Amplifiers Measured to 65 nm of Wavelength Shift. *IEEE Phot. Tech. Letters*, vol. 6, n. 8, 1994, pp. 984–987.

From Metacomputing to Grid Computing: Evolution or Revolution?

Domenico Laforenza

Advanced Computing Department
CNUCE-Institute of the Italian National Research Council
via Vittorio Alfieri, 1, I-56010 Ghezzano, Pisa, Italy
Domenico.Laforenza@cnuce.cnr.it

Abstract. The last decade have seen a considerable increase in computer and network performance, mainly as a result of faster hardware and more sophisticated software. In fact, the need for realistic simulations of complex systems relevant to the modeling of several modern technologies and environmental phenomena increasingly stimulates the development of new advanced computing approaches.

Today it is possible to couple a wide variety of resources including supercomputers, storage systems, data sources, and special classes of devices distributed geographically, and use them as a single unified resource, thus forming what is popularly known as a computational grid. The initial aim of Grid Computing activities was to link supercomputing sites (Metacomputing); current objectives go far beyond this. According to Larry Smarr, NCSA Director, a Grid is a seamless, integrated computational and collaborative environment. Many applications can benefit from the grid infrastructure, including collaborative engineering, data exploration, high throughput computing, and of course distributed supercomputing [1, 2, 3].

Grid applications (multi-disciplinary applications) couple resources that cannot be replicated at a single site even or may be globally located for other practical reasons. These are some of the driving forces behind the inception of grids. In this light, grids let users solve larger or new problems by pooling together resources that could not be coupled easily before.

Hence the Grid is not only a computing paradigm for just providing computational resources for grand-challenge applications. It is an infrastructure that can bond and unify globally remote and diverse resources ranging from meteorological sensors to data vaults, from parallel supercomputers to personal digital organizers.

Currently, there are many grid projects underway worldwide. A very comprehensive listing can be found in [4][5]. Moreover, two important international open forums, Grid (www.gridforum.org), and E-Grid (www.egrid.org) have been created in order to promote and develop Grid Computing technologies and applications. This talk aims to present the state-of-the-art of grid computing and attempts to survey the major international adventures in this area.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 73–74, 2001.

References

1. Smarr L., *Infrastructure for Science Portals*, IEEE Internet Computing, January/February 2000, 71-73.
2. Foster I. and Kesselman C. (editors), *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann Publishers, USA, 1999.
3. Leinberger W., Kumar V., *Information Power Grid: The new frontier in parallel computing?*, IEEE Concurrency, October-December 1999, 75-84.
4. Gentzsch W. (editor), *Special Issue on Metacomputing: From Workstation Clusters to Internet computing*, Future Generation Computer Systems, No. 15, North-Holland, 1999.
5. Baker M., Buyya R., and Laforenza D., *The Grid: International Efforts in Global Computing*, in the Proceedings of SSGRR 2000 Computer & eBusiness Conference, Scuola Superiore G. Reiss Romoli, L' Aquila, July 31 – August 6, 2000.

Knowledge-Based Control Systems

Simon Lambert

Business and Information Technology Department,
CLRC Rutherford Appleton Laboratory,
Chilton, Didcot, Oxon. OX11 0QX, UK
S.C.Lambert@rl.ac.uk
http://www.cclrc.ac.uk/

Abstract. The monitoring and control of devices and systems has attracted attention from the field of knowledge-based systems for a number of reasons. The problem is intrinsically knowledge-intensive, benefiting from awareness of the behaviour and interactions of components. There are complexities such as time constraints, partial and qualitative information, and in many cases there is a need for a degree of human understandability in performance. In this paper the problems of control are characterised, and a selection of applicable knowledge-based techniques described. Some architectural issues for control system design are also discussed. Two case studies are described which illustrate the variety of problems and approaches: one for control of flooding in a city, the other for control of an anaerobic waste treatment plant.

1 Introduction

The word "control", whether used in the context of information technology or more generally, implies the existence of a desired state of the world from which the actual state of the world may differ. It furthermore implies the possibility of taking actions which will result in adjustments to the state of the world, with a view to bringing the actual state closer into line with the desired. From this starting point a number of the problems of control systems are already visible on the horizon, and likewise the importance of knowledge in coping with them.

Let us start thinking about control systems in terms of the underlying system that they control. Such a system implements a process, which may be more or less well understood, and the process is characterised by inputs and outputs. Furthermore, it operates in an environment which may influence its behaviour. Already there are complexities emerging in this view of the situation: for example, feedback which blurs the distinction between inputs and outputs, or distinguishing between inputs to the process and important environmental factors which influence it.

Examples of such systems include:

- The currency of a nation, where a key output is the exchange rate with respect to other currencies, one of the inputs is the interest rate set by the national bank, and the environment includes the economic state and

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 75–89, 2001.

prospects of the economy, the beliefs and expectations of investors, and a multitude of other factors.
- The central heating of an office building, where the output is the desired temperature in the offices, the input is the operation of the boiler as determined by the setting of the thermostat, and the environment includes the exterior temperature.
- The flood defences of a city, where the outputs are the levels of water in rivers, retention basins, and (in the worst case) city streets, the inputs are the rainfall and the states of the flood defences, and the environment includes the level of the tide in the river.

It is clear from considering these examples and many others that there is a distinction between outputs at the highest level, such as "amount of flooding", and what is directly measurable and available to the controlling system. These measurables will include levels of water at particular points in the river or drainage system, and the state of functioning of apparatus. There is likewise a distinction between the conceptual inputs, such as "rainfall", and the direct controllables, including operation of pumps and sluice gates. Taking these ideas into consideration, we can now formulate our view of a control system diagramatically, as shown in Fig. 1. We side-step the problems of representing inputs, outputs and environment by thinking in terms of controllables and measurables—those entities that may be directly set and directly evaluated by the controlling system.

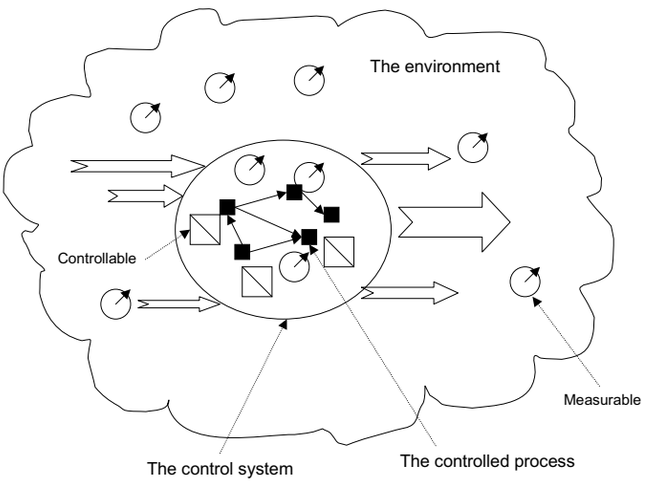


Fig. 1. A view of control

A number of important points are worth mentioning before we proceed to consider the place of knowledge in control systems.

- The entity that performs the control may be a human being, a computer system, or a combination of both. The diagram says nothing about where the responsibility lies for the actions taken on the controllables. Though it is possible that the system operates without intervention or supervision by a human, this is only one end of a spectrum of possibilities. In other cases, the "locus of responsibility" [21] might lie more with the human.
- It is possible that the system being controlled might not be the physical system of ultimate interest, but rather a model of it in the information system. In this case, the control is taking place on the model, which is being used to improve control of the real system—the "management flight simulator" idea ([17], [12]).
- A question arises about the relationship between control and diagnosis. Diagnosis too is concerned with differences between ideal states and actual states. But in diagnosis, the measurables are used to deduce the malfunctioning of the system, and controllables to vary behaviour so as to shed more light on the malfunction. Diagnosis is not the subject of this paper; nonetheless, some knowledge-based techniques such as model-based reasoning do have applications in control.

2 Knowledge and the Problems of Control

The representation we have shown of a general control system allows us to identify some of the problems of control, and to see why knowledge-based techniques can help to alleviate them.

First and most obvious is the problem of incomplete information. The measurables to which the controller has access may only give a very partial view of the situation, perhaps because of physical constraints on their number and location, perhaps because they measure only a subset of quantities of interest. In the case of anaerobic waste treatment plants, to which we shall return later, the simplest and most robust sensors measure elementary properties such as temperature and pH. Clearly there are many more properties of the plant which are valuable for monitoring its state.

Related to this point, but at a deeper level, are the problems arising from a lack of understanding of how the controlled system functions. The controller has access to measurements and can make adjustments to certain controllable factors, but how does he, she or it know the consequences of those actions? How can it know that adjusting thermostats or flow rates will result in changes to behaviour in the desired direction? Now it should be said that in some simple control situations there is no need for any understanding of the system's functioning—see [15] for an explanation of simple linear feedback control and why this works in many cases even when the underlying process is quite complex and not understood. But for complex systems with many dependencies, knowledge of the functioning at some level can be exploited to improve control—see [5] for the use of models to improve diagnostic capabilities.

To the problems of incomplete information and ill-understood processes we may add the problems of diverse information. In managing many macroscopic

systems such as waste water treatment plants, the operators' visual impressions may contain highly valuable information. Not all of our "measurables" are numerical and read by a mechanical, chemical or electrical sensor. If the operator can see black sludge packets at the surface of the clarifier, this suggests that the sludge is staying too long in the clarifier. How then can observations of this kind be integrated with numerical measurements in the control process?

A further problem in some domains arises from the timescales of the processes involved. A common analogy for this is "steering a super-tanker". We imagine trying to direct an enormous oil tanker as it approaches the oil terminal. Changes to the rudder will take effect very slowly, and continue inexorably. The worst way to manage such a system is by continually making changes based on the perceived behaviour at any instant: this will lead to over-correction and instability.

Another time-based problem is that of adaptation. It may be that the behaviour and properties of the controlled system are changing over time, and that therefore the controller should adapt its behaviour accordingly. What worked at one point in time may be unsuitable later. Of course, one has to tread very carefully here: the change in behaviour may be due to a malfunction in the system, and the controller will find itself attempting to correct for this when the best response would in fact be to flag a problem. However, there certainly are cases when processes change their characteristics in a normal way over time.

Finally, we return to the issue highlighted in the previous section of the locus of responsibility for the control. If a human is to have meaningful responsibility, they must be provided with the information they need to evaluate decisions and judgements. A cooperating human-computer system needs to take this into account. It is no use to propose a single course of action for the human to accept or reject unless they are also provided with a justification of why that they can critique. There are also other models for such an interaction: the human might make a suggestion which the computer system then critiques [19].

Fig. 2 shows our representation of a control system with some of these issues identified.

3 Knowledge-Based Techniques

Before considering the architecture of a control system, we will examine some particular knowledge-based techniques that may be used to implement elements of the control system. These techniques are knowledge-based in the sense that they manipulate explicitly represented knowledge of the domain, the system to be controlled, even the process of control itself. There should be some component that can be identified as "where the knowledge resides". For this reason, systems such as neural networks are excluded from the scope of this paper, since the representations they utilise are not explicit embodiments of knowledge [9].

For each of the techniques, their use in knowledge-based control will be explained, and their advantages summarised.

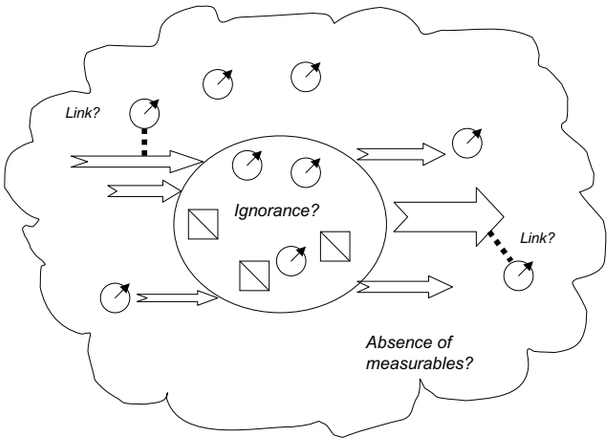


Fig. 2. The problems of control

3.1 Rule-Based Reasoning

Systems based on production rules have a long and successful history within the knowledge-based systems community [10]. The principle is very familiar: rule sets containing rules of the form `IF <condition> THEN <action>` are applied to particular situations to draw inferences which may then cause further rules to fire eventually leading to conclusions. This is elementary forward chaining; backward chaining is when hypotheses are tested by evaluating the rules that conclude those hypotheses, testing their premises successively until the hypothesis is confirmed or eliminated. In the context of control systems, rule-based reasoning may be applied to making the connection between measurables and controllables, as shown schematically in Fig. 3.

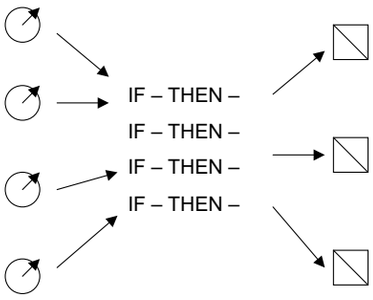


Fig. 3. Rule-based reasoning in control

Rule-based reasoning is frequently employed as one stage of knowledge-based control systems. Such reasoning is often semi-heuristic, i.e. it expresses knowledge that is founded in a model of the world though without an explicit underlying model in the knowledge base. A typical example is found in [7], a rule for the detection of acidification in an anaerobic waste water treatment plant.

```
IF pH evolution is low
AND methane evolution is low
AND there is no feeding overload
AND there is no change in feeding
THEN acidification pre-alarm state and generate report of fact
```

In this system the rules are implemented in the language OPS-83.

Rule-based reasoning may also be valuable in other components of a control system. It may be used to implement a component for proposing actions to the user—what has been called an Option Generator [14]. One approach is to use heuristic rules obtained from human experts about the range of actions they consider in particular situations to construct a sequence of actions, which may then be simulated by a model to ascertain the consequences before being executed in reality.

Two questions which must be asked when considering any choice of knowledge-based technique are: where does the knowledge come from? And how may it be validated? The answer to the first can take three forms: the knowledge comes direct from human experts; from textbooks and operations manuals; or is induced from historical data, using techniques of rule induction or data mining. The first two have the advantage that explanations based on it will tend to make sense to human controllers, while the latter may give better coverage in practice but may also be fragile in new situations.

Rule-based reasoning can be effective for addressing the following problems of control.

- Lack of understanding of the controlled system may not be required if heuristic rules are used, derived either from human experts or past history.
- Diverse information can be handled in the pre-conditions of rules—the above example rule illustrates this.
- Justification for humans can be provided by rule trace, which gives some degree of explanation (but see [3] for more advanced approaches to explanation).

3.2 Case-Based Reasoning

Case-based reasoning (CBR) is based on the principle that at least some human reasoning is based on recognition of similar cases to previous experience, with modification to new situations [16]. The basic CBR method entails the following steps:

- Matching the new situation to the library of prior cases, identifying the closest matches.

– Retrieving the solutions corresponding to the matches.
– Combining and modifying the solutions to produce a solution to the new case.
– Storing the new case and its solution (when validated) in the case base.

Issues in case-based reasoning include indexing, matching, retrieval, retaining and generalising cases.

CBR can be applied to control problems in the recognition of similar situations and hence reuse of "what worked before" (Fig. 4).

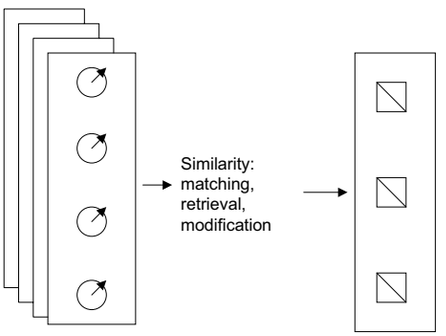


Fig. 4. Case-based reasoning in control

The control problems addressed by CBR include:

– Incomplete information, since a complete description is not necessarily needed to perform matching.
– Lack of understanding of the controlled system, since the method is based on recognition of previous cases rather than deep reasoning.
– Justification, which may be expressed with reference to prior cases.

3.3 Model-Based Reasoning

In this context a model is an explicit knowledge-based representation of some aspects of the controlled system, allowing reasoning to take place about its behaviour under varying conditions. A model may be structural (representing the physical components of the system) or functional (representing their inputs and outputs). Typically a representation will specify physical or logical components with relationships between them. For example, a water supply network might be represented as a set of demand areas (with associated demand profiles), storage reservoirs (with certain capacities), pipes, valves, etc. The reasoning is based on simple supply and demand matching. The reasoning may in practice be implemented by rules – so the two approaches are not exclusive. A context for the use of models is given in [4].

Applied to control problems, if it is possible to construct a model of the controlled system then it is possible to simulate behaviours and assess their consequences. This permits "what if" reasoning for either human or control system (Fig. 5).

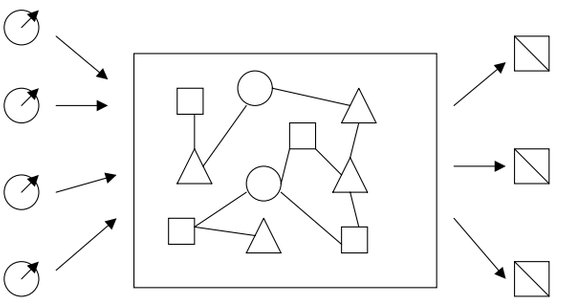


Fig. 5. Model-based reasoning in control

The problems addressed by model-based reasoning include:

- Timescales, since a suitable model will be able to indicate the rates of evolution of behaviour.
- Adaptation over time, since the model may be used to determine limits on correct functioning and measured values.
- Justification, based on explanation of the interactions between components.

3.4 Combining the Techniques

Most real problems of control will need the application of a variety of techniques for greatest effectiveness. As we have seen, individual techniques address different aspects of the problems of control. The issue for the system designer is one of selecting and combining techniques to address the problems of a particular case. This combination may be done at two levels: the reasoning level and the architectural level. The architectural level will be the subject of the next section; here we take a case study showing how different reasoning techniques may be applied to a single problem.

The TAP-EXTRA project [13] was a Trial Application under the European ESPRIT programme in which an emerging software technology, namely the enhancement of information systems with cooperative, explanatory capabilities, was applied in a new area to show the benefits and allow the risks to be judged. The specific application, called Aleph, was for assisting in flood control in the city of Bordeaux, and the co-operative, explanatory capabilities were added using the tools and methods developed within the previous ESPRIT project I-SEE.

The underlying problem is a control problem in which the locus of responsibility lies with the human operators. The RAMSES control centre in Bordeaux

is manned permanently and its staff constantly monitor incoming sensor readings (e.g. from rain gauges) and alarm signals (e.g. of a pump that has failed to start as expected). In times of heavy rainfall, their task is complicated by the likelihood of information overload. The fact that catastrophic flood events are rare makes matters more difficult for two reasons: equipment that is seldom used is more likely to malfunction, and the controllers' own experience of such situations is by definition limited. Furthermore, when a crisis threatens, an on-call engineer is called out who has the ultimate responsibility for the actions taken, but lacks the intimate familiarity with the flood control system.

A basic alarm filtering and combining application called Aleph had been developed and installed in the control centre, but was receiving only limited use. The reason was that the messages presented on the screen, although a synthesized subset of the basic incoming alarm signals, did not provide an adequate basis for decision making: they lacked context. Hence the basic control problem was one of incomplete knowledge of the situation and the underlying processes. The human controllers had many years of experience working in RAMSES, yet in situations of crisis they needed more support from the information systems.

Three specific user needs were identified:

- Risk assessment, particular in winter when the long duration of rains means that it is a difficult decision to determine when action must be taken.
- Identification of "mental model mis-matches", that is, situations which deviate from the normal and might go unnoticed, for example, a pump which continues to run when there is no longer any need for it.
- A synoptic overview of the situation, particularly for the on-call engineer.

The approach taken to satisfy these needs was a combination of rule-based with model-based reasoning, using the model to amplify and interpret the rule-based reasoning. For example, the failure of a pump to start is a single alarm message. If this message is repeated over a short period of time, then the pump can be considered to have failed and should be reported to the user—the job of Aleph. Then model-based reasoning about the layout and interconnections of the flood control network of pipes, pumps and retention basins is used to produce a risk assessment. The risk assessment is based on the following structure:

- The suggestive risk factor ("stimulus")
- The risk "content", i.e. what is there a risk of?
- Contributing factors
- Any potentially mitigating circumstances which might exist
- An overall assessment of the risk and the effect of the other factors. The characterization of the risk might include a timescale if this is appropriate.

Heuristic rules are used to detect the suggestive risk factors and the likely risk content, then the static model of the flood control network is queried to determine other relevant factors, for example, whether the water level in connecting basins is already high. Such factors allow for an informed and context-sensitive presentation of the risk to the user, giving much more helpful information for decision making than a simple statement that a certain pump has failed.

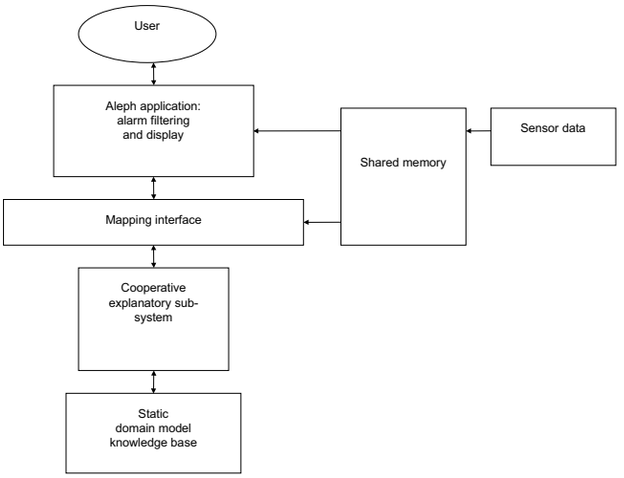


Fig. 6. The TAP-EXTRA system architecture

The overall architecture of the TAP-EXTRA system is shown in Fig. 6.

From the point of view of control systems, what we have here is an environment in which the crucial requirement is the effective presentation of relevant and timely information to the human controller to enable superior decision making. The information system does not in itself cause actions to be taken, or even propose actions to the user; nevertheless the combined human-computer system is acting as a control system to prevent flooding.

4 Architectural Issues

As well as the knowledge integration perspective, there is also a system design perspective: how the control system is put together from software components, possibly distributed. The motivation for developing a control system in such a way may arise if there are to be multiple instances of the controlled system, which are generically the same but vary in individual detail. For example, different versions of a waste treatment plant, installed in different locations and possessing different configurations and operating conditions.

In such cases, it can make sense to have repositories of knowledge physically distinct: human controllers and experts in different places, with a general knowledge base at central location and local knowledge bases associated with the individual installations.

4.1 Internet-Based Systems

It is clear that the Internet offers opportunities for developing such distributed knowledge-based control systems. The prospects for expert systems on the Internet have been identified and studied [8]: the simplest model is the Web-based

system in which the knowledge base is located and reasoning is performed on the central server, and the users access the system through Web browsers. Such an approach has the advantages of making the system highly accessible, using familiar interfaces, and being portable. However there are other more sophisticated models possible, including intelligent agents and cooperating expert systems.

Alternatively, a distinction may be drawn between "knowledge server" approaches (as just outlined) and "information client" [2]. The latter differs from the former in that the client is more active in seeking the information that it needs to solve the problem at hand, drawing on distributed knowledge sources according to what they can provide.

The FirstAID system for diagnosis of scanning electron microscopes [2] is an example of such a system. The motivation for the approach is that there are few experts, they are geographically distant, and the visual aspects are important for diagnosis. The basic protocols used are HTTP and CUSeeMe, the latter for for transmitting dynamic images. The diagnostic system uses rule-based reasoning with extensions, such as the ability to encode actions to be taken on the instrument as part of the rule consequence.

For Internet-based control systems of any type, a new consideration enters: that of information security. This means not just protecting the information from unauthorised parties, but also guarantees of integrity and provision for network failures (e.g. in the middle of a sequence of actions being transmitted).

4.2 Agent Architectures

Intelligent agents [11] have attracted a great deal of attention recently in many areas of application. Characteristics of agents include autonomy, independence, specialisation and cooperative negotiation. Each agent is responsible for its own area of problem solving, applying specialist knowledge. An example from the control of waste water treatment plants (WWTPs) is given in [1]. Here agents embody knowledge of every sub-process within the WWTP, for example the settler, the pumping system, COD removal. The agents are responsible for validation of data, detection of abnormal conditions, prediction of future values, etc. There is a supervisor agent responsible for controlling the behaviour of the domain agents and activating their communication. Communication is by means of a "minimal language", in fact tables of values (numerical and symbolic).

It is also worth mentioning blackboard architectures: once a popular architecture for knowledge-based systems [6], now less fashionable but similar in some respects to the agent-based approach. Diverse knowledge sources cooperate to build up a solution to a problem using a central, globally accessible "blackboard" as a common working area. Such an approach is promising for control applications because of the integration of different kinds of knowledge and data. An example application is given in [18].

4.3 The TELEMAC Project

The TELEMAC project is a project under the European Commission's Information Society Technologies (IST) programme, commencing in September 2001 [20]. It is concerned with the management of anaerobic digestion in waste treatment, specifically the wastes resulting from wine and spirit production processes. This is an unstable biological process. The TELEMAC project aims to develop a set of adaptive and customisable tools for small units in order to improve the quality of their depollution process, reduce the treatment cost and increase the output of derivative products. It emphasizes the synergy expected from the merger of robust advanced control algorithms and knowledge-based supervision systems.

The process to be controlled, anaerobic digestion, exemplifies most of the problems of control identified in Section 2, and suffers from a lack of tools to take benefit of its full potential. The process is based on a complex ecosystem of anaerobic bacterial species that degrade the organic matter. It presents very interesting advantages compared to the traditional aerobic treatment: it has a high capacity to degrade difficult substrates at high concentrations, produces very little sludge, requires little energy and in some cases it can even recover energy using methane combustion (cogeneration).

But in spite of these advantages, industrial companies are reluctant to use anaerobic treatment plants, probably because of the drawback of its efficiency: it can become unstable under some circumstances (like variations of the process operating conditions). A disturbance can lead to a destabilisation of the process due to accumulation of intermediate toxic compounds resulting in biomass elimination and several months are necessary for the reactor to recover. During this period, no treatment can be performed by the unit. It is therefore a great challenge for computer and control sciences to make this process more reliable and usable at industrial scale.

The technical objective of TELEMAC is to provide a set of tools to improve the process reliability and quality of managing a wastewater treatment plant with a remote centre of expertise using Internet resources. The advanced control system must ensure the optimal working of the process and the supervision system must set an alarm in case of failure. If the problem is simple enough, the supervision system must trigger dedicated automatic control algorithms that will help the system to recover. If not tackled automatically, the system must decide which human intervention is required, local technician (e.g. for pump failure, leak, ...) or remote expert. All these incidents and the measures taken must increment the database to feed back to the supervision system. The TELEMAC solution must therefore provide an integrated procedure that will increase the reliability of the process in order to guarantee that the plants respect environmental regulations.

From the point of view of knowledge-based control, there are a number of innovations in TELEMAC.

- The reliability of low information content sensors (the "measurables") is enhanced by linking together operational low information level sensors through

analytical models. This will allow testing of their global coherency and deriving highly informative estimates of some process variables through software sensors (observers). This sensor network will be perfected using the concept of aspecific sensors that are based on a robust methodology.
- The inclusion of fault detection and diagnosis techniques within the data analysis system, instead of a supervision system, will enhance the quality of the smart sensors' outputs. The fact that the measured value is matched with a characterisation of its uncertainty is another innovative feature of the proposed smart sensor. The supervision system will then decide more easily how to optimise the use of the received data. If they are of high quality, very accurate decisions can be made, whereas in the other case, more prudent decisions will be adopted.
- In order to build an efficient control and monitoring procedure, particularly in case of failure, models will be developed especially for abnormal working conditions. The first step will be to identify the most frequent failures (pump failures, inhibition of the biomass), then to reproduce them during lab experiments and finally to develop models for these situations.
- The fault detection and isolation (FDI) module must be able to detect if the process is working in a faulty environment, and determine the problem's origin. If a failure is detected, the model corresponding to the symptoms of the process will be chosen from the model base developed specially for faulty situations. Then, the software sensors and the control algorithms based on the selected model will be activated. The supervision system must not only test the integrity of the process, it also has to verify that the selected algorithms (controllers, software sensors, fault detection, ...) do their job properly. For that, it must be able to check the coherence of the algorithms' outputs with their theoretical properties (convergence rate, dynamical behaviour, ...). If they turn out to be inefficient, an alarm will be triggered.
- Significant information will be extracted from events occurring on the plant. Data from the sensor network, faults, controller outputs, simulations, expert consultations are not regarded as single and isolated events. They are combined and joined together by the supervision system to enrich the knowledge base.

Fig. 7 shows the TELEMAC approach in comparison with current approaches.

TELEMAC is a contribution to advancing the state of the art of knowledge-based control systems, most particularly in its use of a distributed, Internet-based architecture to optimise the distribution of monitoring and problem-solving capabilities between human and computer and between local and remote sites. It confronts many of the problems of knowledge-based control identified in Section 2.

5 Conclusion

Knowledge-based control is a broad field which encompasses a wide range of application domains and problem characteristics. The challenges that arise are

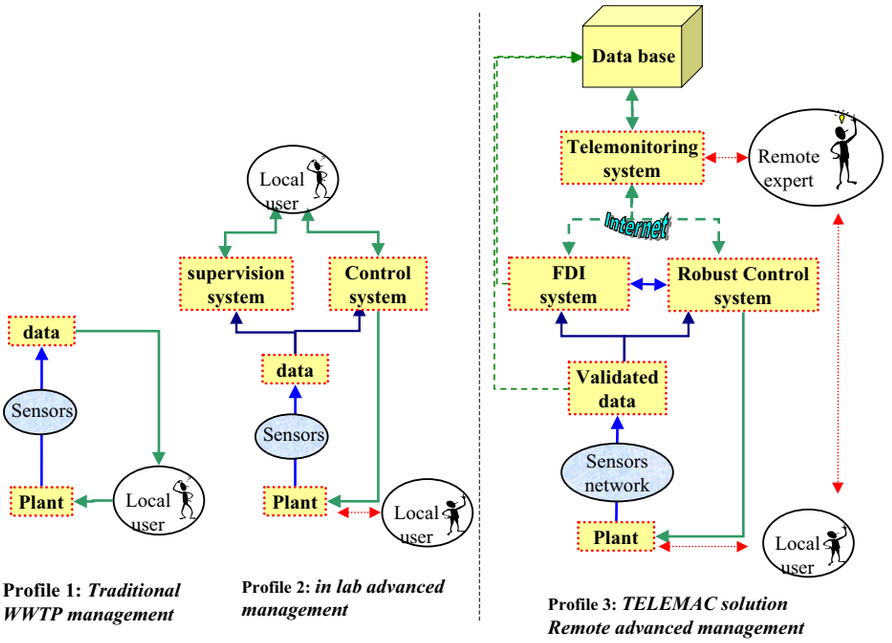


Fig. 7. TELEMAC compared with current approaches

partly due to the complexity of the control problem itself, partly due to the inherent difficulties in codifying and deploying knowledge in information systems. Various knowledge-based techniques are valuable and have been applied, and the current trend is towards integrating different techniques and distributing problem solving to take advantage of generalisation and customisation in knowledge.

References

1. Baeza, J., Gabriel, D., Lafuente, J.: An Agent Architecture for Supervising a Pilot WWTP. Environmental Decision Support Systems and Artificial Intelligence: Papers from the AAAI Workshop. AAAI Press, Menlo Park, CA (1999)
2. Caldwell, N.H.M., Breton, B.C., Holburn, D.C.: Remote Instrument Diagnosis on the Internet. IEEE Intelligent Systems **13** (1998) 70–76
3. Cawsey, A.: Explanation and Interaction. The Computer Generation of Explanatory Dialogues. MIT Press, Cambridge MA (1992)
4. Cortés, U., Sanchèz-Marrè, M., R-Roda, M., Poch, M.: Towards Environmental Decision Support Systems. Environmental Decision Support Systems and Artificial Intelligence: Papers from the AAAI Workshop. AAAI Press, Menlo Park, CA (1999)
5. De Kleer, J., Williams, B.C.: Diagnosing Multiple Faults. Artificial Intelligence **32** (1987) 97–130

6. Engelmore, R., Morgan, T. (eds.): Blackboard Systems. Addison-Wesley (1988)
7. Flores, J., Arcay, B., Dafonte, J.C.: Knowledge-Based System for Telecontrol of Anaerobic Waste Water Treatment Plants. Expert Systems **17** (2000) 71–80
8. Grove, R.: Internet-Based Expert Systems. Expert Systems **17** (2000) 129–135
9. Haykin, S.S.: Neural Networks. A Comprehensive Foundation. 2nd edition. Prentice Hall (1999)
10. Jackson, P.: Introduction to expert systems. 3rd edition. Addison Wesley (1998)
11. Jennings, N., Wooldridge, M.J. (eds.): Agent Technology. Foundations, Applications and Markets. Springer Verlag (1997)
12. Lambert, S.C.: Management Flight Simulators: An Analysis and A New Approach. Unpublished working paper available from the author (1999)
13. Lambert, S.C., Ringland, G.A., Bailly, C. Sacareau, P.: TAP-EXTRA Project Final Report. Project Deliverable CCLRC/TAP-EXTRA/DEV 27/09-97 (1997)
14. Lambert, S.C., Chappel, H.R. Ringland, G.A.: The Benefits of Knowledge Reuse in Knowledge-Based Systems: A Case Study. Proceedings of Expert Systems 92, Cambridge, England (1992)
15. Leigh, J.R.: Control Theory. A Guided Tour. IEE Control Series 45. Peter Peregrinus, London (1992)
16. Lenz, M. (ed.): Case-Based Reasoning Technology. From Foundations to Applications. Springer, Berlin (1998)
17. Morecroft, J., Sterman, J.: Modelling for Learning Organizations. Productivity Press, Portland OR (1994)
18. Russomano, D.J.: A Function-Centred Framework for Reasoning about System Failures at Multiple Levels of Abstraction. Expert Systems **16** (1999) 148–169
19. Silverman, B.G.: Critiquing Human Error. A Knowledge-Based Human-Computer Collaboration Approach. Academic Press (1992)
20. TELEMAC Description of Work. IST Project Number IST-2000-28156 (2001)
21. Whitby, B.: Artificial Intelligence: A Handbook of Professionalism. Ellis Horwood (1988)

Beyond the Turing Limit: Evolving Interactive Systems *

Jan van Leeuwen[1] and Jiří Wiedermann[2]

[1] Institute of Information and Computing Sciences, Utrecht University, Padualaan 14, 3584 CH Utrecht, the Netherlands
jan@cs.uu.nl
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. Modern networked computing systems follow scenarios that differ from those modeled by classical Turing machines. For example, their architecture and functionality may change over time as components enter or disappear. Also, as a rule their components interact with each other and with the environment at unpredictable times and in unpredictable manners, and they evolve in ways that are not pre-programmed. Finally, although the life span of the individual components may be finite, the life span of the systems as a whole is practically unlimited. The examples range from families of cognitive automata to (models of) the Internet and to communities of intelligent communicating agents.
We present several models for describing the computational behaviour of evolving interactive systems, in order to characterize their computational power and efficiency. The analysis leads to new models of computation, including 'interactive' Turing machines (ITM's) with advice and new, natural characterizations of non-uniform complexity classes. We will argue that ITM's with advice can serve as an adequate reference model for capturing the essence of computations by evolving interactive systems, showing that 'in theory' the latter are provably more powerful than classical systems.

1 Introduction

In the twentieth century, computability theory explored the limits of what can be digitally computed. The prominent claim, known as the Church-Turing thesis, asserts that every *algorithm* can be captured in terms of a standard Turing machine. The classical computing scenario consists of a fixed program, a finite input supplied in either off-line or online mode, and a meaningful result only if the computation halts in finite time. No changes to the program of the machine or its 'architecture' are allowed in the meantime, intermediate results cannot

* This research was partially supported by GA ČR grant No. 201/00/1489 and by EC Contract IST-1999-14186 (Project ALCOM-FT).

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 90–109, 2001.

influence the input and no information is carried over to future runs of the machine. This even applies in the case of ω-Turing machines.

Does this correspond to the way modern networked computers operate? Clearly it does not. Today's systems operate practically uninterruptedly since the moment of their installation. They obtain their inputs via many different channels at unpredictable times, deliver corresponding responses continuously at times when they are ready, accumulate information over the course of their entire existence and use it across the boundaries of separate 'runs'. Also, parts of their underlying hardware and software are updated, whenever an 'external agent' that operates some component decides to do so, without loss of vital data.

One can object, as many people do when confronted with this observation, that this use of computers is simply different from the way assumed in the Church-Turing thesis, and that this change is insignificant and can be easily accommodated by adjusting the original model. While the latter is true, the former is questionable: is it really an insignificant change? Answering this question will be a main concern in this paper. It will appear that at least in theory, the traditional notion of algorithmic computation must be extended for it.

Compared to the classical computing scenario, the essence of the changes in modern computing technology we have in mind can be subsumed under three complementary issues: *interactivity*, *non-uniform evolution* (and adaptivity), and *infinity of operation*. Interactivity is often also called 'reactivity' [3]. The systems performing according to these three qualities together constitute what is meant by *evolving interactive computing*.

The most prominent example is the Internet. It can be seen as a wide-area computing infrastructure, a kind of global computer ([4]). As a 'computer', the Internet is hindered by diverse administrative, architectural, and physical constraints. Worse even, it is 'undesigned', evolving unpredictably, with unpredictable computing characteristics. Many people, especially in the software engineering community (cf. [4], [16]), noticed that we are facing a new computing phenomenon that does not fit the classical Turing machine paradigm. For instance, Cardelli [4] writes:

> *'In order to program a global computer we first need to understand its model of computation. For example, does computation on the Web correspond naturally to a traditional model? There are indications that it does not. [For example] when browsing, we actively observe the reliability and bandwidth of certain connections (including zero or time-varying bandwidth), and we take action on these dynamic quality-of-service observables. These observables are not part of traditional models of computation, and are not handled by traditional languages. What models of computation and programming constructs can we develop to automate behavior based on such observables?'*

In this paper we will concentrate only on the first part of Cardelli's question, calling for models of computation that capture the essence of global computing. It will lead us to the concepts of non-uniform computation and to a new approach to several non-uniform complexity classes.

Wegner [16,17] went even further, by claiming that interactivity alone can lead to computations that are more powerful than computations by Turing machines. In other words, 'interactive computing' would violate the Church-Turing thesis. We will argue that interactivity alone is not sufficient to break the Turing barrier. Interactivity merely extends the objects that one computes on: from finite strings to infinite ones, and the feedback mechanism remains computable. The computing power of a system could go beyond that of classical computing if *non-uniformity* is considered. Non-uniformity enters when one allows 'evolving' changes of the underlying hardware and/or software in the course of uninterrupted computing. This is a standard case with the Internet.

Surprisingly, evolving interactive computing seems to pervade not only the current, highly networked information processing systems but also – and mainly so – the information processing in (societies of) living organisms. It is a 'technology' that has been invented by nature long ago that has very much the same characteristics as we described. Note that evolution in our setting is fundamentally different from the notion of learning that is often mentioned in connection with interactivity. Learning is usually understood as a software evolution, whereas we will also and especially consider hardware evolution.

The structure of the paper is as follows. In Section 2 we introduce an elementary, and therefore fundamental, tool for dealing with interactive systems: the interactive finite automaton. In Section 3 we introduce sequences of interactive finite automata that share global states, leading us to the model of evolving interactive systems that we have in mind. Next, in Section 4 we describe the basic interactive Turing machine (ITM) that will serve as a platform for the design of its non-uniform variants. In Section 5 we define the ITM with advice, in Section 6 the so-called site machine that models a site in a computer network, and finally in Section 7 we present the web Turing machine – a model of the Internet. Then, in Section 8 we prove the computational equivalence of the non-uniform models. In Section 9 we investigate the efficiency of web Turing machine computations in more detail, and show that these machines belong to the most efficient computational devices known in complexity theory. Finally, in Section 10 we will discuss some interesting issues related to our results.

Most of the results mentioned here can be found in more detail in the original papers [12,14,13,15,20]. The present paper primarily outlines the overall research framework. The notion of sequences of interactive finite automata with global states and the respective results are new.

2 Interactive Finite Automata

Under the classical scenario, finite automata are used for recognizing finite strings. Under the interactive scenario, we consider *interactive finite automata* (IFA) which are a generalization of Mealy automata. They process potentially infinite strings (called streams) of input symbols and produce a potentially infinite stream of output symbols, symbol after symbol. To stress the interactiveness, we assume that there is no input tape: the automaton reads the input stream via

a single input port. Likewise, it produces the output via a single output port. There is no way to return to symbols once read except when they're stored internally. We assume throughout that the input and output symbols are taken from the alphabet $\Sigma = \{0, 1, \lambda\}$. Symbol λ at a port means that 'presently, there is neither 0 nor 1 appearing at this port'. The steps of an IFA follow a finite, Mealy-type transition function.

Any IFA realizes a translation ϕ that transforms infinite input streams over Σ into similar output streams. The λ's are not suppressed in the translation. Clearly, instead of an IFA one could consider any other device that is capable of entering into only a finite number of different configurations, such as discrete neural (cf. [8]) or neuroidal (cf. [10]) nets, neuromata [9], combinatorial circuits (cf. [2]), and so on. From [2,19] the next theorem follows:

Theorem 1. *For translations $\phi : \Sigma^\omega \to \Sigma^\omega$ the following are equivalent:*

(a) ϕ is realized by a interactive finite (Mealy) automaton.
(b) ϕ is realized by a neuroid.
(c) ϕ is realized by a discrete neural net.
(d) ϕ is realized by a discrete neuroidal net.
(e) ϕ is realized by a combinatorial circuit.

In the theorem, all respective devices are assumed to work in an interactive mode in processing infinite input streams. Devices such as neural nets or combinatorial circuits that read their input in parallel, process the input stream in blocks that correspond to the number of their input ports.

IFA's embody two features of evolving interactive computing: interactivity, and infinity of operation. The interactivity enables one to describe (albeit *a posteriori*) the interaction between the machine and its environment: inputs succeeding to some outputs may be reactions to these outputs. Note that we did not yet impose the third desideratum of evolving interactive computing: the evolvability of the underlying computing mechanism. Of course, due to their simplicity, IFA's do not have universal computing power either. Note that the motivation and computational scenario of IFA's differ from those usually considered for ω-automata (cf. [12] or [13]).

3 Sequences of Interactive Finite Automata with Global States

In order to achieve universal computing power and support the evolvability property, we consider *sequences* of IFA's. This enables us to realize more complicated translations and will also reveal the dependence of computational efficiency on the size of the underlying devices. The approach is inspired by the similar practice in non-uniform complexity theory where e.g. sequences (or families) of combinatorial circuits are considered (cf. [2,6]). Our approach differs not only in the use of different 'building blocks' – namely IFA's instead of combinatorial circuits,

but also in the use of a communication mechanism between neighboring members in a sequence. Both changes are motivated by the need to accommodate all ingredients of evolving interactive computing.

Definition 1. *Let $\mathcal{A} = \{A_1, A_2, \ldots\}$ be a sequence of IFA's over Σ, and let Q_i be the set of states of A_i. Let $G = \{G_1, G_2, \ldots\}$ be a sequence of nonempty finite sets such that $G_i \subset Q_i$ and $G_i \subseteq G_{i+1}$. Then $\mathcal{A}$ with G is called a* sequence of IFA's with global states.

For a sequence $\mathcal{A}$, there need not exist an algorithmic way to compute the description of the A_i, given i. Thus, the only way to describe the sequence may be to enumerate all its members. The set $\bigcup_i G_i$ is called *the set of global states*. From now on we always assume sequences of IFA's to have global states.

On an infinite input stream over Σ, a sequence $\mathcal{A}$ computes as follows. At the start, A_1 is the active automaton. It reads input and produces output for a while, until it passes control to A_2. In general, if A_i is the current active automaton, it performs its computation using the *local states* from the set $Q_i - G_i \neq \emptyset$. If an input symbol causes A_i to enter a global state $g \in G_i$, then A_i stops processing and passes control to A_{i+1}. The input stream is redirected to the input port of A_{i+1}, A_{i+1} enters state $g \in G_{i+1}$ and continues processing the input stream as the new active automaton, starting with the next input symbol.

Thus, in effect the input stream is processed by automata with increasing index. This models the property of evolution. The 'transfer' of control to the next automaton is invoked by the automaton currently processing the input. The next automaton continues from the same state in which the previous automaton stopped. This mechanism enables the transfer of information from the previous stage. In a sequence of IFA's with global states the next automaton can be seen as a 'next generation' machine. Note that in finite time only a finite part of a sequence of IFA's can have become active.

Alternatively, instead of a sequence of automata, one may consider a single automaton that 'evolves' so at any time it acts as A_i iff $A_i \in \mathcal{A}$ is the currently active automaton. That is, the transition function of the automaton at hand is the same as that of A_i as long as A_i is active. Of course, the condition concerning the global states must still be maintained. The resulting automaton may appropriately be called an *evolving interactive finite automaton*.

A sequence of IFA's is called *polynomially bounded* iff there is a polynomial p such that for every $i \geq 1$, the size of A_i is at most $p(i)$. The classes of translations realized by sequences of IFA's with global states and polynomially and exponentially bounded size will be denoted as IFA-POLY and IFA-EXP, respectively. We will also consider the classes NA-LOG (the translations realized by sequences of neuromata [8] of logarithmic size), NN-POLY (the translations realized by sequences of standard recurrent, or cyclic, discrete neural nets of polynomial size reading their inputs in parallel), and CC-POLY (the translations realized by sequences of combinatorial circuits with a polynomial number of gates).

4 Interactive Turing Machines

The next tool we consider are interactive Turing machines (ITM's). ITM's differ from standard TM's in one important aspect: they allow for a infinite, never ending exchange of data with their environment.

An ITM reads input from its *input port* and produces output at its *output port*. We assume that in each step the machine reads a symbol from its input port and writes a symbol to its output port. As before, the input and output symbols are taken from the alphabet $\Sigma = \{0, 1, \lambda\}$. We will normally require that an ITM reacts to any non-empty input by producing a non-empty output symbol at its output port after at most some finite time (the *interactiveness* or *finite delay condition*). The finite-delay condition ensures that if an input stream has an infinite number of non-empty symbols, then there must be an infinite number of non-empty symbols in the output stream.

Definition 2. *A mapping* $\phi : \Sigma^\omega \to \Sigma^\omega$ *is called the* interactive translation *computed by an ITM* $\mathcal{I}$ *iff for all* $\mathbf{x}$ *and* $\mathbf{y}$, $\phi(\mathbf{x}) = \mathbf{y}$ *if and only if* $\mathcal{I}$ *produces* $\mathbf{y}$ *on input* $\mathbf{x}$.

As input streams have infinite length, complexity measures for ITM's cannot be defined in terms of the total amount of resources used for processing the entire input, as the resulting values will generally be infinite as well. Therefore we measure the pace of growth of the resource utilizations, as a function of the length of the input stream processed so far.

Definition 3. *We say that for a given input stream the ITM* $\mathcal{I}$ *is of* space complexity $S(t)$ *iff for any* $t > 0$, *after processing* t *input symbols from the given stream no more than* $S(t)$ *cells on* $\mathcal{I}$*'s (internal) tapes were ever needed. If this condition holds for every input stream, then we say that the ITM* $\mathcal{I}$ *is of space complexity* $S(t)$.

The definition of time complexity is more involved. This is so because, as long as empty symbols are counted as legal symbols in input and output streams, any initial segment of a computation by an ITM is of linear time complexity w.r.t. the input read thus far. Yet it is intuitively clear that computing some non-empty output symbols can take more than one step and that in the meantime empty, or other 'ready-made' symbols must have been produced. We will measure the complexity of producing non-empty output symbols from a given input stream, at concrete times, by the *reaction time*. For $t \leq j$, we say that the j-th output *depends* on the input prefix of length t if and only if any change of the $(t+1)$-st and later input symbols cause no change of the output up to and including the j-th symbol. The value j is a (lower)bound to the reaction time for the prefix.

Definition 4. *We say that for a given input stream the ITM* $\mathcal{I}$ *is of* reaction time complexity $T(t)$ *iff the reaction time of* $\mathcal{I}$ *to the input prefix of length* t *is (upper-)bounded by* $T(t)$, *for any* $t > 0$. *If this condition holds for any input stream, then we say that the ITM* $\mathcal{I}$ *is of reaction time complexity* $T(t)$.

ITM's alone do not lead to a non-recursive computational power. Allowing ITM's to process infinite input streams only extends the operational scope, compared to classical TM's which process merely finite streams. For further details see [12,13].

5 Interactive Turing Machines with Advice

Next we introduce *interactive Turing machines with advice* (ITM/A's). An ITM/A is an ITM as described above, enhanced by an advice (cf. [6,2]). Advice functions allow the insertion of external information into the course of a computation, in this way leading to a non-uniform operation.

Definition 5. *An advice function is a function* $f : \mathbf{Z}^+ \to \Sigma^*$. *An advice is called* $S(n)$*-bounded if for all* n, *the length of* $f(n)$ *is bounded by* $S(n)$.

A standard TM with advice and input of size n, is allowed to call for the value of its advice function only for this particular n. An ITM/A can call its advice at time t only for values $t_1 \leq t$. To realize such a call an ITM/A is equipped with a separate *advice tape* and a distinguished *advice state*. By writing the value of the argument t_1 on the advice tape and by entering into the advice state at time $t \geq t_1$ the value of $f(t_1)$ will appear on the advice tape in a single step. By this action, the original contents of the advice tape is completely overwritten.

We will be interested in advice functions whose values are bounded in length by known (computable) functions of t, especially in polynomially or logarithmically bounded functions. Note that the mechanism of advice is very powerful and can provide an ITM/A with highly non-recursive 'assistance'.

The complexity measures for ITM/A's are defined as for ITM's without advice (see Definitions 3 and 4). The length of the rewritten part of the advice tape is counted in the space complexity of the respective machine, not including the actual read-only advice value.

Definition 6. *The class* $ITM - \mathcal{C}/\mathcal{F}$ *consists of the translations* ϕ *computed by* $ITM - \mathcal{C}$ *machines using an advice function from* $\mathcal{F}$.

Common choices for $ITM - \mathcal{C}$ that we shall use are: $ITM - LOGSPACE$ (deterministic logarithmic space), $ITM - PTIME$ (deterministic polynomial time), and $ITM - PSPACE$ (polynomial space). Common choices for $\mathcal{F}$ are *log* (logarithmically bounded advice functions) and *poly* (polynomially bounded advice functions).

For completeness we show that ITM's with advice are indeed more powerful than ITM's without advice. (The result also follows from a countability argument.) Care must be taken that the finite-delay condition is correctly observed.

Consider the translation κ defined as follows. As a special input, we first consider the string consisting of the infinite enumeration of all Turing machine descriptions, in blocks of non-decreasing size. For this input, the translation should assign to each machine description a '1' if and only if the machine at

hand accepts its own description, and '0' otherwise. If the input is not of this form and starts to differ at some point from the blocked form described, then κ is assumed to work as described up until the last complete encoding in the sequence, and then copy every (empty or nonempty) symbol that follows after this.

Lemma 1. *Translation κ can be realized by an ITM/A, but there is no ITM (without advice) that can realize κ.*

Proof (sketch): Define the function f that to each n assigns the description of the 'busy beaver machine' of length n. The busy beaver machine is a Turing machine which, among all machines with encodings of length n, performs the maximum number of steps before halting, on an input that is equal to its own description. If no machine description of size n exists, then $f(n)$ is assigned the empty string.

Now design an ITM $\mathcal{A}$ using advice f as follows. $\mathcal{A}$ checks every time whether the input stream contains a 'next' block as expected, i.e. a next machine description. If the next input segment is not a block as expected, $\mathcal{A}$ will know within finite time that this is the case (because the blocks must come ordered by size). If the next input segment is not a valid encoding, $\mathcal{A}$ copies the segment to output and then copies every (empty or nonempty) symbol that follows after this. This is consistent with κ and satisfies the finite-delay condition.

If the input stream presents $\mathcal{A}$ with a next block that is the valid description w of a Turing machine M, $\mathcal{A}$ works as follows. Let $|w| = n$. $\mathcal{A}$ calls its advice for value n and gets the description $\langle B \rangle$ of length n of the respective busy beaver machine B. Now $\mathcal{A}$ alternately simulates one step of M on input w, and one step of B on input $\langle B \rangle$. Under this arrangement, one of the two simulations must halt as the first one. If it is the simulation of M that halts then $\mathcal{A}$ 'accepts' w, i.e. $\mathcal{A}$ outputs 1. Otherwise, $\mathcal{A}$ outputs 0. Thus, $\mathcal{A}$ realizes κ and, as it satisfies the finite-delay condition in all cases, it is an ITM.

The second part of the lemma is proved by using a modification of the standard diagonal argument. For details, see [15]. □

The lemma serves as a means for proving the super-Turing computing power of machines that are computationally equivalent to ITM/A's. As a first result of this kind we prove that sequences of IFA's with global states are equivalent to ITM/A's, implying that the former also posses super-Turing computing power.

Theorem 2. *For translations $\phi : \Sigma^\omega \to \Sigma^\omega$, the following are equivalent:*

(a) ϕ is computed by a polynomially bounded sequence $\mathcal{A}$ of IFA's with global states.
(b) ϕ is computed by a logarithmically space-bounded ITM/A $\mathcal{M}$ with polynomially bounded advice.

Proof (sketch): $(a) \to (b)$. Let ϕ be computed by $\mathcal{A}$. Simulate the action of $\mathcal{A}$ step by step using an ITM $\mathcal{M}$, with the following advice function. It will be invoked

each time when an automaton A_i in $\mathcal{A}$ currently reading the input enters a global state $g \in G_i$. At this time the advice function returns the description of A_{i+1} and $\mathcal{M}$ proceeds with the simulation of $\mathcal{A}$ by simulating A_{i+1} starting from state $g \in G_{i+1}$. To simulate A_{i+1}, logarithmic space is enough for $\mathcal{M}$ since all it has to store is a pointer to the advice tape remembering the current state of A_{i+1}.

$(b) \rightarrow (a)$. For the reverse simulation, split the input string into blocks of size 1,2,3, $\ldots, i, \ldots$ and consider first the case when $\mathcal{M}$ does not call its advice. Sequence $\mathcal{A}$ will be designed so the i-th automaton in the sequence can 'continue' the processing of the i-th block, making use of its local states. Each automaton A_i is designed as a simulator of $\mathcal{M}$ on input prefixes of lengths $\ell_i = 1+2+3+\ldots+i = O(i^2)$. On these prefixes $\mathcal{M}$ needs $O(\log i)$ space and therefore can enter $O(p(i))$ different states, for some polynomial p. This design warrants not only that A_i has enough states to represent each configuration of $\mathcal{M}$ on an input prefix of length $O(\ell_i)$, but it also enables the 'information transfer', via the corresponding global states, from A_i to A_{i+1} prior to the time when the 'storage capacity' of A_i gets exhausted. Hence, $\mathcal{A}$ can remain polynomially bounded for this case.

Consider now the case when $\mathcal{M}$ can call its advice, which is q-bounded for some polynomial q. In input block i this can happen up to i times. At these moments advices will be of size $O(q(i))$, and at most $O(i^2)$ of them are needed on a length ℓ_i prefix. One can take this into account, by modifying the A_i's so they have all these advices encoded in their states and by simulating $\mathcal{M}$ on the given prefix of the input, this time also including the use of advice by $\mathcal{M}$. The resulting sequence of IFA's with global states is still polynomially bounded.

It is clear that in the given simulations $\mathcal{A}$ satisfies the finite-delay condition iff $\mathcal{M}$ does. □

The theorem implies several analogues for other types of computing devices operating with finite configuration spaces. To circumvent the different input-output conventions in some cases, we call two complexity classes 'equal' only when the devices corresponding to both classes read their inputs sequentially; otherwise, when the devices in one class read their inputs in parallel, we say that they 'correspond'.

Theorem 3. *The following relations hold:*

(a) IFA-POLY equals ITM-LOGSPACE/poly.
(b) NA-LOG equals ITM-LOGSPACE/log.
(c) CC-POLY corresponds to ITM-PTIME/poly.
(d) NN-POLY equals ITM-PSPACE/poly.
(e) IFA-EXP equals ITM-PSPACE/exp.

For later reference we let DSPACE$(S_1(t))/advice(S_2(t))$ denote the complexity class of all $S_1(t)$-space bounded deterministic TM computations making use of $S_2(t)$-space bounded advice.

6 Site Machines

Our next aim is to consider *networks* of machines. Under this scenario the individual interacting machines will be called *site machines*, or simply *sites*. A site machine is an ITM enhanced by a mechanism allowing message sending and receiving very much like well-known *I/O-automata* [7], but it also allows the 'instantaneous' external influencing of its computational behaviour by changes of its transition function in the course of the interaction with the environment. More precisely, sites are viewed as follows.

Individual sites in the network are identified by their address, some symbolic number. The addresses of the sites are managed by a special mechanism that exists outside of the site machines (see Section 7). In order to support the efficient communication among the sites, the respective ITM's are equipped with an *internet tape*. This is a tape whose contents can be sent to any other site. To do so, the sending machine must write the address of the receiving site and its own 'return' address, followed by the message, in an agreed-upon syntax, to its internet tape. By entering into a special distinguished state, the message is sent to the site with the given address. Messages sent to sites with non-existing addresses at that time do not leave the sending machine and the sending machine is informed about this by transiting to another special state.

By sending the message successfully, the internet tape of the sending machine becomes empty in a single step. The message arrives at the receiving machine after some finite time. If at that time the receiving machine finds itself in a distinguished 'message expected' state (which can be superimposed onto other states) then the message is written onto its internet tape, in a single step. The receiver is informed about the incoming message by (enforced) entering into a distinguished state called 'message obtained'. Then the receiving machine can read the message, or copy it onto an auxiliary tape. After reading the whole message the machine can enter into a 'message expected' state again. When it does, its internet tape is automatically emptied in one step.

Otherwise, if the receiving machine is not ready to obtain a message (meaning that the machine is engaged in writing onto or reading from its internet tape), the message enters into a queue and its delivery is tried again in the next step. It may happen that two or more messages arrive to a site simultaneously. This 'write conflict' is resolved by giving priority to the machine with the lowest address, and the remaining messages enter the queue at the site.

Each site is operated by an (external) agent. An agent can work in two modes: *network mode*, in which its machine is logged-in and can communicate with other sites, and *stand-alone* mode, in which its machine is not logged-in. Switching between the two modes is done by the agent with a special instruction.

By entering a suitable input sequence via the input port in network mode, an agent can instruct its machine to do various specific things. First, the agent can instruct it to operate its current 'program', sending or receiving messages, and performing any computation making use of all data stored on the machine's working tapes, on its internet tape, and the data read from the input port. However, while working in network mode the agent is not allowed to change the

machine's hardware and software, i.e., its transition function. This can only be done in stand-alone mode. A change of a transition function may change the number of the machine's tapes, its working alphabet and the number of states. Such an action is done in finite time, during which time the machine is not considered to be the part of the network. Changing the transition function does not affect the data written on the machine's tapes at that time (except for the case when the number of tapes is decreased, when only data on the remaining tapes persist). After changing the transition function, the agent switches back to network mode. The machine then continues operating following the new transition function. Only the inputs read during network mode are considered to be part of the input stream. The same holds for the output stream.

The *instantaneous description* (ID) of a site machine after performing t steps is given by the description of all its working tapes (including its internet tape), the current symbol at its input port, and the corresponding state of its finite control at time t. The current position of the tape heads is assumed to be marked by special symbols on the respective tape descriptions.

At time t, the 'program' of the site machine M at that time, is described by a binary code denoted as $\langle M \rangle$. It encodes, in an agreed-upon syntax, the transition function of the machine. Note that at different times t, a machine with the same address can be described by (operationally) different codes, depending upon the activities of its agent.

To formally describe a site machine, we assume that there is a *site encoding function* δ that maps, for each time t, the address i of a machine to its encoding at that time. The *configuration of a site* at time $t > 0$ after processing t inputs consists of its address, followed by its encoding and its ID at that time.

Theorem 4. *For interactive translations $\phi : \Sigma^\omega \to \Sigma^\omega$, the following are equivalent:*

(a) ϕ is computed by a site machine.
(b) ϕ is computed by an ITM $\mathcal{M}$ with advice.

Proof (sketch): If ϕ is computed by a site machine then the value of the site encoding function at time t can serve as an advice to the simulating ITM/A. Vice versa, when the latter machine has to be simulated by a site machine, then each advice reading can be substituted by a site machine update where the advice value is encoded in the description of the update. □

7 The Web Turing Machine

The ultimate non-uniform model we introduce is the web Turing machine. A web Turing machine (WTM) is a 'time-varying' finite set of interacting sites. The cardinality of this set, the programs of the machines in the set, as well as the message delivery delays can unpredictably vary with time. We only assume that the sites share the same notion of time, i.e., we assume a uniform time-scale within a given WTM.

Let $\mathbb{Z}^+$ denote the set of non-negative integers, and let $\mathbf{N}$ denote the set of natural numbers.

Definition 7. *A web Turing machine $\mathcal{G}$ is a triple $\mathcal{G} = (\alpha, \delta, \mu)$ where:*

- $\alpha : \mathbb{Z}^+ \to 2^{\mathbb{Z}^+}$ *is the so-called* address function *which to each time $t \geq 0$ assigns the finite set of addresses of those sites that at that time are in network mode. Thus, at each time $t \geq 0$, $\mathcal{G}$ consists of the $|\alpha(t)|$ sites from the set $S_t = \{M_i | i \in \alpha(t)\}$ where M_i is the site at the address i.*
- $\delta : \mathbb{Z}^+ \times \mathbb{Z}^+ \to \Sigma^*$ *is the so-called* encoding function *which to each time $t \geq 0$ and address $i \in \alpha(t)$ assigns the encoding $\langle M_i \rangle$ of the respective site M_i at that time at that address.*
- $\mu : \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{N}$ *is the so-called* message transfer function *which to each sending site i and each receiving site j and each time $t \geq 0$ assigns the duration of message transfer from i to j at time when the message is sent, for $i, j \in \alpha(t)$.*

The *description* of a WTM at time $t > 0$ is given by the set of *encodings* of all its sites at that time. The *configuration* of a WTM at time $t > 0$ corresponding to the input read thus far by each site consists of the list of configurations of its sites at that time. The list is ordered according to the addresses of the sites in the list.

A *computation* of a WTM proceeds as follows. At each time, each site which is in network mode and whose address is among the addresses given by the address function for this time, reads a symbol from Σ, possibly the empty symbol, from its input. Depending on this symbol and on its current configuration the machine performs its next move by updating its tapes, state and outputting a symbol (possibly λ) to its output, in accordance with its transition function. Within a move the machine can send a message to or receive a message from an other machine. Also, when the machine is in stand-alone mode, its agent can modify the transition function of the machine or the data represented on the machine's tape. Moreover, at any time an agent can 'log in' ('log out') a site into (from) the WTM by entering the respective mode of operation. This fact is recorded by the values of the functions α and δ that must change accordingly at that time, to reflect the new situation.

Any WTM acts as a *translator* which at each time reads a single input symbol and produces a single output symbol at each site (some of the symbols may be empty). In this way a WTM computes a mapping from finite or infinite streams of input symbols at the sites to similar streams of output symbols. The number of streams varies along with the number of sites. The incoming messages are not considered to be a part of the input (as they arrive over different ports). Of course, the result of a translation does depend on the messages received at individual sites and on their arrival times at these sites. However, all messages are results of (internal) computations and therefore their sending times are uniquely determined; the arrival times to their destinations are given by μ. Therefore, for a given 'packed' stream of inputs (with each packed symbol unfolding to an input at every site), the result of the translation is uniquely determined.

Definition 8. *Let $\mathcal{G} = (\alpha, \delta, \mu)$ be a WTM. The* web translation *computed by $\mathcal{G}$ is the mapping Γ such that for all packed streams* $\mathbf{x}$ *and* $\mathbf{y}$*, $\Gamma(\mathbf{x}) = \mathbf{y}$ iff on input* $\mathbf{x}$ *to its sites, machine $\mathcal{G}$ produces* $\mathbf{y}$ *at its sites.*

The *space complexity* $S(t)$ of $\mathcal{G}$ at time t is the maximum space consumed by any site of $\mathcal{G}$, over all input streams of length t. The respective complexity class will be denoted as $WTM - DSPACE(S(n))$. By $WTM - PSPACE$ and $WTM - DLOGSPACE$ we will denote the classes of all polynomially and logarithmically space-bounded web translations, respectively. For a further discussion of the complexity issues, see Section 9.

We conclude this section by a few comments related to the definition of a WTM and its operation. First note that we did not require either α, δ or μ to be recursive functions. Indeed, in general there is neither a known computable relation between the time and the addresses of the site machines, nor between the addresses and the site descriptions or between sender-receiver addresses and message transfer times. Thus, for each $t \geq 0$ the three functions are given by finite tables at best. Second, note that we assumed that at each time $\mathcal{G}$ consists only of a finite number of sites, as implied by the definition of α. Also note that the transfer time of a message depends not only on the address of the sending and receiving sites but also on the message issuing time. This means that even if a same message is sent from i to j at different times, the respective message transfer times can differ. Message delivery time is assumed to be independent of the message length. This assumption may be seen as being too liberal but dependences can be considered to be amortized over the time needed to write the message to the internet tape.

8 The Power of Web Computing

At each moment in time, the architecture and the functionality of a WTM are formally described by its functions α and δ. These two functions model the fact that in practice (as in the case of the Internet) the evolution of the machine depends both on the input to the individual sites and on the decisions of the respective agents from which the changes in network architecture and site functionality may result. The agent decisions may in turn also depend on the results of previous computations and on messages received at individual sites as seen by their respective agents. Under this scenario the description of a WTM may change from time to time in a completely unpredictable manner.

Due to the finiteness of a WTM at each time, its δ at a given time is always finite. Nevertheless, the size of the encoding function over its entire existence, for $t = 1, 2, \ldots$, is in general infinite. Intuitively, this is the reason why a WTM cannot be simulated be a single ITM with a finite encoding. However, for each time t the encoding of a WTM can be provided by an advice function, as shown in the following theorem.

There is one technical problem in the simulation of a WTM by an ITM/A. Namely, by its very definition a WTM computes a mapping from packed input streams to packed output streams, with packed symbols of variable size. The

respective symbols are read and produced in parallel, synchronously at all sites. However, a normal TM, working as a translator of infinite input streams into infinite output streams using a single input and a single output tape, cannot read (and produce) packed symbols of variable size in one step, in a parallel manner. What it can is to read and produce packed symbols component-wise, in a sequential manner.

In order to solve this technical problem we assume that the simulating ITM/A has a specific 'architecture' tailored to the problem at hand. First, we assume that it has a single, infinite one-way read-only input tape at which the original stream of packed inputs $\{(x_{i_1,t}, x_{i_2,t}, \ldots, x_{i_k,t})\}_{t=0}^{\infty}$ to G's sites with $\{i_1, i_2, \ldots, i_k\} = \alpha(t)$, is written as follows: for consecutive $t = 1, 2, \ldots$, it contains a 'block' of inputs $(x_{i_1,t}, x_{i_2,t}, \ldots, x_{i_k,t})$. Blocks and input symbols are separated by suitable marking symbols. Second, we assume that the ITM/A has one infinite one-way write-only output tape to which outputs are written of the form $\{(y_{i_1,t}, y_{i_2,t}, \ldots, y_{i_k,t})\}_{t=0}^{\infty}$, again in a block-wise manner. A pair of two infinite streams of input and output symbols thus obtained is called the *sequential representation* of a web translation Γ.

Theorem 5. *For every WTM $\mathcal{G}$ there exists a single ITM/A $\mathcal{A}$ that acts as a sequential translator of the web translation computed by $\mathcal{G}$, and vice versa.*

Proof (sketch): On its tapes $\mathcal{A}$ keeps the ID's of all sites in $\mathcal{G} = (\alpha, \delta, \mu)$, with their current modes. In the advice, the values of all three functions α, δ and μ are stored. Thanks to this, $\mathcal{A}$ can sequentially update the sites in accordance with the instructions and updates performed by each site.

The idea of the reverse simulation is to show that a single site operated by a suitable agent can simulate a ITM/A. The role of the agent will be to deliver the values of the advice function at times when needed. The machine can do so by switching to stand-alone mode and letting its agent exchange its program for the program that has the value of the advice encoded in its states. Then the interrupted computation will resume. The details are given in [15]. □

9 The Efficiency of Web Computing

The equivalence between WTM- and ITM/A computations was proved with the help of simulations. Because we were primarily interested in characterizing the computing power of WTM's, no attempt was made to make the simulations as efficient as they could be and to relate the complexity classes of the two models. In this section we investigate the computational efficiency of 'web space' and 'web time'.

To get rid of some repeated assumptions in the theorems below we will bound the growth of the 'parameters' of a WTM over time, viz. its number of sites, the sizes of its site descriptions, and the message transfer times. More precisely:

Definition 9. *A WTM $\mathcal{G} = (\alpha, \delta, \mu)$ is called* S(t)-bounded *if it satisfies the following restrictions:*

- *the space complexity of* $\mathcal{G}$ *is* $S(t)$, *for all* $t \geq 0$,
- *the address length of any site in* $\mathcal{G}$ *does not grow faster than the space complexity of* $\mathcal{G}$, *i.e., for all* $t \geq 0$ *and any address* $i \in \alpha(t)$ *we have* $|i| = O(S(t))$,
- *the size of any site encoding does not grow faster than the space complexity of* $\mathcal{G}$, *i.e., for* $t \geq 0$ *we have* $|\langle M_j \rangle| = O(S(t))$ *for all* $j \in \alpha(t)$, *and*
- *the message transfer times are not greater in order of magnitude than the total size of* $\mathcal{G}$, *i.e., for all* $t \geq 0$ *and* $i, j \in \alpha(t)$ *we have* $\mu(i, j, t) = O(|\alpha(t)|)$.

The first restriction bounds the space complexity of each site. The second one allows at most an exponential growth (in terms of $S(t)$) of the synchronous WTM with time. The third restriction is also quite realistic; it says that the 'program size' at a site should not be greater than the size of the other data permanently stored at that site. The fourth restriction together with the second one guarantees that the size (e.g. in binary) of the values of the message transfer function is also bounded by $O(S(t))$. Note that by restriction one, the message length is also bounded by $O(S(t))$, since in the given space no longer messages can be prepared.

In the complexity calculations that follow we will always consider a $S(t)$-bounded, or 'bounded' WTM. We will first show that any bounded WTM is equivalent to an exponential space-bounded deterministic ITM using an exponential size advice.

Theorem 6. *For all space bounding functions* $S(t) \geq 0$,

$$\bigcup_{c>0} WTM - DSPACE(cS(t)) = \bigcup_{c>0} ITM - DSPACE(c^{S(t)})/advice(c^{S(t)}).$$

In particular,

$$WTM - DLOGSPACE = ITM - PSPACE/poly$$

Proof (sketch): The left-to-right inclusion is proved by keeping a 'mirror image' of the WTM on the tapes of the ITM/A. Since our WTM is $S(t)$-space bounded it can have up to $O(c^{S(t)})$ sites of size $S(t)$, for some c. Hence the mirror image of the WTM can be maintained in exponential space as claimed. For each t the advice size is also bounded by the same expression and the ITM/A uses it to simulate the WTM updates. For proving the opposite inclusion we simulate the i-th cell of the ITM/A by a special site that keeps the contents of this cell plus the information whether the machine's head is scanning this cell. The advice tape is represented in a similar manner. For the full proof see [15]. □

This result for space-bounded WTM computations is analoguous to similar results known for so-called synchronized computations in uniform models (see for example [5], [18]).

Next we study 'time' as a computational resource for WTM's, viz. the potential of a WTM to perform parallel computations. In order to make use of this potential one has to ensure e.g. when sending requests to two sites to run some

computations, that these requests will be accomplished with only a small delay. Similarly, after finishing the computation, one has to ensure that both results will be returned again with only a small delay. This cannot be guaranteed under the original mild assumption that each message will be delivered in an finite, albeit unpredictable time.

In order to enable a genuinely parallel realization of computations we therefore strengthen the restriction on the duration of message deliveries within a bounded WTM further. We introduce the *unit cost* WTM in which each message is assumed to be delivered to its destination within unit time.

Definition 10. *A unit-cost WTM $\mathcal{G}$ is a bounded WTM $\mathcal{G} = (\alpha, \delta, \mu)$ in which $\mu(i, j, t) = 1$ for all $i, j \in \alpha(t)$ and $t \geq 0$.*

Let $WTM_U - PTIME$ denote the class of all translations that can be realized by a unit-cost WTM within polynomial reaction time.

It turns out that a unit-cost WTM can simulate an ITM/A very fast, by involving an exponential number of sites in the simulation. Vice versa, a fast WTM with 'many processors' can be simulated by an ITM/A in 'small' space. The simulation is sketched in the proof of the following theorem. When speaking about the respective models we shall use similar input/output conventions as those in Section 8.

Theorem 7.

$$WTM_U - PTIME = ITM - PSPACE/poly$$

Proof (sketch): When attempting to simulate a polynomial time-bounded WTM in polynomial space on a ITM/A, we run into the problem that a WTM can activate an exponential number of processors whose representations cannot all be kept on a tape simultaneously. Thus, a strategy must be designed for reusing the space and recomputing the contents of each site when needed. The non-uniformity of WTM updates is simulated, as expected, by advice calls.

To simulate a ITM/A of polynomial space complexity $S(t)$ on a WTM of polynomial time complexity, imagine the infinite computational tree T of the ITM/A computations. For a given input, consider the subtree of T of depth $c^{S(t)}$ with the same root as T and an exponential number of nodes. The simulated ITM/A processes the first t inputs successfully iff the path in T that starts in an initial ID, ends in an ID that produces the 'further' output which is a reaction to the t-th input. The existence of such an accepting path is found by making use of the parallel version of algorithm that computes the transitive closure of T in polynomial space w.r.t. $S(t)$. This simulation must be run for each $t = 1, 2, \ldots$. This is achieved by starting the simulation at each time t for that particular value of t at a suitable site on the WTM. The involved details of both parts of the sketched proof can be found in [15]. □

Corollary 1. *For any $S(t) \geq 0$*

$$\bigcup_{c>e0} WTM - DSPACE(cS(t)) = \bigcup_{c>0} WTM_U - TIME(c^{S(t)}).$$

In particular,

$$WTM - DLOGSPACE = WTM_U - PTIME$$

The last result says that (bounded!) WTM's make use of their space in an optimal way: in the given space one cannot perform more time-bounded computations than a unit-cost WTM does.

The result on time-bounded unit-cost WTM's and its proof, mirrors the similar result for uniform, idealized computational models from the 'second machine class' (cf. [11]). Its members fulfill the so-called *Parallel Computation Thesis* which states that sequential polynomial space is equivalent to parallel polynomial time on devices from this class. In the non-uniform setting similar results are known for infinite families of neural networks of various kinds (for a recent overview of known results see [8]).

The results on time and space efficiency of WTM computations rank WTMs among the most powerful and efficient computational devices known in complexity theory.

10 Afterthoughts

Theorem 1 points to a rich world of interactive devices that can serve as a basis for the investigation of evolving interactive computing systems. In fact, in [20] these devices have been interpreted as *cognitive automata*. Each of them can serve as a model of a 'living organism' and can be used for further studies of the computational aspects of complex systems created from these elementary computing units.

Theorem 3 reveals that not all cognitive automata are equally efficient from the viewpoint of their descriptional economy: systems representing their configuration space in unary representation (this is the case of finite automata) suffer from space inefficiency. Other systems that make use of more efficient state representations and can reuse their space, such as neural nets, are much more effective from this point of view.

Theorems 2, 4, and 5 point to the central equivalence of the various models, summarized in the following Theorem. It points to the fact that the notion of evolving interactive computing is a robust and fundamental one.

Theorem 8. *For translations $\phi : \Sigma^\omega \to \Sigma^\omega$, the following are equivalent:*

(a) ϕ is (sequentially) computed by a sequence of IFA's with global states.
(b) ϕ is (sequentially) computable by an ITM/A.
(c) ϕ is (sequentially) computable by a site machine.
(d) ϕ is computable by a WTM.

Lemma 1 shows the super-Turing computing power of the ITM/A's and separates the model from ITM's. It also implies that the WTM, which can be seen as a quite realistic model of the Internet as far as its computing power is concerned, can perform computations that cannot be replicated by any standard

interactive TM. This answers Wegner's claim concerning the power of interactive computing. Interaction can lead to computations that no Turing machine can mimic, providing that we allow updates of the underlying machinery and consider unbounded, potentially infinite computations. The latter condition is a crucial one since otherwise one would have but a finite number of updates that could be built-in beforehand into the architecture of the ITM.

Theorem 8 also points to the various ways in which non-uniform features may enter into a computing system. First, non-uniformity can be hidden in the 'architecture' of a computing system. This is the case in sequences of finite automata with global states where the description of the system as a whole is given by an infinite, in general non-computable string. Second, non-computable information may enter into a computing system from an 'external' source. E.g. in the case of ITM/A's, this is done by advice functions, and in the case of site machines or WTM's there are agents that can change the machine architecture in an unpredictable manner.

The results from theorem 8 have interesting interpretations in the world of cognitive automata and computational cognition. The basic idea is as follows: in a 'robotic' setting, any interactive finite automaton (viewed as a *cognitive automaton*) can be seen as a simple model of a living organism. A (finite) set of cognitive automata can communicate basically in two different ways. First, the automata can communicate using a fixed 'pre-wired' communication pattern, so to speak holding hands with their physically immediate neighbors. By this we get systems equivalent to sequences of interactive finite automata. If the automata communicate in arbitrary patterns or do not have global states, the corresponding system of cognitive automata resembles certain types of *amorphous computing systems* [1]. In principle it is no problem to define cognitive automata in such a way that they will also possess a replication ability. Then one can consider systems of cognitive automata that grow while computing. As a result one gets various *morphogenetic computational systems*.

The second possibility for cognitive automata to communicate, is the case when the automata are equipped with sensors and effectuators by which they scan and change their environment. In the case of ordinary TM's the 'living environment' of a single cognitive automaton working under such conditions is modelled by TM tapes and read/write heads. Following this analogy further, a WTM can be seen as a set of cognitive automata. They share the same living environment and communicate via message exchange. They can even move and exchange messages, either by encountering each other, or leaving a message elsewhere (probably in a distinguished place) in the environment or by sending a message via a chain of neighbors. A specific view of a human society as that of a community of agents communicating by whatever reasonable means (language, e-mail, letters, messengers, etc.) also leads to a model of WTM with specific parameters. What is important and interesting from the point of view of cognitive sciences is the fact that irrespectively which possibility is taken, we always get a system equivalent to a WTM and hence in general possessing a super-Turing computing power.

A challenging question still remains unanswered: could one indeed make use of the super-Turing potential of the underlying machines to one's advantage? Can one solve certain concrete undecidable problems by such machines? The answer is, (un)fortunately, no. From a practical point of view our results mean that the corresponding devices cannot be simulated by standard TM's working under a classical scenario. This is because the evolving interactive machinery develops in an unpredictable manner, by a concurrent unpredictable activity of all agents operating the sites.

Nevertheless, the above results point to quite realistic instances where the classical paradigm of a standard Turing machine as the generic model which captures all computations by digital systems, is clearly insufficient. It appears that the time has come to reconsider this paradigm and replace it by its extended version, viz. by ITM's with advice. For a more extended discussion of the related issues, see [14].

References

1. H. Abelson, D. Allen, D. Coore, Ch. Hanson, G. Homsy, T.F. Knight, R. Nagpal, E. Rauch, G.J. Sussman, R. Weiss: Amorphous computing, *Comm. ACM*, Vol. 42, No. 5, May 2000, pp. 74-82.
2. J.L. Balcázar, J. Díaz, J. Gabarró : *Structural Complexity* I, Second Edition, Springer-Verlag, Berlin, 1995.
3. G. Berry: The foundations of Esterel, in: G. Plotkin, C. Stirling and M. Tofte (Eds), *Proof, Language, and Interaction - Essays in Honour of Robin Milner*, The MIT Press, Cambridge MA, 2000, pp 425-454.
4. L. Cardelli: Global computation, *ACM Sigplan Notices*, Vol. 32, No. 1, 1997, pp. 66-68.
5. J. Dassow, J. Hromkovič, J. Karhumäki, B. Rovan, A. Slobodová: On the power of synchronization in parallel computing, Computing, in: A. Kreczmar and G. Mirkowska (Eds), *Mathematical Foundations of Computer Science 1989*, Proceedings, Lecture Notes in Computer Science, Vol 379, Springer-Verlag, Berlin, 1989, pp. 196-206.
6. R.M. Karp, R.J. Lipton: Some connections between non-uniform and uniform complexity classes, in: *Proc. 12th Annual ACM Symposium on the Theory of Computing* (STOC'80), 1980, pp. 302-309, revised as: Turing machines that take advice, *L'Enseignement Mathématique*, II^{e} Série, Tome XXVIII, 1982, pp. 191-209.
7. N.A. Lynch: *Distributed algorithms*, Morgan Kaufmann Publishers Inc., San Francisco CA, 1996.
8. P. Orponen: An overview of the computational power of recurrent neural networks, in: Proc. Finnish AI Conference (Espoo, Finland, August 2000), Vol. 3: *AI of Tomorrow*, Finnish AI Society, Vaasa, 2000, pp. 89-96.
9. J. Šíma, J. Wiedermann: Theory of Neuromata. *Journal of the ACM,* Vol. 45, No. 1, 1998, pp. 155-178.
10. L.G. Valiant: *Circuits of the Mind*, Oxford University Press, New York, 1994.
11. P. van Emde-Boas: Machine models and simulations, in: J. van Leeuwen (ed.). *Handbook of Theoretical Computer Science*, Vol. A: Algorithms and Complexity, Elsevier Science Publ, Amsterdam, 1990, pp. 3-66.

12. J. van Leeuwen, J. Wiedermann: On algorithms and interaction, in: M. Nielsen and B. Rovan (Eds), *Mathematical Foundations of Computer Science 2000*, 25th Int. Symposium (MFCS'2000), Lecture Notes in Computer Science, Vol. 1893, Springer-Verlag, Berlin, 2000, pp. 99-112.
13. J. van Leeuwen, J. Wiedermann: A computational model of interaction in embedded systems, Technical Report UU-CS-02-2001, Dept. of Computer Science, Utrecht University, 2001.
14. J. van Leeuwen, J. Wiedermann: The Turing machine paradigm in contemporary computing, in: B. Enquist and W. Schmidt (Eds), *Mathematics Unlimited - 2001 and Beyond*, Springer-Verlag, Berlin, 2001, pp. 1139-1155.
15. J. van Leeuwen, J. Wiedermann: Breaking the Turing barrier: the case of the Internet, manuscript in preparation, February 2001.
16. P. Wegner: Why interaction is more powerful than algorithms, *C. ACM* 40, (1997) 315-351.
17. P. Wegner, D.Q. Goldin: Interaction, computability, and Church's thesis, *The Computer Journal* 2000 (to appear).
18. J. Wiedermann: On the power of synchronization, *J. Inf. Process. Cybern.* (EIK), Vol. 25, No. 10, 1989, pp. 499-506.
19. J. Wiedermann: The computational limits to the cognitive power of neuroidal tabula rasa, in: O. Watanabe and T. Yokomori (Eds), *Algorithmic Learning Theory*, Proc. 10th International Conference (ALT'99), Lecture Notes in Artific. Intelligence, Vol. 1720, Springer-Verlag, Berlin, 1999, pp. 63-76.
20. J. Wiedermann, J. van Leeuwen: Emergence of super-Turing computing power in artificial living systems, in: J. Kelemen (Ed.), *Artificial Life 2001*, Proceedings 6-th European Conference (ECAL 2001), Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, 2001 (to appear).

Distributed Computations by Autonomous Mobile Robots

Nicola Santoro

School of Computer Science, Carleton University, Ottawa
santoro@scs.carleton.ca

1 Introduction

Most of the concerns of Distributed Computing may appear in settings which are quite different from its traditonal applications ares such as distributed systems, data and communication networks, etc. An important setting of this type is the one of *autonomous mobile robots*.

In fact, the attention in robotics research has moved from the study of very few, specialized, rather complex robots to a multiplicity of generic, simple units. This shift in attention has occurred both in the robotics engineering and in the artificial intelligence communities. Leading research activities in the engineering area include the Cellular Robotic System (CEBOT) of Kawaguchi et al [12], the Swarm Intelligence of Beni et al. [4], the Self-Assembly Machine ("fructum") of Murata et al. [14], etc. In the AI community there has been a number of remarkable studies, eg., on social interaction leading to group behavior by Matarić [13], on selfish behavior of cooperative robots in animal societies by Parker [16], on primitive animal behavior in pattern formation by Balch and Arkin [3], to cite a just a few. An investigation with an algorithmic flavor has been undertaken within the AI community by Durfee [6], who argues in favor of limiting the knowledge that an intelligent robot must possess in order to be able to coordinate its behavior with others.

The motivations for this research shift are different, ranging from economic concerns (simpler robots are less expensive to design, produce and deploy) to philosophical questions (can complex behaviour emerge from the interaction of extremely simple entities?). The setting being considered is a community of identical extremely simple mobile robots which are possibly capable, collectively, to perform a given task. In all these investigations, the algorithmic (i.e., the computational and software) aspects were somehow implicitly an issue, but clearly not a major concern. We now provide an overview of some recent algorithmic developments on the coordination and control of such a community of autonomous mobile robots. These developments stem from research on the algorithmic limitations of what a community of such robots can do.

2 Overview

The algorithmic research has been carried out by two groups. The earliest and pioneering work is that of Suzuki and Yamashita and their collaborators

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 110–115, 2001.

[1,15,18,17] who established several results under the assumption that movement (as well as any other robot's action) is *instantaneous*. The more recent research work is by Flocchini et al [7,8,9] who remove this assumption. We now describe their general model.

Each robot is capable of sensing its immediate surrounding, performing computations on the sensed data, and moving towards the computed destination; its behavior is an (endless) cycle of sensing, computing, moving and being inactive. The robots are viewed as points, and modeled as units with computational capabilities, which are able to freely move in the plane. They are equipped with sensors that let each robot observe the positions of the others and form its *local view* of the world. This view includes a unit of length, an origin (which we will assume w.l.g. to be the position of the robot in its current observation), and a Cartesian coordinate system with origin, unit of length, and the *directions* of two coordinate axes, identified as x axis and y axis, together with their *orientations*, identified as the positive and negative sides of the axes. Notice, however, that the local views could be totally different making impossible for the robots to agree on directions or on distances.

The robots are *anonymous*, meaning that they are a priori indistinguishable by their appearances, and they do not have any kind of identifiers that can be used during the computation. Moreover, there are no explicit direct means of communication. The robots are *fully asynchronous*: the amount of time spent in observation, in computation, in movement, and in inaction is finite but otherwise unpredictable[1]. In particular, the robots do not (need to) have a common notion of time. The robots may or may not remember previous observations or computations performed in the previous steps; they are said to be *oblivious* if they do not remember.

The robots execute the same deterministic algorithm, which takes as input the observed positions of the robots within the visibility radius, and returns a destination point towards which the executing robot moves.

A robot is initially in a *waiting* state (*Wait*); asynchronously and independently from the other robots, it *observes* the environment in its area of visibility (*Look*); it *calculates* its destination point based only on the observed locations of the robots in its (*Compute*); it then *moves* towards that point (*Move*); after the move it goes back to a waiting state.

The sequence: *Wait* - *Look* - *Compute* - *Move* will be called a *computation cycle* (or briefly *cycle*) of a robot.

The operations performed by the robots in each state will be now described in more details.

1. **Wait** The robot is idle. A robot cannot stay infinitely idle (see Assumption A1 below).
2. **Look** The robot observes the world by activating its sensors which will return a snapshot of the positions of all other robots with respect to its local coordinate system. (Since robots are viewed as a point, their positions in the plane is just the set of their coordinates).

[1] Suzuki and Yamashita assume instead that is *instantaneous*.

3. **Compute** The robot performs a *local computation* according to its deterministic, oblivious algorithm. The result of the computation is a destination point; if this point is the current location, the robot stays still (*null movement*),
4. **Move** The robot moves towards the computed destination; this operation can terminate before the robot has reached it[2]. The movement can not be infinite, nor infinitesimally small .

3 Results

3.1 Pattern Formation with Unlimited Visibility

Consider the coordination problem of forming a specific geometric pattern in the unlimited visibility setting. The *pattern formation* problem has been extensively investigated in the literature (e.g., see [5,17,18,19]), where usually the first step is to gather the robots together and then let them proceed in the desired formation (just like a flock of birds or a troupe of soldiers). The problem is practically important, because, if the robots can form an given pattern, they can agree on their respective roles in a subsequent, coordinated action.

The geometric patterns is a set of points (given by their Cartesian coordinates) in the plane, and it is initially known by all the robots in the system.

The robots *form the pattern*, if, at the end of the computation, the positions of the robots coincides, in everybody's local view, with the points of the pattern, where the pattern may be *translated*, *rotated*, *scaled*, and *flipped* into its mirror position in each local coordinate system. Initially, the robots are in arbitrary positions, with the only requirement that no two robots be in the same position, and that, of course, the number of points prescribed in the pattern and the number of robots are the same.

The pattern formation problem is quite a general member in the class of problems that are of interest for autonomous, mobile robots. It includes as special cases many coordination problems, such as leader election, where the pattern is defined in such a way that the leader is uniquely represented by one point in the pattern.

Suzuki and Yamashita [18] solve this problem with instantaneous movements, characterizing what kind of patterns can be formed. All their algorithms are non-oblivious.

Without instantaneous movements, the following theorem summarizes the results holding for a set of n autonomous, anonymous, oblivious, mobile robots:

Theorem 1. *([7])*

1. *With common knowledge of two axis directions and orientations, the robots can form an arbitrary given pattern.*
2. *With common knowledge on only one axis direction and orientation, the pattern formation problem is unsolvable when n is even.*

[2] e.g. because of limits to the robot's motorial autonomy.

3. *With common knowledge only on axis direction and orientation, the robots can form an arbitrary given pattern if n is odd.*
4. *With no common knowledge, the robots cannot form an arbitrary given pattern.*

The class of patters which can be formed with common knowledge on only one axis direction and orientation when n is even (i.e., when the arbitrary pattern formation problem is unsolvable) has been fully characterized [9].

3.2 Flocking with Unlimited Visibility

Consider *flocking*: a set of mobile units are required to follow a leader unit while keeping a predetermined formation (i,e,. they are required to move in a flock, like birds or a group of soldiers).

Flocking has been studied for military and industrial applications [2,3,19], assuming that the path of the leader is known to all units in advance.

The more interesting and challanging problem arises when the units in the flock do *not know* beforehand the path the leader will take; their task is just to follow it eherever it goes, and to keep the formation while moving. An algorithmic solution, without assuming neither a priori knowledge of the path nor its derivability, has been recently presented in [10]; the algorithm only assumes the robots share a common unit of distance, but no common coordinate system is needed.

3.3 Gathering with Limited Visibility

Consider *gathering*: the basic task of having the robots meet in a single location (the choice of the location is not predetermined). Since the robots are modeled as points in the plane, the task of robots gathering is also called the *point formation problem*. Gathering (or point formation) has been investigated both experimentally and theoretically in the *unlimited visibility* setting, that is assuming that the robots are capable to sense ("see") the entire space (e.g., see [7,11,17,18]). In general, and more realistically, robots can sense only a surrounding within a radius of bounded size. This setting, called the *limited visibility* case, is understandably more difficult, and only few algorithmic results are known [1,18].

In the limited visibility setting, this problem has been investigated by by Ando et al. [1], who presented an oblivious procedure that *converges* in the limit, but *does not reaches*, the point. Furthemore, instantaneous actions are assumed.

In [8], Flocchini et al. proved that the availability of *orientation*[3] enables anonymous oblivious robots with limited visibility to gather within a *finite* number of *moves* even if they are fully asynchronous.

[3] i.e., agreement on axes and directions (positive vs. negative) of a common coordinate system, but not necessarily on the origin nor on the unit distance

This shows that gathering can be performed in a finite number of moves by simpler robots with fewer restrictions than known before, provided they have a common orientation.

From a practical point of view, this result has immediate consequences. In fact, it solves the problem without requiring the robots to have physically unrealizable motorial and computing capabilities ("instantaneous actions"), and using instead a property ("orientation") which is both simple and inexpensive to provide (e.g., by a *compass*).

4 Open Problems

The known results are few and open many interesting research directions, among them:

- the study of coordination problems under relaxed assumptions about the robots' capabilities (e.g., the robots are not totally oblivious, and can remember only a constant amount of information);
- the investigation of simple tasks under different conditions of the environment (e.g., in presence of obstacles, or on uneven terrains);
- the study of the impact that sensorial errors, possibly arising during the *Look* and the *Move* state, have on the overall correctness of the algorithms;
- the design of new algorithms under different assumptions on the visibility power of the robots (e.g., the accuracy of the robots' ability to detect the other robots' positions decreases with the distance);
- the study of new problems like *scattering* (i.e., the robots start from the same location and their goal is to evenly scatter on the plane), *rescue* (i.e., the robots have to find a small object which is not initially visible), *exploration* (i.e., the robots have to gather information about the environment, with the purpose, for example, of constructing a map), and many more.

At a more general level, there are interesting fundamental research questions. For example, how to compare different solutions for the same set of robots? So far, no complexity measure has been proposed; such a definition is part of our future research.

Acknowledgements

This work has been supported in part by NSERC, NORTEL Networks, and MITACS Center of Excellence.

References

1. 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.

2. R.C. Arkin. Motor Schema-based Mobile Robot Navigation. *Int. J. of Robotics and Research*, 8(4):92–112, 1989.
3. T. Balch and R.C. Arkin. Behavior-based Formation Control for Multi-robot Teams. *IEEE Transaction on Robotics and Automation*, 14(6), 1998.
4. G. Beni and S. Hackwood. Coherent swarm motion under distributed control. In *Proc. DARS'92*, pages 39–52, 1992.
5. Q. Chen and J. Y. S. Luh. Coordination and control of a group of small mobile robots. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 2315–2320, 1994. San Diego, CA.
6. E. H. Durfee. Blissful Ignorance: Knowing Just Enough to Coordinate Well. In *ICMAS*, pages 406–413, 1995.
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 *Proc. of 10th International Symposium on Algorithm and Computation (ISAAC '99)*, pages 93–102, December 1999.
8. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Autonomous Mobile Robots With Limited Visibility. In *Proc. of 18th International Symposium Theoretical Aspects of Computed Science (STACS 2001)*, pages 247–258, 2001.
9. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Pattern formation by anonymous robots without chirality. In *Proc. 8th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2001)*, pages 147–162, 2001.
10. V. Gervasi and G. Prencipe. Need a Fleet ? Use the Force. In *Proc. of 2nd International Conference on Fun with Algorithms*, pages 149–164, June 2001. Elba.
11. D. Jung, G. Cheng, and A. Zelinsky. Experiments in Realising Cooperation between Autonomous Mobile Robots. In *Fifth International Symposium on Experimental Robotics (ISER)*, June 1997. Barcelona, Catalonia.
12. Y. Kawauchi and M. Inaba and. T. Fukuda. A principle of decision making of cellular robotic system (cebot). In *Proc. IEEE Conf. on Robotics and Automation*, pages 833–838, 1993.
13. M. J Matarić. *Interaction and Intelligent Behavior*. PhD thesis, MIT, May 1994.
14. S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proc. IEEE Conf. on Robotics and Automation*, pages 441–448, 1994.
15. Y. Oasa, I. Suzuki, and M. Yamashita. A Robust Distributed Convergence Algorithm for Autonomous Mobile Robots. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 287–292, October 1997.
16. L. E. Parker. On the Design of Behavior-Based Multi-Robot Teams. *Journal of Advanced Robotics*, 10(6), 1996.
17. K. Sugihara and I. Suzuki. Distributed Algorithms for Formation of Geometric Patterns with Many Mobile Robots. *Journal of Robotics Systems*, 13:127–139, 1996.
18. I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *Siam J. Comput.*, 28(4):1347–1363, 1999.
19. P. K. C. Wang. Navigation Strategies for Multiple Autonomous Mobile Robots Moving in Formation. *Journal of Robotic Systems*, 8(2):177–195, 1991.

Formal Verification Methods for Industrial Hardware Design

Anna Slobodová

Compaq Computer Corporation

Abstract. Functional validation of hardware designs is a major challenge for circuit design companies. Post-delivery software problems can be addressed by subsequent software releases; however, fixing hardware bugs in any shipped product is expensive. Simulation remains the dominate functional validation method, but in the last decade, formal verification (FV) has emerged as an important complementary method. We describe basic FV methods: theorem proving, model checking, and equivalence checking with some illustrations from their applications to Alpha microprocessor designs. The last one is described in detail. Although theoretically, FV can provide much more complete verification coverage than simulation, our ability to apply FV is limited due to capacity limits of existing FV tools and the availability of trained personnel. The application of FV to industrial designs is an active research area with huge opportunities for academic and industrial researchers.

1 Introduction

The difficulty of validating modern microprocessor designs has dramatically increased during the last two decades as the complexity of microprocessor circuits has increased. Increasing transistor density (which quadruples every three years), device speed, and die size, impact all aspects of design. Designers must correctly target future technology so they produce the right design at the right time. Besides technology changes, chip designers are driven by changes in the competitive marketplace, where emerging applications may require new functionality, redistribution of the resources, higher bandwidth, more parallelism, etc. All these factors have influence on microprocessor architecture and its final design. The rising volume shipments increase the repair costs that a company bears in case of a flawed design. Designers experience tremendous pressure when coping with two controversial requirements – to engineer a *correct* design on a *short development schedule*; under these circumstances, design validation is a real challenge.

A correct design has to obey various physical design rules while providing the required functionality. In this paper, we will discuss only a small subset of validation methods; specifically, *functional* verification, further restricted to *formal verification* (FV) methods.

The goal of functional verification is to assure the logical correctness of a design, exclusive of physical requirements like timing and power usage. In this paper, we only address the design verification problem using formal methods,

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 116–135, 2001.

and we do not discuss other verification methods like simulation and post-manufacturing testing. This paper is not intended to be an overview of all formal verification methods, neither does it provide an exhaustive description of the way formal methods can be applied in an industrial design flow. Note, design methods may be quite different from company to company, and even from project to project within a single company. In some cases, different methods may be used on different parts of the same design and in different phases of the project. Our goal is to give you the flavor of industrial hardware verification problems, and how some of them were addressed using FV methods in Alpha Development Group.

The author would like to share with you her experience in the very exciting and creative environment of microprocessor design, where she moved from academia, being quite naive and a bit arrogant, to the complexity of the industrial engineering problems. Soon, the author discovered how difficult is it to create a tool that really works in all (not just in most) cases. It takes a short time to implement the core of a tool, but it takes an enormous effort to get such a tool to work for all possible inputs, taking into account the unpredictability of its usage. For ideas that take a few days to conceive and implement, it may take weeks, or even months, to refine them to a point where they provide stable, correct, and user friendly operation.

The other lesson learned was that not always the most sophisticated solution is the best. For example, one of the faster satisfiability checkers, Chaff [14], has a very simple implementation but it is tailored to run well on available microprocessors. It outperforms more complex algorithms designed from the results of long-term research. These observations do not invalidate the work of theoreticians (because without their effort, we would not have today's understanding of the problem), they just confirm that there is a large gap between theoretical results and their application.

This paper consists of two main parts: an overview of the main formal verification methods deployed in the industrial hardware verification, and a close look at our in-house verification tools tailored for the needs of Alpha microprocessor designers. Specifically, we focus on the use of the BOVE verification tool [12] and its features that provide a useful formal verification environment for Alpha designers.

2 Formal Functional Verification

Formal functional verification is the use of formal methods to establish the correctness of a design in much the same way that simulation is used. The difference is that instead of checking the results of a simulation, which produces concrete values, we check the results of a *symbolic* simulation which are equations. For instance, imagine we wish to verify the correctness of a 64-bit adder – a very common and often used device in a microprocessor design. We can simulate the operation of this 64-bit adder design; however, to check all of the possible input combinations requires 2^{128} simulator runs! If this adder is represented in an un-

ambiguous hardware description language, we can derive equations describing the functionality of the design, and prove formally that its output provides the sum of its inputs. The result of such formal verification is equivalent to running 100% of the possible input combinations. Often, it is much cheaper (less time, effort, and computing resources) to use formal verification than simulation. However, formal methods can be used only in places where we have a rigorous description of a model against which (a part of) a design can be compared.

Note, that there are different formal methods, several of which we will summarize, that provide different approaches for obtaining verification results. As the paper progresses, we become more and more specific, finally describing the BOVE equivalence tool and its use in verifying Alpha microprocessor designs.

2.1 Design Flow

Before we start to talk about our verification methods, the reader should have a basic understanding of our design flow, which is shown in a simplified manner in Figure 1. At the top level, the architecture is defined by its instruction set. The Instruction Set Architecture (ISA) is usually specified by several hundred pages of English text, and in some places it is enhanced with pseudo-code or other pseudo-formal descriptions of its functionality. The Alpha Architecture Reference Manual [2], which includes the ISA specification, has about 800 pages. It describes the influence of each operation on the architectural state, and how exceptions and interrupts are treated. Each generation of Alpha microprocessor is obliged to comply with this specification and to preserve backward compatibility of the ISA.

For each microprocessor development project, an executable model (usually in some imperative language like C, or C++) is written for measuring performance of architectural decisions. This model is simulated more than any other model, giving architects confidence in their ideas. New versions of such Alpha models can be produced by augmenting a previous microprocessor generation model with new required features. This model is tested for months and is used throughout the design effort.

Another critical model is the description of the microarchitecture at the register-transfer level (RTL). How closely this model relates to the physical partition of the design differs from project to project. Our company uses its own hierarchical hardware description language, named Merlin. The Merlin RTL model is thoroughly simulated against the higher-level model, and is also subjected to timing analysis and power estimation.

The RTL design is the specification for the transistor-level schematics. To achieve the highest possible performance, the majority of each Alpha design has been fully custom. Only small, non-critical portions of the design are created by synthesis tools. After the transistor-level schematics are completed, these schematics are passed to layout engineers.

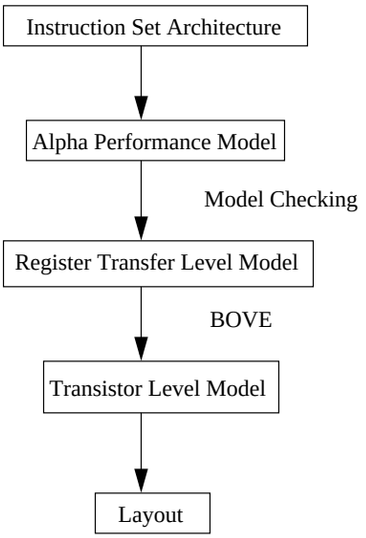


Fig. 1. Design flow and points where formal verification is applied

2.2 Verification Task

The general verification goal is to assure that the result of the design process is correct. In this paper, we consider only functional correctness. To achieve this goal, we need to assure that no logical flaws (bugs) are introduced into a design as it moves through the design flow. In each design stage, the refined design is compared to the result produced by the previous stage. Bugs can be introduced in any of the design phases, and there are different phase-specific methods that help to eliminate them. Some of the validation methods are incomplete, but they are still useful in finding many bugs in the early stages of the design process.

2.3 Formal Verification Techniques

In this section, we briefly describe basic formal verification approaches, starting with high-level verification approaches and concluding with methods applied to the lower-level design artifacts.

Theorem Proving. Let us consider an ideal situation when our ISA is written in a strict mathematical way, and our RTL language has clear semantics. In that case, having the system and its specification formulated by means of logic formulae allows its correctness to be established as a logical proof. Because of the size of industrial hardware designs, this task cannot be done by hand. There are several automated proof checkers for first and higher-order logics available, e.g., HOL [27], PVS [38], Otter [36], and ACL2 [1]. Theorem proving has been

successfully used on number of university projects, but also at AMD [44], Intel [26], and Rockwell-Collins [31].

Unfortunately, we do not have this *ideal* situation. There are no formal specifications available, and verification engineers have to write them by hand, using informal specifications as a guide. It requires a lot of effort, and the availability and willingness of architects to discuss ambiguous specifications. Also, hardware description languages often have constructs that have no clear semantics, and are, therefore, hard to formalize. In better cases, it is possible to translate RTL code into a formal language. This transition needs to be automatic, because designs consist of hundreds thousands of lines of code, and they are frequently changed. The translation also needs to be straightforward enough not to introduce even more bugs. Despite the huge progress in the development of theorem provers, the application of theorem proving to RTL verification is far from automatic. It has been used only in restricted way – to a simple design that was created with the idea of formal methods in mind [29], or restricted to a part of a design like a floating-point arithmetic unit that has historically had a formal (or close to formal) specification [44,26]. Because of the problems mentioned and the unavailability of skilled verification engineers, theorem-proving-based validation covers at most a very small part of the chip design process. In our environment, theorem proving was applied for the verification of network protocols [33].

Model Checking (MC) is an automatic verification technique to prove temporal properties of finite-state systems. The idea of temporal logic is that the truth value of a formula is dynamic with respect to time – it can be true in some state of the model and false in another. Temporal logic allows a user to specify the order of events in time, e.g., "if package A arrives before package B, it departs the queue before B", or "if signal A is set to 1, it will remain set until B gets reset". An efficient algorithm for model checking was introduced independently by Emerson and Clarke [21,16] and Queile and Sifakis [42]. An excellent source for results in the application of model checking is a recent textbook [18]. MC started as a software verification technique, and later, found its way to hardware verification, where it has been successfully used to verify RTL code.

The original MC algorithm was based on the idea of explicit state space exploration; therefore, it could handle systems with only a modest number of states. The big breakthrough for MC was the introduction of BDDs (see Section 2.4) to represent formulas and state sets; this kind of MC is called *symbolic* model checking. Modern symbolic MC systems [47] are able to handle finite state machines with very large state spaces, e.g., up to 2^{200} states. However, this is still a big restriction for industrial-sized hardware. In order to verify a meaningful piece of hardware, tools with vastly more capacity are required.

Although MC is considered to be an automatic verification method, its application involves a lot of engineering. The reason is the limited capacity of available model checkers. There are different ways to reduce the state space, but any reduction that is not done properly may mask errors, or even introduce new errors. In the case when model-checking is used to find bugs instead of proving

their absence, this approach with reduction is considered safe. We can reduce the portions of circuit in a way that does not change the behavior of the circuit with respect to the verified property. Reductions can be done rigorously with the aid of a theorem prover.

Another effort required to use MC is the modeling of the environment in which the hardware unit to be checked operates. Not all inputs to the unit are possible. Finding a violation of a desired property for invalid inputs is irrelevant. Therefore, engineers write *transactors* to model the environment of the unit so as to rule out impossible behaviors. Once, the transactors are written (which is hard work), the model can be further reduced by other methods: constant propagation, removing redundant circuitry, reduction of state-holding elements, etc. Writing properties and decoding failure traces can be a non-trivial task that requires good knowledge of the tools used and of the unit being verified.

The capacity limitation of MC has motivated the search for other approaches to the verification task. For instance, *bounded* model checking [10,9,46] is a result of such effort. A fixed-length prefix of a computation is described by a formula which is passed to a satisfiability checker. The existence of a satisfying assignment indicates a problem within the prefix of the computation. Although incomplete, this method is often fast for discovering bugs, and has been successfully used in our verification methodology [13]. An alternative MC method known as *symbolic trajectory evaluation* (STE) [48,5,28] uses symbolic simulation techniques. A STE logic allows the formulation of simple temporal properties of a transition system. Its advantage is that it is less sensitive to state explosion. Examples of verifications of designs with 5000 or more latches, i.e., systems operating over the state space of 2^{5000} states, have been reported. It has been successfully used by IBM, Motorola, and Intel, to verify transistor-level and RTL designs [51,39]. Our company used STE based on a SAT-solver to prove simple temporal properties of the Alpha memory subsystem [13].

Model checking and theorem proving may be considered complementary technologies. Theorem provers provide formalisms expressive enough to specify an entire processor, but they require manual guidance. The MC verification algorithm is fully automatic, but its application is unfeasible for designs with large state spaces. It seems natural to combine these techniques into a system that uses strengths of both [45,32,3,24]. Currently, such systems are built either on a top of model checkers that use theorem proving to decompose the problem into smaller subproblems, or conversely on a top of a theorem prover that uses model checking to prove subgoals.

Equivalence Checking is a method applied to the following task: given two models of a circuit, prove that they have the same input-output behavior. This task is relevant for the new designs, that we want to verify against their specifications, or to modified designs. The latter case occur often in the later stages of the design project, when designers get feedback from the timing and power estimation tools, and the designs undergo many changes. We may also apply equivalence checking to modified specifications.

Equivalence checking requires different approaches for combinatorial and sequential circuits. A combinatorial circuit is an acyclic circuit without state-holding elements. Each output can be described as a Boolean function of its inputs. The verification task is then reduced to the equivalence of Boolean functions. The basic method used for combinatorial equivalence checking is the construction of a canonical representation of those functions (see Section 2.4). Non-equivalence can be formulated as a satisfiability problem, and it is useful for finding bugs.

Even though most circuits are not combinatorial, in many situations, when the modifications are done within the latch boundaries, or the design methodology preserves state encoding of the machines, the verification problem can be decomposed into many combinatorial equivalence checking tasks. Verity is an example of a tool based on this strategy [50]. However, our methodology, does not bind designers to keep the state encoding. Therefore we have to deal with sequential circuits that are modeled by finite state machines.

Finite state machines considered in this paper are *deterministic Finite State Machines (FSM) of Mealy type* [30]. A FSM is a 6-tuple

$$\langle I, S, \delta, S_0, O, \lambda \rangle$$

where: I is an input alphabet
S is a finite set of states
$\delta : S \times I \mapsto S$ is a transition function
$S_0 \subseteq S$ is a set of initial states
O is an output alphabet, and
$\lambda : S \times I \mapsto O$ is an output function

Nondeterministic FSM can be defined similarly, if we replace the notion of the transition function by a transition relation. Nondeterministic FSMs can be used to model systems that are incompletely specified, or have unpredictable behavior.

Digital systems are modeled by FSMs with binary encoded input and output alphabets. Two FSMs with the same input and output alphabets, and disjoint sets of states are equivalent, if starting from initial states, for any sequence of inputs they give the same sequence of outputs. Equivalence of digital systems modeled by FSMs can be formulated also by means of their product machine. Let

$$F^{(i)} = \langle \{0,1\}^n, S^{(i)}, \delta^{(i)}, S_0^{(i)}, \{0,1\}, \lambda^{(i)} \rangle, \text{for } i \in \{0,1\}$$

are two FSMs, that model a system with n input signals and one output signal. The *Product machine* $F = F^{(0)} \times F^{(1)}$ is an FSM

$$\langle \{0,1\}^n, S, \delta, S_0, \{0,1\}, \lambda \rangle$$

where: $S = S^{(0)} \times S^{(1)}$ is Cartesian product of $S^{(0)}$ and $S^{(1)}$
$\forall s_0 \in S^{(0)}, s_1 \in S^{(1)}, x \in \{0,1\} : \delta((s_0, s_1), x) = [\delta^{(0)}(s_0, x), \delta^{(1)}(s_1, x)]$
$S_0 = S_0^{(0)} \times S_0^{(1)}$
$\forall s_0 \in S^{(0)}, s_1 \in S^{(1)}, x \in \{0,1\} : \lambda((s_0, s_1), x) = (\lambda^{(0)}(s_0, x) \equiv \lambda^{(1)}(s_1, x))$

$F^{(0)}$ and $F^{(1)}$ are equivalent, if their product machine always produces 1 as output. This can be seen as a special case of proving an invariant, which is a property that is expected to hold on all reachable states. There are three operators used in reachability analysis: image, pre-image, and back-image. Let R and S be predicates, each representing a set of states, and δ be a transition function of a system.

$$\begin{aligned} Img(R,\delta) &= S \ \textit{such that}\ S(x') = \exists x \exists y : R(x) \wedge (x' = \delta(x,y)) \\ Preimg(S,\delta) &= R \ \textit{such that}\ R(x) = \exists y : S(\delta(x,y)) \\ Backimg(S,\delta) &= R \ \textit{such that}\ R(x) = \forall y : S(\delta(x,y)) \end{aligned}$$

From now on, we will not distinguish between a set and the predicate that defines it. There are two basic methods for invariant checking, both based on fixpoint computation. *Forward* reachability analysis follows all the possible machine steps from initial states, using the *Img* operator, and checks whether an invariant holds on all reachable states. In contrast, *backward* reachability is based on the other two operators. It explores the state space starting from valid states (states where the invariant holds) or from invalid states (states where the invariant does not hold). In the former, it computes the superset of initial states for which the invariant holds on all computations of fixed length, incrementing this length iteratively until the fix point is reached. In the latter, it tries to prove that none of the states from which the machine could get into an invalid state is an initial state of the machine. Figure 2 shows heavily simplified algorithms for invariant checking. Their efficient implementation has been intensively studied. The result of that research are techniques that involve different tricks with transition function and state set representation, decomposition, and approximation.

2.4 Symbolic Computations

Many of the problems mentioned in the previous sections involve the manipulation of large sets; for example, set of states or set of mismatch cases. A tiny piece of hardware containing one hundred state-holding devices is modeled by a system with 2^{100} states. Any attempt to use an algorithm based on enumeration of this set is a priori going to fail. Fortunately, these sets have often a regular structure, and their characteristic function has manageable size with appropriate representation. *Appropriate* in this context means that there are efficient algorithms for the operations required by the application. In our case, these operations include: quantification, image computation, basic set operations, emptiness check, on-set computation, and Boolean operations. One such representation of Boolean functions, and consequently sets provide Ordered Binary Decision Diagrams. They were first applied by Bryant [4], who recognized their advantage for the verification of combinatorial circuits.

An Ordered Binary Decision Diagram (BDD) over a set of Boolean variables X_n is a directed acyclic graph with the following properties:

Forward Traversal (V, δ, S_0)
{
  $N = R = S_0$
  while $((N \neq \emptyset) \wedge (N \cap \overline{V} = \emptyset))$
  {
    $T = Img(N, \delta)$
    $N = T \cap \overline{R}$
    $R = R \cup T$
  }
  if $(N = \emptyset)$ return 1
  else return 0
}

Backward Traversal 1 (V, δ, S_0)
{
  $R = N = \overline{V}$
  while $(N \neq \emptyset) \wedge (N \cap S_0 = \emptyset)$
  {
    $F = Preimg(N, \delta)$
    $N = F \cap \overline{R}$
    $R = R \cup N$
  }
  if $(N \cap S_0 = \emptyset)$ return 1
  else return 0
}

Backward Traversal 2 (V, δ, S_0)
{
  $R = S$ /* set of all states */
  $F = V$
  while $(R \supset S_0) \wedge (R \neq F)$
  {
    $R = F$
    $F = Backimg(R, \delta)$
  }
  if $(S_0 \not\subset R)$ return 0
  else return 1.
}

Let V be the set of valid states, δ be the transition function, and S_0 be the set of initial states of the FSM to be analyzed. After i iterations of the loop in *Forward Traversal*, T is the set of states reachable in i steps, and N is a subset of T that contains all states reachable in i, but not fewer steps. After i iterations of the *Backward Traversal 1* loop, F is the set of states that can bring machine to an invalid state in i steps, and N is a subset of unexplored states of F, i.e., the states which predecessors have not yet been considered. After i iterations of the *Backward Traversal 2* loop, we know that any computation of length less than i, that starts in R, satisfies the invariant. All procedures return 1, if all reachable states are valid; otherwise they return 0.

Fig. 2. Forward and Backward Traversals

1. Sink nodes are labeled by Boolean constants.
2. Each internal node is labeled by a variable from X_n, and has two successors – one labeled by *low*, the other by *high*.
3. There is a strict order of variables, that is obeyed on all root-to-sink paths.

Any assignment of Boolean values to variables in X defines a root-to-sink path. The label of the sink matches the value of the represented Boolean function for

that assignment. The size of a BDD is defined as the number of its internal nodes. Any BDD can be reduced to its minimal form in time linear with respect to its size. This form is a canonical form, i.e., it is unique for each function and fixed variable order. The main advantage of BDDs is the efficiency of their manipulation. The following operations can be performed in polynomial time with respect to the size of the manipulated BDDs:

- Boolean, and set-theoretic operations;
- cofactors, i.e., restriction of the function by fixing a value of a variable;
- evaluation of the function for a given input assignment;
- satisfiability, tautology test, and equivalence; and
- existential and universal quantification over a constant number of variables.

Significant engineering effort has been spent in tuning the performance of BDD algorithms [35], and today, there are a number of high-quality, publicly available BDD-packages [15]. Most microprocessor design companies have their own in-house BDD packages that works as the core engine of their equivalence checking software.

The first application of BDDs to sequential verification was McMillan's approach to symbolic reachability analysis and model checking [47,8]. The practical advantage of symbolic computation is that it provides the set of all solutions at once. Symbolic computations unify representation of sets, relations, and functions, which is useful, in particular, for fixpoint computation based on image operators.

The main concern about the use of BDDs was their sensitivity to variable ordering that may make an exponential difference in their size. This is a serious drawback as it requires an additional engineering effort even for those functions that may be represented efficiently. This unpleasant feature forces verification engineers to become experts not only in design but also in BDD techniques.

When it became clear that dealing with designs that had more than a couple of hundreds of state-holding devices would be beyond the capacity of BDD-based verification tools, researchers started experimenting with new ideas. The satisfiability problem [41] has been studied for decades as a basic NP-complete problem. It is easy to formulate, but hard to solve, and it remains a big challenge for researchers. Recently, several research groups reported the results of their experiments with SAT-based verification methods [7,25,46]. There are several publicly available satisfiability checkers ([34,14]), and a ranking of the top 10 SAT solvers can be found on *http://www.lri.fr/~simon/satex/satex.php3*. The development of many of them was motivated by the needs of industry. Most of them are based on the Davis-Putnam procedure [20] enhanced with learning methods, advanced backtracking techniques, and sophisticated branching heuristics. An interesting approach to the SAT problem is the Stålmarc's method [49]. As a complementary technique to BDDs, SAT has proved to be particularly effective in the context of bug-finding methods, like bounded model checking, or STE, but there is no clear winner with respect to all methods. A combination of both seems to be a good solution [52,43]. In our environment, SAT-based methods

reduced the time spend waiting for failure traces from days to minutes, or even seconds. This allows fast tuning of transactors and finds many of the bugs. The BDD-based methods have been shown to be more useful in the later stages after the most bugs have been found and validity needs to be proved.

3 Sequential Equivalence Checking

In the last decade, the Alpha [22,23] has often been considered to be the fastest available microprocessor. This performance advantage was achieved by advanced architecture and full custom circuit design. Designers are capable of much higher optimization by unusual creative solutions than any state-of-the-art synthesis tool. However, humans make mistakes, and no matter how smart and experienced circuit designers are, their designs require validation; the transistor-level design needs to be checked against RTL model.

The traditional way to check that transistor-level design is functionally equivalent to RTL code is through simulation. Generally, there is a suite of regression tests on which the RTL model is run in parallel with a model extracted from its transistor-level design. In each cycle, signal values for both models are computed and compared, and an error is reported in the case of mismatch. The capacity and speed of our simulator is good enough to perform tests on a full chip. Unfortunately, for obvious reasons, there is no way to exhaustively simulate an entire microprocessor, and it is also hard to estimate how good the test coverage is. When a mismatch does occur, it can be difficult to locate its source. It is expensive to allocate designer time to create tests for "small" pieces of their designs. There is a need for a tool that can do both – debugging and verification.

There are several vendors that offer equivalence checkers, but it is hard to find one that would satisfy all our needs. Our in-house BOolean VErifier (BOVE) is tailored to our methodology and designers' style. It runs on our Alpha platforms, and users get full support and fast response to their requests. Each project often require modifications to the methodology that are promptly implemented.

Initially, BOVE was developed primarily for debugging, but the tool went through substantial modifications to extend its functionality and user friendliness. It has become an indispensable part of our validation methodology. The first application of the tool was performed by its developers on a part of Alpha 21264 design [12]. During the development of the Alpha 21364, it was run mostly by designers and architects, with occasional assistance of the developers. The methodology flow of Alpha 21464 project expected architects and designers to use the tool for the entire design.

BOVE is used in bottom-up manner: each designer applies it repeatedly on the transistor-level schematic under development. In the process of debugging, a designer creates the mapping of the equivalent signals, and specifies don't cares and preconditions, that are necessary to prove the correctness. Once a schematic has been verified, the verification artifacts are stored to be used in nightly regression runs that check all subsequent modifications of the design.

They can also be reused later in the verification task of the yet larger unit containing several schematics.

3.1 Verification Scheme

BOVE is an equivalence checker for gate-level networks. It performs all computations symbolically using BDDs [11,15]. Translators from RTL code and from transistor-level networks to equivalent gate-level networks, allow us to use BOVE for different tasks: checking correctness of modified RTL, modified schematics, or new schematics against RTL. We will focus on the latter task in this paper. We will refer to the network that is to be verified as the *implementation*, while the network against which we verify the implementation is called the *specification*.

A problem instance for BOVE consists of a specification, an implementation, a mapping between inputs and the outputs of both modules, and a mapping of additional internal compare points that are assumed to be equivalent. The formulation of the goal is: assuming equivalence of the inputs of the modules as described by the input mapping, prove equivalence of all mapped nodes (internal compare points and outputs). Compare points allow us to break the task into smaller subtasks. BOVE keeps track of the individual subtasks that are successfully solved and those that need to be done, and at the end of each session reports its progress. For different reasons described in the following sections, BOVE may fail to prove some of these equivalences, falsely reporting mismatches. We made a substantial effort to eliminate these false negatives. The main rule BOVE is built upon is that it never returns a false positive, i.e., if the tool successfully finishes its task and reports that all compare points are equivalent, the models being compared are equivalent, given that the inputs satisfy the relation described by the mapping, and the machines are resetable (see next section).

3.2 The Equivalence Problem on Slices

The mappings of compare points implicitly specify the set of separate equivalence problems. To prove the equivalence of two nodes, BOVE analyzes the fan-in of the nodes (either back to primary inputs or to compare points), which we call a *slice*. The slice inputs are assumed to satisfy the given mappings. BOVE tries to prove that the nodes as functions of slice inputs are equivalent. For most of the signals, we do not need to investigate the entire fan-in back to the primary inputs. BOVE attempts to find the smallest slice suitable for proving the equivalence of the two signals. Then it analyzes the slice to determine what algorithm can be used for the comparison.

A slice that does not contain state elements or loops, is called *combinatorial*. The algorithm for combinatorial equivalence is the simplest: we build the BDD representation for both nodes, and because of the canonicity of BDDs and the use of hash tables that assure that each unique BDD is stored only once, the equivalence check is finally reduced to a pointer comparison.

In a design with an aggressive clock speed, signals are latched frequently. This results in slices that have latches but no loops, called *acyclic* or *pipelined*

slices. The equivalence checking of pipelined slices is based on the idea of looking at the output signals as functions of inputs indexed by the delays caused by the latches. This idea allows for the transformation of the problem into a combinatorial problem. The values of an input for different delays are considered to be independent. In reality, they are consequent output values of some circuitry; this might cause a false mismatch report. BOVE uses a data structure for the representation of the outputs called Timed-Ternary BDDs; this kind of BDDs are unique to BOVE [12]. Despite the fact that Timed-Ternary BDDs do not provide a canonical representation, it has worked very well on our designs. The availability of equivalence checking for pipelined designs gives designers freedom to depart from simple RTL timing, and to perform the timing optimizations that are necessary to achieve the highest performance.

The most general equivalence checking algorithm implemented in BOVE can handle slices with state holding devices and loops. Sequential systems of this type are modeled by FSMs, with input set $\{0,1\}^n$ and output set $\{0,1\}$. Storage elements that encode the state of a machine take values from the set $\{0,1,X\}$, where X has the meaning of *unknown*. The extension of Boolean logic to ternary is implied by the extension of the AND and NOT operations: $\mathrm{AND}(X,0) = \mathrm{AND}(0,X) = 0$, $\mathrm{AND}(X,1) = \mathrm{AND}(1,X) = X$, $\mathrm{NOT}(X) = X$, and storage devices propagate X. Computation with ternary logic is implemented using pairs of BDDs.

Our FSMs are completely specified, i.e., both transition and output functions are defined for any combination of inputs and internal states. Unlike in the most commercial equivalence checkers, we do not require any correspondence between storage elements; designers are free to change the state encoding of the machines! A FSM is called *resetable*, if there is a sequence of inputs that brings the machine from the state where all storage elements are set to X to a known binary state. Resetability is out of the scope of BOVE, but it is assumed by BOVE, and we use COSMOS [6] to ensure that a FSM is resetable.

Our goal is to prove that once two machines under consideration are successfully initialized (reset), they will always give the same output given the same inputs. The state sets of the two machines are disjoint. BOVE builds the product machine of the specification and implementation machines as described in Section 2.3. The invariant we want to establish is that the output of the product machine is true on the care set specified by architect. To prove the invariant, BOVE uses both forward and backward traversal. However, in most cases, the set of initial states is not known; therefore, backward traversal is used more frequently. The FSM equivalence checking in BOVE [12] was extended by integrating a traversal package developed at the University of Torino [37] which implements both forward and backward traversals. The tool is integrated as a separate engine that receives the BDD representations of the next-state functions from BOVE, performs an equivalence check, and in the case of an invariant violation, generates an initial state and a sequence of inputs that exhibit an invalid state. Assuming resetability of both machines, if BOVE reports a match

for the product machine, the product machine can be reset to a valid state and there is no sequence of inputs that cause the machine to enter an invalid state.

Theoretically, FSM-equivalence checking could satisfy all our verification needs. However, in reality, all three equivalence checking algorithms: for combinatorial, pipelined, and FSM slices, play an important role in the BOVE compare procedure. The combinatorial algorithm is simple and BOVE has good heuristics for combinatorial slice discovery. Pipelined slices with a large number of latches would normally cause a state explosion of the FSM verification algorithm, but easily pass through BOVE's pipelined verification algorithm. Finally, the FSM slice algorithm is critical for verifying slices with feedbacks and devices that generate Xs.

3.3 Preconditions, Don't Cares, and Precharge Mapping

Because of the complexity of modern designs and the restricted capacity of our tools, validation is applied on designs piece by piece. This is a non-trivial task, because for verification purposes a design cannot be decomposed into completely independent parts. Even though two signals are equivalent in the context of the whole design, they may not be equivalent in the context of one schematic, unless we place restrictions on the range of input values for this schematic. BOVE allows users to specify *preconditions* that describe constraints on the values of inputs. A typical precondition is the mutual exclusion of some inputs, but any FSM may serve as a precondition.

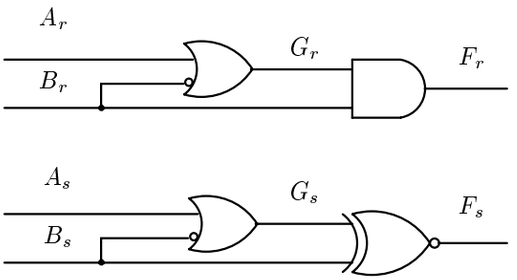
Architects writing RTL code describe required functionality, but at the same time, they leave designers some freedom for their implementation. This is expressed as *don't care conditions*. For instance, a signal may take any value if the clock is high or that some combination of inputs can never occur. The interpretation of don't care conditions is different for slice inputs than for outputs. If there is a don't care condition associated with a slice output, the output equivalence needs to be proved for the inputs that violate this condition only. In turn, if a don't care condition associated with a slice input is satisfied, its mapping is disregarded; this assures soundness of BOVE results.

Another feature that reflects the gap between specification and implementation is an extension of the mapping of compare points to *precharge mapping*. Precharge mapping expresses a more complex relation between the static specification and the dynamic implementation of signals; for instance, a schematic signal can be precharged high in one clock phase and equivalent to the RTL signal in a subsequent phase. Based on our methodology, precharged mapping can be derived from the names of the signals. Like equivalence mapping, precharge mapping is used as an assumption for slice inputs, but must be proved for slice outputs.

3.4 Features for False Mismatch Elimination

The main reason for false mismatches is the restriction of the context in which we attempt to prove the equivalence of signals. In the previous section, we described

how preconditions associated with the inputs of a piece of the design can help avoid false mismatches. In other cases, a slice that is created automatically by BOVE is too small to capture the signal equivalence (see Figure 3). We developed different *heuristics for finding appropriate slice boundaries*. Users have also some impact on slice expansion by temporarily dissociating internal compare points.



Assume that a user specified following mappings: $A_r \equiv A_s$, $B_r \equiv B_s$, $G_r \equiv G_s$, $F_r \equiv F_s$. Let us assume we are trying to prove the last equivalence. We create a slice for F_r and F_s by expansion of their fan-in to the first compare points that are defined by the mappings, i.e., B_r, B_s, G_r, and G_s become the slice boundaries. Since, $F_r = \text{AND}(G_r, B_r)$ and $F_s = \text{XNOR}(G_s, B_s)$, we will get a mismatch. The mismatch analysis will show that the mismatch occur when $G_r = G_s = B_r = B_s = 0$. Looking deeper into the circuit, we can see that this case cannot ever happen, because $(B_r = 0) \Rightarrow (G_r = 1)$ and similarly $(B_s = 0) \Rightarrow (G_s = 1)$.

Fig. 3. False mismatch resulting from an inadequate slice

The frequency with which compare points appear in a design to be verified is very important as it influences the minimal size of a slice. The distance between compare points defines the smallest slice. In cases when the user does not provide enough compare points, BOVE can use its *automatic mapping* routine, that is based on pseudo-random simulation to find candidates for compare points. The accuracy of the routine depends on the number of pseudo-random test runs. The compare points found are not necessarily equivalent and it is up to the user to remove those points that cause mismatches. The mismatch report on those nodes cannot be ignored, because their equivalence is used as assumptions in subsequent equivalence proofs.

In contrast to false mismatches resulting from slices that are too small, slices that are too large are hard to debug and sometimes even too hard to verify. In an effort to eliminate equivalence checks that fail due to the capacity of the tool, we implemented *heuristics for variable orders* that help keep BDD sizes manageable.

Some of the false mismatches can be avoided by keeping track of equivalences that occur just within an RTL description or within a schematic. Since the equivalences defined by mappings between the RTL and implementation are

assumed to be valid for slice inputs, we can propagate all equivalence relations from specification to implementation and back using these mappings. The *propagation of the slice input equivalences* is demonstrated with the example shown in Table 1. The scheme can be more complicated if we allow more than one node to be mapped to several nodes and if precharge mapping is involved.

Table 1. Example of slice input propagation

Mapping	RTL Equivalences	Schematic Equivalences
$m1$: $R1 \equiv S1$		
$m2$: $R2 \equiv S2$	$r1 : \neg R2 \equiv R3 \equiv R4$	$s1 : S1 \equiv \neg S2$
$m3$: $R4 \equiv S4$	$r2 : R5 \equiv \neg R6$	$s2 : S3 \equiv \neg S4 \equiv S5$
$m4$: $R5 \equiv \neg S5$		

The first column describes the mapping between specification and implementation. The second column contains equivalences within just the specification, and the third column lists only the equivalences within implementation. $A \equiv B \equiv C$ is an abbreviation of $(A \equiv B) \wedge (B \equiv C)$. Given this information, we are able to deduce following information:

$$\begin{array}{l} m1, s1, m2\text{imply } R1 \equiv \neg R2 \\ m2, r1, m3\text{imply } S2 \equiv \neg S4 \\ m3, s2, m4\text{imply } R4 \equiv R5 \\ m4, r2, m5\text{imply } S5 \equiv S6 \end{array}$$

Consequently,

$$R1 \equiv \neg R2 \equiv R3 \equiv R4 \equiv R5 \equiv \neg R6$$

and

$$S1 \equiv \neg S2 \equiv \neg S3 \equiv S4 \equiv \neg S5 \equiv \neg S6$$

and we are able to prove relations like $\text{NAND}(R1, Y) \equiv \text{OR}(S5, Y)$; this could not be done without slice input propagation.

3.5 Debugging Features

Simulators that are used to validate chip-level schematics are usually run on the big parts of a design, the full chip, or even the entire system. Tests are either created by large verification teams or they are passed along from previous projects. We have both focused and random tests. However, these tests are practically useless for designers that work on their relatively small pieces of the design. There is no time in their tight schedule for designing their own test sets that would satisfactorily cover the schematics on which they work. BOVE takes this worry away from the designers by providing 100% coverage of the design. A designer needs to have a global understanding of the tool only, and is not required to understand the verification algorithms used.

Since BOVE is often used for debugging - it is meant to run quickly on small pieces, and to provide an indication of the problem if a mismatch occurs. The main features that helps a user to find a bug is *mismatch analysis*, which varies with the comparison algorithm used. In the case of the combinatorial algorithm, users receive a condensed table of all cases where an implementation differs from its specification. In the case of a pipelined slice, mismatches for the underlying combinatorial circuit are given. Mismatches of FSMs are a bit more complicated because they are presented with witnesses of the mismatches; such witnesses describe an initial state and a sequence of inputs that causes the machines to produce a different output. By using its built-in simulator, BOVE can create a visual representation of an error trace containing inputs, outputs, and signals identified by a user. The error trace is presented either as a waveform or as list of values for each time step. In addition to mismatch analysis, BOVE is able to *detect simple errors*, like a missing invertor, and can make suggestions of how to correct it. If the mismatch is a potential false mismatch due to an inappropriate choice of slice, BOVE reports the slice input misalignment. In any event, a user may always ask for slice analysis, that shows different slice statistics.

3.6 Conclusion

Although simulation remains the dominate functional verification method, the use of formal methods, and particularly equivalence checking is a critical part of the Alpha microprocessor development process. Our in-house equivalence checker BOVE has replaced simulation as a means to verify transistor-level schematics. As complexity increases, the performance advantage offered by BOVE is critical to our validation needs. Many of the nice features in BOVE were motivated by requests from users. Without the tight interaction between developers and designers, BOVE would not be what it is now. We see the use of other formal methods by the Alpha development team, but in a less systematic manner.

Acknowledgment

BOVE is a long-term project that has been supported by Gabriel Bischoff, Karl Brace, and for a three-year period by Samir Jain. My two and half years with the BOVE team was an excellent experience in tool construction and a technology transfer environment. It taught me the skills that I would not be able to gain anywhere in academia.

All team members have developed, extended, and enhanced BOVE, and all members have provided user support. Some of BOVE's features were the exlusive responsibility of a specific member: precharge mapping was implemented by Karl Brace, while the integration of the FSM traversal engine was performed by the author and Gabriel Bischoff, with the help of Gianpiero Cabodi and Stefano Quer.

References

1. Kaufmann, M., Manolios, P., Moore, J.S.: Computer-Aided Reasoning. An Approach. Kluwer Academic Publishers, 2000.
   http://www.cs.utexas.edu/users/moore/acl2.
2. Alpha Architecture Reference Manual. Digital Press 1998. ISBN 1-55558-202-8.
3. Aagaard, M.D., Jones, R.B., Seger, C.-J.H.: Lifted-FL: A Pragmatic Implementation of Combined Model Checking and Theorem Proving. Proc. of *Theorem Proving in Higher Order Logics 1999*, LNCS 1690, pp.323-340, 1999.
4. Bryant, R.E.: Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35, pp. 677-691, 1986.
5. Bryant, R.E., Beatty, D.L., Seger, C.H.: Formal Hardware Verification by Symbolic Trajectory evaluation. Proc. of *Design Automation Conference*, 1991.
6. Bryant. R.E., Beatty, D., Brace, K.S., Cho, K., Sheffler, T.: COSMOS: A Compiled Simulator for MOS Circuits. IEEE Proc. of *Design Automation Conference*, 1987.
7. Bjesse, P., Claessen, K.: SAT-Based Verification without State Space Traversal. Proc. of *Formal Methods in Computer-Aided Design 2000*, LNCS 1954, pp. 373-389, 2000.
8. Burch, J.R., Clarke, E.M., McMilla, K.L., Dill, D.L., Hwang,L.J.: Symbolic Model Checking 10^{20} States and Beyond. Information and Computation 98(2), pp. 142-170.
9. Bierre, A., Clarke, E.M., Raimi, R., Zhu, Y.: Verifying Safety Properties of a PowerPC^{TM} Microprocessor Using Symbolic Model Checking Without BDDs. Proc. of *Computer Aided Verification*, 1999.
10. Bierre, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic Model Checking Without BDDs. Proc. of Conference on Tools and Algorithms for the Construction and Analysis of Systems, 1999.
11. Brace, K.S., Rudell, R.L., Bryant, R.E.: Efficient Implementation of a BDD Package. IEEE Proc. of *Design Automation Conference*, pp. 40-45, 1990.
12. Bischoff, G., Brace, K., Jain, S., Razdan, R.: Formal Implementation Verification of the Bus Interface Unit for the Alpha 21264 Microprocessor. IEEE Proc. of *International Conference on Computer Design: VLSI in Computers and Processors*. pp. 16-24, 1997.
13. Bjesse, P., Leonard, T., Mokkedem, A.: Finding Bugs in an Alpha Microprocessor Using Satisfiability Solvers. In Proc. *Computer Aided Verification*. LNCS 2102, pp.454-464, 2001.
14. Moskewicz, M.W., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. IEEE Proc. of *Design Automation Conference*, 2001.
15. Somenzi, F.: CUDD *ftp://vlsi.colorado.edu/pub*, 1996.
16. Clarke, E.M., Emerson, E.A.: Design an Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. Proc. of Workshop *Logic of Programs*, LNCS 131, 1981.
17. Cabodi, G., Camurati, P., Quer, S.: Symbolic Exploration of Large Circuits with Enhanced Forward/Backward Traversals. Proc. of *European Design Automation Conference*, 1994.
18. Clarke, E.M., Grumberg, O., Doron, A.P.: Model Checking. The MIT Press, 1999.
19. Coudert, O., Madre, J.: A Unified Framework for the Formal Verification of Sequential Circuits. IEEE Proc. of *International Conference on Computer Aided Design*, pp.126-129, 1990.

20. Davis, M., Putnam, H.: A Computing Procedure for Quantification Theory. *J. of ACM*, Vol 7, pp.201-215, 1960.
21. Emerson, E.A.: Branching Time Temporal Logicand the Design of Correct concurrent Progams. PhD thesis, Harvard University, 1981.
22. Bernschneider, B.J., Sungho Park, Allmon, R., Anderson, W., Arneborn, M., Jangho Cho, Changjun Ghoi, Clouser, J., Sangok Han, Hokinson, R., Gyeoocheol Hwang, Daesuk Jung, Jaeyoon Kim, Krause, J., Kwack, J., Meier, S., Yongsik Seok, Thierauf, S., Zhou, C.: A 1 GHz Alpha Microprocessor. IEEE Proc. of *International Solid-State Circuits Conference*, pp. 86-87, 2000.
23. Jain, A., Anderson, W., Benninghoff, T., Bertucci, D., Braganza, M., Burnette, J., Chang, T., Eble, J., Faber, R., Gowda, D., Grodstein, J., Hess, G., Kowaleski, J., Kumar, A., Miller, B., Paul, P., Pickholz, J., Russell, S., Shen, M., Truex, T., Vardharajan, A., Xanthopoulos, D., Zou, T.: A 1.2 GHz Alpha Microprocessor with 44.8 GB/sec of chip bandwith. IEEE Proc. of *International Solid-State Circuits Conference*. pp. 240-241, 2001.
24. Gordon, M.J.C.: Reachability Programming in HOL98 Using BDDs. Proc. of *Theorem Proving in Higher Order Logics 2000*, LNCS 1869, pp. 179-196, 2000.
25. Gupta, A., Yang, Z., Ashar, P., Gupta, A.: SAT-Based Image Computation with Application in Reachability Analysis. Proc. of *Formal Methods in Computer-Aided Design 2000*, LNCS 1954, pp. 355-371, 2000.
26. Harrison, J.: Formal Verification of Floating Point Trigonometric Functions. Proc. of *Formal Methods in Computer-Aided Design 2000*, LNCS 1954, pp. 217-233, 2000.
27. Gordon, M.J.C., Melham, T.F.: Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic. Cambridge University Press,1993.
28. Hazelhurst, s., Seger, C.H.: Symbolic Trajectory Evaluation. LNCS 1997. State-of-the-art Survey.
29. Hunt, W.A.,Brock, B.C.: The Dual-Eval Hardware Description Language and Its Use in the Formal Specification and Verification of the FM9001 Microprocessor. *J. Formal Methods in System Design* 11 (1), pp. 71-104, 1997.
30. Hachtel, G.D., Somenzi, F.: Logic Synthesis and Verification Algorithms. Kluwer Academic Publishers, 1998.
31. Widing, M., Greve, D., Hardin, D.: Efficient Simulation of Formal Processor Models. *Formal Methods in System Design. to appear.*
32. McMillan, K.: A Methodology for Hardware Verification Using Compositional Model Cheking. Technical Report, Cadence Berkley Labs, April 1999. *http://www-cad.eecs.berkeley.edu/~kenmcmil.*
33. Mokkedem, A., Leonard, T.: Formal Verification of the Alpha 21364 Network Protocol. Proc. of *Theorem Proving in Higher Order Logics 2000*, LNCS 1869, pp.443-461, 2000.
34. Marques-Silva, J.P., Sakallah, K.A.: Grasp: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5), pp. 506-521, 1999.
35. Meinel, Ch., Theobald, T.: Algorithms and Data Structures in VLSI Design. Springer, 1998.
36. McCune, W.: Otter 3.0 reference manual and guide. Tech. Report ANL94/6, Argonne National Laboratory, Argone, IL, 1994. *http://www.mcs.anl.gov/AR/otter.*
37. Cabodi, G., Quer, S.: PdTRAV. *http://www.polito.it/~quer/software/tools.htm.*
38. Owre, S., Rushby, J.M., Shankar, N.: PVS: A Prototype Verification System. Proc. of *International Conference on Automated Deduction*, Lecture Notes in Artificial Intelligence 607, pp. 748-752, 1992.
39. O'Leary, J., Zhao, X., Gerth, R., Seger, C.H.: Formally Verfying IEEE Compliance of Floating-Point Hardware. *Intel Technical Journal*, Q1:147-190, 1999.

40. Pixley, C.: A Computational Theory and Implementation of Sequential Hardware Equivalence. *Workshop on Computer Aided Verification*, Rutgers University, 1990.
41. Papadimitriou, C.H.: Computational Complexity. Addison Wesley 1994.
42. Queile, J.P., Sifakis, J.: Specification and Verification of Concurrent Systems in CESAR. Proc. of *International Symposium on Programming*, LNCS 137, pp.337-351, 1982.
43. Paruthi, V., Kuehlmann, A.: Equivalence Checking Using a Structural SAT-Solver, BDDs, and Simulation. IEEE Proc. of *International Conference on Computer Design*, 2000.
44. Russinoff, D.M.: A case Study in Formal Verification of Register-Transfer Logic with ACL2: The Floating Point Adder of the AMD Athlon Processor. Proc. of *Formal Methods in Computer-Aided Design 2000*, LNCS 1954, pp. 3-36, 2000.
45. Rajan, S., Shankar N., Srivas, M.K.: An Integration of Model Checking with Automated Proof Checking. Proc. of *Computer-Aided Verification 1995*, LNCS 939, pp.84-97, 1995.
46. Strichman, O.: Tuning SAT Checkers for Bounded Model-Checking. Proc. of *Computer Aided Verification*, LNCS 1855, pp.480-494, 2000.
47. McMillan, K.L.: Symbolic Model Checking: An Approach to the state explosion Problem. Kluwer Academic, 1993.
48. Seger, C.H., Joyce, J.J.: A Mathematically Precise Two-Level Formal Hardware Verification Methodology. *Technical Report TR-92-34*, University of British Columbia,1992.
49. Sheeran, M., Stålmarck, G.: A Tutorial on Stålmarck's Proof Procedure for Propositional Logic. *Formal Methods in System Design*, 16, pp. 23-58, 2000.
50. Kuehlmann, A., Srinivasan, A., LaPotin, D.P.: VERITY - a Formal Verification Program for Custom CMOS Circuits. *IBM Journal of Research and Development*, 1/2, Vol. 39, pp.149-165, 1995.
51. Wang, L., Abadir, M., Krishnamurthy, N.: Automatic Generation of Assertions for Formal Verification of PowerPC$^{\text{TM}}$ Microprocessor Arrays Using Symbolic Trajectory Evaluation. Proc. of *Design Automation Conference*, pp.534-537, ACM Press, 1998.
52. Williams, P.F., Biere A., Clarke, E.M., Gupta, A.: Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. Proc. of *Computer Aided Verification*, LNCS 1855, pp.124-138, 2000.
53. Zhang, H.: SATO: An Efficient Propositional Prover. Proc. of *CADE*, LNCS 1249, pp. 272-275, 1997.

How Can Computer Science Contribute to Knowledge Discovery?*

Osamu Watanabe

Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology
watanabe@is.titech.ac.jp

Abstract. *Knowledge discovery*, that is, to analyze a given massive data set and derive or discover some knowledge from it, has been becoming a quite important subject in several fields including computer science. Good softwares have been demanded for various knowledge discovery tasks. For such softwares, we often need to develop efficient algorithms for handling huge data sets. *Random sampling* is one of the important algorithmic methods for processing huge data sets. In this paper, we explain some random sampling techniques for speeding up learning algorithms and making them applicable to large data sets [15,16,4,3]. We also show some algorithms obtained by using these techniques.

1 Introduction

For knowledge discovery, or more specifically, for a certain kind of data mining task, it would be quite helpful if we can use learning algorithms on a very large data set. In this paper, I explain some random sampling techniques we have developed for scaling up learning algorithms. But before going into a technical discussion, let us see in general what is expected to us computer scientists in knowledge discovery, and locate our problem in there.

For investigating various problems for knowledge discovery, we had in Japan a three-year research project "Discovery Science" (chair *Setsuo Arikawa*, Kyushu Univ.) from April 1998 to March 2001, sponsored by the Ministry of Education, Science, Sports and Culture. This is a quite large project involving many computer scientists and researchers in related fields, from philosophers and statisticians to scientists struggling with actual data. As a member of this project, I have learned many aspects of knowledge discovery. What I will explain below is my personal view on knowledge discovery that I have developed through my experience. Due to the space limit, I cannot explain examples in detail; I cannot even cite and list related papers either. Please refer to a coming book reporting our achievements in the Discovery Science Project that will be published from Springer. (On the other hand, for those explained in the following sections, please refer [15,16,4,3] as their original sources.)

* A part of this work is supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas (Discovery Science), 1998−2001.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 136–151, 2001.

The ultimate goal of computer science is to provide good computer softwares (sometimes, computer systems) that would help activities of human beings. This is also true for knowledge discovery. We want good softwares helping us to analyze "complex data" and discover some "useful rules" hidden in the data. But our approach may differ depending on the type of the "complexity" of data we are given and also the type of the "usefulness" of a rule we are aiming for, and roughly speaking, there are two basic approaches.

The first approach is to develop (semi) automatic discovery systems. This is the case where a given data set consists of a large number of pieces of information of the same type and it is not required to obtain the very best rule explaining the data.

For example, *K. Yada* etal used several data mining tools and analyzed a huge sales data of a big drugstore chain to find out a purchase pattern of a "loyal customer", a customer who will become profitable to the stores. Here it may not be so important to obtain the best rule for this loyal customer prediction. As pointed out by *Yada* etal, a simple rule would be rather useful for business so long as it has a reasonable prediction power. An important and nontrivial task that they did was to design an efficient and robust software that converts actual purchase records to a data set applicable to data mining tools. In general, it is an interesting subject of software engineering to design such a converting software systematically. On the other hand, once we have a nice and clean data set from the actual data, we could use various learning algorithms. Note that these learning algorithms should work very efficiently and should be applicable to a very large data set. Here is the point where we can use random sampling techniques explained in the following sections.

On the other hand, it is sometimes the case that the complexity of data is due to not only its volume. Or we cannot assume any similarity in data. A typical example is a DNA sequence. Suppose that we have a set of complete DNA sequences of some single person. Certainly, the amount of information is huge, but each piece of information does not have the same role. In such a case, we cannot hope for any automatic discovery system. Then we aim for developing discovery systems that work with human intervention. This the second approach.

For example, *R. Honda* etal tried to develop a software that processes satellite images of lunar and finds out craters. They first tried a semi automatic software by using an unsupervised learning algorithm, Kohonen's self organizing map. But it turned out that the level of noise differs a lot depending on images. That is, the data is complex and not of the same type. Thus they changed a strategy and aimed for a software that works in collaboration with a human expert; then a quite successful software was obtained.

In the extreme case, the role of a software is to help human experts to make their discovery. This applies to the case where we want to find the very best rule from a limited number of examples. Note that even if the number of examples is limited, each example is complex, consisting of a large amount of data like the DNA sequence of one person. Thus, an efficient software that filters out and suggests important points would be quite useful. *Y. Ohsawa* introduced the

notion of "key graph" to state potential motivations or causes of some events. He developed a software that extracts a key graph from data and applied it for the history of earthquakes to predict some risky areas. *H. Tsukimoto* etal made a software that processes f-MRI brain images and finds out activated areas in brain. By using his algorithm, it is possible to point out some candidate areas of activation from a very fuzzy original image data. Learning algorithms are also useful to filter out unnecessary attributes. *O. Maruyama*, *S. Miyano*, etal proposed a software that, using some learning algorithm, not only filters out irrelevant attributes but also creates a "view", a combination of attributes, that gives a specific way of understanding a given data. Visualization is also useful for human experts.

So far, we have been considering software developments. But techniques or methods developed in computer science themselves could be also useful for scientific discovery. *H. Imai* etal used several data compression algorithms to study the redundancy or incompressibility of DNA sequences, which may lead, in future, some important scientific discovery on DNA sequences. *S. Kurohashi* etal used a digitalized Japanese dictionary to study the Japanese noun phrase "N_1 *no* N_2". Though "*no*" in Japanese roughly corresponds to "of" in English, its semantic role had not been clearly explained among Japanese language scholars. By computer aided search through some digitalized Japanese dictionary, *S. Kurohashi* etal discovered that the semantic role of "*no*" (of certain types) can be found in the definition statement of the noun N_2 in the dictionary. This may be a type of discovery that researchers other than computer scientists could not think of.

Now as we have seen, there are various types of interesting subjects from knowledge discovery. But in the following, we will focus on the case where we need to handle a large number of examples of the same type, and explain some sampling techniques that would be useful for such a case.

2 Sequential Sampling

First let us consider a very simple task of estimating the proportion of instances with a certain property in a given data set. More specifically, we consider an input data set D containing a huge amount of instances and a Boolean function B defined on instances in D. Then our problem is to estimate the proportion of instances x in D such that $B(x) = 1$ holds. (Let us assume that D is a multiset; that is, D may contain the same object more than once.) Clearly, the value p_B is obtained by counting the number of instances x in D such that $B(x) = 1$. But since we consider the situation where D is *huge*, it is impractical to go through all instances of D for computing p_B. A natural strategy that we can take in such a situation is random sampling. That is, we pick up some elements of D randomly and estimate the average value of B on these selected examples.

If we were asked the precise value of p_B, then trivially we would have to go through the whole data set D. Here consider the situation where we only need to get p_B "approximately". That is, it is enough to compute p_B within

a certain error bound required in each context. Also due to the "randomness nature", we cannot always obtain a desired answer. Thus, we must be satisfied if our sampling algorithm yields a good approximation of p_B with "reasonable probability". We discuss this type of estimation problem. That is, we consider the following problem.

Problem 1. *For given $\delta > 0$ and ε, $0 < \varepsilon < 1$, obtain an estimation $\widetilde{p_B}$ of p_B satisfying the following. (Here the probability is taken over the randomness of the sampling algorithm being used.)*

$$\Pr[\ |\widetilde{p_B} - p_B| \leq \varepsilon \cdot p_B\ ]\ >\ 1 - \delta. \tag{1}$$

This problem can be solved by the following simple sampling algorithm: Choose n instances from D uniformly at random, count the number m of instances x with $B(x) = 1$, and output the ratio m/n as $\widetilde{p_B}$. Here *sample size* n is a key factor for sampling, and for determining appropriate sample size, so called concentration bounds or large deviation bounds have been used. (see, e.g., [19]). For example, the Chernoff bound gives the following bound for n.

Theorem 1. *For any $\delta > 0$ and ε, $0 < \varepsilon < 1$, if the sample size n satisfies the following inequality, then the output of the simple sampling algorithm mentioned above satisfies (1).*

$$n\ >\ \frac{3}{\varepsilon^2 p_B} \ln\left(\frac{2}{\delta}\right).$$

Notice, however, that this bound is hard to use in practice, because for estimating an appropriate sample size n, we need to know p_B, which is what we want to estimate! Even if we could guess some appropriate value p for p_B, we need to use p such that $p < p_B$, just to be safe, Then if we underestimate p_B and use much smaller p, we have to use unnecessarily large sample size.

This problem may be avoidable if we perform sampling in an *on-line* or *sequential* fashion. That is, a sampling algorithm obtains examples sequentially one by one, and it determines from these examples whether it has already seen enough number of examples. Intuitively, from the examples seen so far, we can more or less obtain some knowledge on the input data set, and it may be possible to estimate the parameters required by the statistical bounds. Thus, we do not fix sample size in advance. Instead sample size is determined *adaptively*. Then we can use more appropriate sample size for the current input data set.

For Problem 1, such a *sequential sampling algorithm* has been proposed by Lipton etal [28,29], which is stated in Figure 1. (Here we present a slightly simplified version.)

As we can see, the structure of the algorithm is simple. It runs until it sees more than A examples x with $B(x) = 1$. To complete the algorithm, we have to specify a way to determine A. For example, the Chernoff bound guarantees the following way. (In [28] the bound from the Central Limit Theorem is used, which gives a better sample size.)

```
program SampleAlg-1
    m ← 0;  n ← 0;
    while m < A do
        get x uniformly at random from D;
        m ← m + B(x);   n ← n + 1;
    output m/n as an approximation of p_B;
end.
```

Fig. 1. Sequential Sampling Algorithm for Problem 1

Theorem 2. *In the algorithm of Figure 1, we compute A by*

$$A \;=\; \frac{3(1+\varepsilon)}{\varepsilon^2} \ln\left(\frac{2}{\delta}\right).$$

Then for any $\delta > 0$ and ε, $0 < \varepsilon < 1$, the output of the algorithm satisfies (1).

Note that A does not depend on p_B. Thus, we can execute the sampling algorithm without knowing p_B. On the other hand, the following bound on the algorithm's sample size is also provable. Comparing it with the bound of Theorem 1, we see that the number of required examples is almost the same as the situation where we *knew* p_B in advance.

Theorem 3. *Consider the sample size n used by the algorithm of Figure 1. Then with probability $> 1 - \delta/2$, we have*

$$\textit{sample size } n \;\leq\; \frac{3(1+\varepsilon)}{(1-\varepsilon)\varepsilon^2 p_B} \ln\left(\frac{2}{\delta}\right).$$

While this is a typical example of sequential sampling, some sequential sampling algorithms have been developed recently (i) that can be applied to general tasks, and (ii) that have theoretical guarantees of correctness and performance [14,15,34]. Let us see here one application of such algorithms.

In some situations, we would like to estimate not $\widetilde{p_B}$ but some other value computed from $\widetilde{p_B}$. In [15], a general algorithm for achieving such a task is proposed. Here, as one example, we consider the problem of estimating $u_B = p_B - 1/2$. (This problem arises when implementing "boosting" techniques, which will be explained in the next section.)

Problem 2. *For given $\delta > 0$ and ε, $0 < \varepsilon < 1$, obtain an estimation $\widetilde{u_B}$ of u_B satisfying the following. (For simplifying our discussion, let us assume here that $u_B \geq 0$.)*

$$\Pr[\ |\widetilde{u_B} - u_B| \leq \varepsilon \cdot u_B\ ] \;>\; 1 - \delta. \tag{2}$$

Problem 2 is similar to Problem 1, but these two problems have different critical points. That is, while Problem 1 gets harder when p_B gets smaller, Problem 2 gets harder when $u_B = p_B - 1/2$ gets smaller. In other words, the closer p_B is to $1/2$, the more accurate estimation is necessary, and hence the more sample is

```
program SampleAlg-2
    m ← 0;  n ← 0;
    u ← 0;  α ← ∞;
    while u < α(1 + 1/ε) do
        get x uniformly at random from D;
        m ← m + B(x);   n ← n + 1;
        u ← m/n − 1/2;
        α ← √((1/2n) ln(n(n + 1)/δ));
    output u as an approximation of u_B;
end.
```

Fig. 2. Sequential Adaptive Sampling Algorithm for Problem 2

needed. Thus, Problem 2 should be regarded as a different problem, and a new sampling algorithm is necessary.

For designing a sequential sampling algorithm for estimating u_B, one might want to modify the algorithm of Figure 1. For example, by replacing its while-condition "$m < A$" with "$m - n/2 < B$" and by choosing B appropriately, we may be able to satisfy the new approximation goal. But this naive approach does not work. Fortunately, however, we can deal with this problem by using a slightly more complicated stopping condition. In Figure 2, we state a sequential sampling algorithm for Problem 2. Note that the algorithm does not use any information on u_B; hence, we can use it without knowing u_B at all. On the other hand, as shown in the following theorem, the algorithm estimates u_B with the desired accuracy and confidence. Also the sample size is bounded, with high probability, by $\mathcal{O}(1/(\varepsilon u_B)^2 \log(1/(\delta u_B))$, which bound is, ignoring the log factor, the same as the situation where u_B were known in advance.

Theorem 4. *For any $\delta > 0$ and ε, $0 < \varepsilon < 1$, the output of the algorithm of Figure 2 satisfies (2). Furthermore, with probability more than $1 - \delta$, we have*

$$\textit{sample size } n \lesssim \frac{2(1+2\varepsilon)^2}{(\varepsilon u_B)^2} \ln\left(\frac{1}{\varepsilon \delta u_B}\right).$$

Some Comments on Related Work

Since the idea of "sequential sampling" or "sampling on-line" is quite natural and reasonable, it has been studied in various contexts.

First of all, we should note that statisticians had already made significant accomplishments on sequential sampling during World War II [36]. In fact, from their activities, a research area on sequential sampling — sequential analysis — had been formed in statistics. For example, performing random sampling until the number of "positive observations" exceeds a certain limit, has been studied in depth in statistics. For recent studies on sequential analysis, see, e.g., [24].

In computer science, adaptive sampling techniques have been studied in the database community. The algorithm of Figure 1 was proposed [28,29] for estimating query size in relational databases. Later Haas and Swami [26] proposed

an algorithm that performs better than this in some situations. More recently, Lynch [30] gave a rigorous analysis to the algorithms proposed in [28,29].

One can see the spirit of adaptive sampling, i.e., to use instances observed so far for reducing a current and future computational task, in some algorithms proposed in the computational learning theory and machine learning community. For example, the Hoeffding race proposed by Maron and Moore [31] attempts to reduce a search space by removing candidates that are determined hopeless from the instances seen so far. A more general sequential local search has been proposed by Greiner [25]. More recently, sequential sampling algorithms have been developed for general data mining tasks and analyzed their performance both theoretically and experimentally; see, e.g., [15,34].

In the study of randomized algorithms, some sequential sampling algorithms have been also proposed and analyzed [12].

3 An Application of Sequential Sampling to Boosting

Problem 2 discussed in the previous section arises when we want to design a simple learner based on random sampling. Such a simple learner may be too weak for making a reliable hypothesis. But it can be used as a *weak learner* in a boosting algorithm. (Here by a *learner* we mean an algorithm that produces a mechanism or a rule — hypothesis — for making a yes/no classification on a given example. Let us again D to denote a given data set, which contains a huge number of instances, each of which expresses some object called *example* in this paper. Note that each example has a yes/no classification label. That is, we are given the way to classify examples given in D, and our goal is to find a good hypothesis explaining these classifications.)

A *boosting algorithm* is a way to design a strong learner yielding a reliable hypothesis by using a weak learner. Almost all boosting algorithms follow some common outline. A boosting algorithm runs a weak learner several times, say T times, under distributions $\mu_1, ..., \mu_T$ that are slightly modified from the given distribution μ on D and collects *weak hypotheses* $h_1, ..., h_T$. A *final hypothesis* is built by combining these weak hypotheses. Here the key idea is to put more weight, when making a new weak hypothesis, to "problematic instances" for which the previous weak hypotheses perform poorly. That is, at the point when $h_1, ..., h_t$ have been obtained, the boosting algorithm computes a new distribution μ_{t+1} that puts more weight on those instances that have been misclassified by most of $h_1, ..., h_t$. Then a new hypothesis h_{t+1} produced by a weak learner on this distribution μ_{t+1} should be strong on those problematic instances, thereby improving the performance of the combined hypothesis built from $h_1, ..., h_{t+1}$.

Boosting algorithms differ typically on a weighting scheme used to compute modified distributions. In [23] an elegant weighting scheme was introduced by Freund and Schapire, by which they defined a boosting algorithm called AdaBoost. It was proved that AdaBoost has a nice "adaptive" property; that is, it is adaptive to the quality of an obtained weak hypothesis so that the boosting process can converge quickly when better weak hypotheses are obtained. Fur-

thermore, many experimental results have shown that AdaBoost is indeed useful for designing a good learner for sevarl practical applications.

Now we would like to use AdaBoost to design a classification algorithm that can handle a huge data set. For this, we have to solve two algorithmic problems: (i) how to design a weak learner, and (ii) how to implement the boosting process.

Consider the first problem. AdaBoost does not specify its weak learner. Of course, a better learner yields a better hypothesis, which speeds up the boosting process. But a heavy weak learner is not appropriate for handing a large data set; for example, C4.5, a well-known decision tree making software, would be too costly to be used as a weak learner if it were used directly on the original huge data set D. Here random sampling can be used to reduce the data set size. For our discussion, let us take the following simplest approach: We consider a simple weak hypothesis so that the set $\mathcal{H}$ of all weak hypotheses remains relatively small. Then search *exhaustively* for a hypothesis $h \in \mathcal{H}$ that performs well on randomly sampled instances from D. For implementing this approach as a weak learning algorithm, Problem 2 (and its variation) becomes important.

Let us consider our simple approach in more detail. AdaBoost works with any weak learner except for one requirement. That is, for AdaBoost (and in fact any boosting algorithm) to work, it is necessary that a weak learner should (almost) always provide a hypothesis that is better than the random guess. Let $\mathrm{cor}_\mu(h)$ be the probability that a hypothesis h classifies correctly on instance of D that are generated randomly according to distribution μ. Then $\mathrm{adv}_\mu(h) = \mathrm{cor}_\mu(h) - 1/2$ is called the *advantage* of the hypothesis h. Note that the correct probability of the random guess is $1/2$; hence, the advantage of h shows how much h is better than the random guess. In order for AdaBoost to work, it is required that a weak learner should yield a hypothesis h_t (at each tth boosting step) whose advantage $\mathrm{adv}_{\mu_t}(h_t)$ is positive; of course, the larger advantage is the better. Then the task of a weak learner is to search for a hypothesis with (nearly) largest accuracy. We are considering a weak learner that uses random sampling to estimate $\mathrm{adv}_{\mu_t}(h)$ for each $h \in \mathcal{H}$ (and select the one with the best estimated advantage). This estimation is indeed our Problem 2. To guarantee that a selected hypothesis $h \in \mathcal{H}$ is nearly best (with a high probability), the sample size must be determined in terms of the best advantage γ; the more instances are needed if γ is close to 0 (in other words, the correct probability of the best hypothesis is close to $1/2$). That is, what is focused is the advantage of each hypothesis and not the correct probability. Hence, the problem needed to solve is Problem 2 and not Problem 1. Furthermore, the best advantage is not known in advance. Thus, this is the situation where sequential sampling is needed, and the algorithm of Figure 2 satisfies the purpose.

From Theorem 4, we can prove that the weak learner designed by using our sequential sampling algorithm needs to see roughly $\mathcal{O}(|\mathcal{H}|/\gamma^2)$ instances (ignoring some logarithmic factor) and runs within time proportional to this sample size. Thus, this weak learner runs with reasonable speed if both $|\mathcal{H}|$ and $1/\gamma$ are bounded within some range; in particular, an important point is that its running time does not depend on the size of the data set D. For example, we

conducted some experiments [17] by using the set of *decision stumps*, one node decision trees, for the class $\mathcal{H}$. Our experiments show that the obtained weak learner runs in reasonable amount of time in most cases, independent from data set size.

Next consider the second problem, i.e., an implementation of boosting process. In the boosting process (of AdaBoost and any other boosting algorithms), it is necessary to generate *training examples* under distributions modified from the original distribution[1]. Thus, some implementation of this example generation procedure is necessary. So far, we have two implementation frameworks:- (i) *boosting by sub-sampling* and *boosting by filtering* [20].

In the sub-sampling framework, before starting the boosting process, a set S of a certain number of instances are chosen first from D, and only elements in S are used during the boosting process. More specifically, at each boosting step t, the modified distribution μ_t at this step is defined on S, and the table of probabilities $\mu_t(x)$ for all $x \in S$ is calculated. Then we run a weak learning algorithm, supplying examples chosen from S according to their probabilities. (In many situations, it is not necessary to use any learning algorithm. We can simply find a weak hypothesis h that is best on S under μ_t by calculating the correct probability $\mathrm{cor}_{\mu_t}(h) = \sum_{x \in S(h)} \mu_t(x)$, where $S(h) = \{\, x \in S \,:\, h \text{ is correct on } x \,\}$.) In the filtering framework, on the other hand, we run a weak learning algorithm directly on the original data set D. Whenever an example is requested by a weak learner, it is generated from D by using a "filtering procedure" that yields an instance of D under the modified distribution μ_t.

In the sub-sampling framework, the whole set S is used throughout the boosting process; that is, the sample size for a weak learner is fixed. Thus, the adaptivity of our proposed weak learner loses its meaning. Also in many practical situations, it may not be easy to determine the size of the training set S in advance. On the other hand, the filtering framework fits very well to our weak learner, because the weak learner can control the sample size. Unfortunately, however, the filtering framework cannot be used in AdaBoost; under its weighting scheme, the weight of some examples may get exponentially large, and the filtering procedure cannot produce examples within a reasonable amount of time.

In [16] we introduced a new weighting scheme, which is obtained from the one used in AdaBoost by simply limiting the weight of each example by 1. Under this weighting scheme, it is possible to follow the filtering framework. This boosting algorithm — MadaBoost — implemented in the filtering framework is stated in Figure 3. Some explanation may be helpful for understanding some of the statements. First look at the main program of MadaBoost. The statement "$(h_t, \gamma_t) \leftarrow \mathrm{WeakLearn}(\mathrm{FiltEx}(t))$" means that (i) some weak learning algorithm is executed by supplying examples by using $\mathrm{FiltEx}(t)$, and then (ii) a hypothesis h_t with advantage γ_t $(= \mathrm{adv}_{\mu_t}(h_t))$ is obtained. For example, our simple weak learner based on the sequential sampling algorithm can be used here, which also gives a rea-

[1] Since we are assuming that the given data set D is huge, we may be able to assume that instances there already reflect the underlying distribution; thus, we may regard the uniform distribution on D as the original distribution.

program MadaBoost
  for $t \leftarrow 1$ **to** ∞ (until FiltEx terminates the iteration) **do**
    $(h_t, \gamma_t) \leftarrow \text{WeakLearn}(\text{FiltEx}(t))$
    $\beta_t \leftarrow \sqrt{1 - 2\gamma_t/(1 + 2\gamma_t)}$;
  end-for
  output the following f_t:
  $f_t(x) = \text{argmax}_{y \in Y} \sum_{i:h_i(x)=y} \log \frac{1}{\beta_i}$
end.

procedure FiltEx(t)
  loop-forever
    generate x uniformly at random from D;
    $w_t(x) \leftarrow \min\{1, \prod_{i=1}^{t} \beta^{\text{cons}(h_i,x)}\}$;
    % $\text{cons}(h_i, x) = 1$ if $h_i(x)$ is correct and -1 otherwise.
    with probability $w_t(x)$, output x and exit;
    if # of iterations exceeds some limit **then**
      exit and terminate the iteration of MadaBoost;
  end-loop
end.

Fig. 3. MadaBoost (in the Filtering Framework)

sonable estimation of the advantage γ_t. Using this advantage γ_t, the parameter β_t is defined next, which is used for defining the weight (i.e., probability) of each example, as well as for determining the importance of the obtained hypothesis h_t. A function f_t defined at the termination of the main iteration is the final hypothesis; intuitively, this function makes a prediction for a given example x by taking the weighted majority vote of the classifications made by weak hypotheses $h_1, ..., h_t$. MadaBoost is exactly the same as AdaBoost up to this point. The difference is the way to compute the weight $w_t(x)$ for each example $x \in D$, which is defined and used in the procedure FiltEX. When FiltEX(t) is called, it draws examples x randomly from D and filter them according to their current weights $w_t(x)$. In AdaBoost, the weight $w_t(x)$ is computed as $\prod_{i=1}^{t} \beta^{\text{cons}(h_i,x)}$; here we simply limit it by 1. Then it is much easier to get one example. Furthermore, if it gets hard to generate an example, then the boosting process (and the whole algorithm) can be terminated. This is because it is provable (under the new weighting scheme) that the obtained hypothesis is accurate enough when it gets hard to generate an example by this filtering procedure.

In [16] it is proved that the weighting scheme of MadaBoost still guarantees polynomial-time convergence. Unfortunately, MadaBoost's convergence speed that can be proved in [16] is exponentially slower than AdaBoost. However, our experiments [17] show that there is no significant difference on the convergence speed between AdaBoost and MadaBoost; that is, more or less the same number of boosting steps is sufficient. On the other hand, our experiments show that MadaBoost with our weak learner performs quite well on large data sets.

Some Comments on Related Work

The first boosting algorithm has been introduced by Schapire [33], for investigating a rather theoretical question asking the equivalence between the strong and weak PAC-learnability notions. But due to the success of AdaBoost for practical applications, several boosting techniques have been proposed since AdaBoost; see, e.g., the Proceedings of *the 13th Annual Conf. on Computational Learning Theory* (COLT'00).

Heuristics similar to boosting or related boosting have been also proposed and investigated experimentally; see, e.g., [7,13,1].

It has been said that boosting does not work well when error or noise exists. That is, the case when some of the instances in D have a wrong classification label. In particular, this problem seems serious in AdaBoost. As we mentioned above, the weighting scheme of AdaBoost changes weights rapidly, which makes AdaBoost too sensitive to erroneous examples. Again this problem is reduced by using a more moderate weighting scheme like MadaBoost. In fact, it is shown [16] that MadaBoost is robust to a standard statistical errors, errors that could be corrected by seeing sufficient number of instances. (Recall that D may be a multiset and one example may appear in D more than once. Then by seeing enough number of instances of D for the same example, we may be able to fix it even if some instance is labeled wrongly. See, e.g., [27] for the formal argument.) For solving this problem of AdaBoost, Freund [21] introduced a new weighting scheme and proposed a new boosting algorithm — BrownBoost —- which is also robust to the statistical errors of the above type. Unfortunately, however, even these algorithms may not be robust against errors that are not fixed by just looking many instances. One typical example is the case where the data set contains some exceptions, examples that should be treated as exceptions.

4 Random Sampling for Support Vector Machines

Boosting provide us with a powerful methodology for designing better learning algorithms. Unfortunately, however, boosting algorithms may not work sufficiently if a given data set contains many errors or exceptions. On the other hand, there is a popular classification mechanism — support vector machine (in short SVM) — that works well even if the data set contains some "outliers", i.e., erroneous examples or exceptions. Here again we show that random sampling can be used for designing an efficient SVM training algorithm. (*Remark.* One important point of the SVM approach is so called the "kernel method" that enables us to obtain hyperplane separation in a much higher dimension space than the original space. In this paper, this point is omitted, and we consider a simple hyperplane separation. See our original paper [3] for an extension to the kernel method. See also [4] that proposes another algorithm derived from a similar approach.)

First we explain basics concerning SVM. Since we only explain those necessary for our discussion, see, e.g., a good textbook [11] for more systematic and comprehensive explanation. A *support vector machine* (discussed here) is a clas-

sification mechanism that classifies examples by using a hyperplane. This time our data set D is a set of m instances in some n dimension space. Here again we consider the binary classification, and we assume that instances in D has a binary label $+1$ for *postive* and -1 for *negative* examples. Then SVM training is to compute a hyperplane separating positive and negative examples with the largest margin. Precisely, our goal is to solve the following optimization problem[2]. (Let us assume for a while that D has no erroneous instances and positive and negative examples can be separated by some hyperplane in the n dimension space. To simplify our discussion, we assume here that D is an ordinary set. That is, an example appears at most once as an instance in D; hence, there is no distinction between example and instance. Also we assume that examples in D are indexed as $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m$, where each $\boldsymbol{x}_i$ has a label y_i.)

Max Margin (P1)

$$\begin{aligned} &\text{min. } \frac{1}{2}\|\boldsymbol{w}\|^2 - (\theta_+ - \theta_-) \quad \text{w.r.t. } \boldsymbol{w} = (w_1, ..., w_n),\ \theta_+, \text{ and } \theta_-, \\ &\text{s.t. } \ \boldsymbol{w} \cdot \boldsymbol{x}_i \ \geq \ \theta_+ \quad \text{if } y_i = 1, \ \text{ and } \ \boldsymbol{w} \cdot \boldsymbol{x}_i \ \leq \ \theta_- \quad \text{if } y_i = -1. \end{aligned}$$

This optimization problem is a quadratic programming (QP in short) problem. Though QP is in general polynomial-time solvable, existing (general) QP solvers are not efficient enough to solve a large QP problems. Here random sampling can be used. In fact, some random sampling techniques have been developed and used for such combinatorial optimization problems (see, e.g., [18]), and we simply use one of them. This technique yields a training algorithm that works particularly well when the dimensionality n (i.e., the number of attributes) is moderate but the size m of D is huge.

The idea of the random sampling technique is in fact similar to boosting. Pick up a certain number of examples from D and solve (P1) under the set of constraints corresponding to these examples. We choose examples randomly according to their "weights", where initially all examples are given the same weight. Clearly, the obtained local solution is, in general, not the global solution, and it does not satisfy some constraints; in other words, some examples are misclassified by the local solution. Then double the "weight" of such misclassified examples, and then pick up some examples again randomly according to their weights. If we iterate this process several rounds, the weight of "important examples", examples defining the solution hyperplane (which are called *support vectors*), would get increased, and hence, they are likely to be chosen. Note that once all support vectors are chosen at some round, then the local solution of this round is the real one, and the algorithm terminates at this point.

More specifically, we can in this way design a SVM training algorithm stated in Figure 4. Let us see the algorithm a bit more in detail. In the algorithm, $w(\boldsymbol{x})$ denotes the current weight of an example $\boldsymbol{x}$. Examples in D are drawn randomly from D so that the probability that each example $\boldsymbol{x}$ is chosen is proportional to its current weight $w(\boldsymbol{x})$. Let R be the set of examples chosen in this way, and

[2] Here we follow [5] and use their formulation. But the above problem can be restated by using a single threshold parameter as given originally in [10].

program RandomSamplingSolver
  set $w(\boldsymbol{x}) \leftarrow 1$ for each $\boldsymbol{x} \in D$;
  $r \leftarrow 6\delta^2$;
  repeat
    $R \leftarrow$ choose r examples $\boldsymbol{x}$ from D randomly according to their weights $w(\boldsymbol{x})$;
    Solve (P1) for R and let $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*)$ be its solution;
    $V \leftarrow$ the set of violators to $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*)$;
    if $w(V) \leq w(D)/3\delta$ **then**
      double the weight $w(\boldsymbol{x})$ of $\boldsymbol{x} \in V$;
  until $V = \emptyset$;
  output the current solution;
end.

Fig. 4. SVM Training Algorithm Based on Random Sampling

we solve (P1) on R to obtain a local solution $(\boldsymbol{w}^*, \theta_+^*, \theta_-^*)$. A *violator* is simply an example that is misclassified by the obtained local solution. The parameter δ denotes a *combinatorial dimension*, a quantity expressing the complexity of a combinatorial optimization problem to be solved; for the problem (P1), we have $\delta \leq n+1$, which is from the fact that $n+1$ support vectors are enough to define the solution hyperplane. Note that the number of examples chosen for R is computed from n and δ $(= n+1)$ and independent from m, the size of D. Thus, our algorithm is appropriate if $m >> n$, i.e., the number of examples is much larger than the number of attributes.

Note that the weight of violators get increased only when their total weight is at most one third of the total weight, that is, not so many violators exist. But we can prove that this situation does not occur often by using "Sampling Lemma" [8,18]. Then the following bound follows from this fact.

Theorem 5. *The average number of iterations executed in the algorithm of Figure 4 is bounded by* $6\delta \ln m$.

Recall that $\delta \leq n+1$; thus, this theorem shows that the algorithm needs to solve (P1) roughly $O(n \ln m)$ times. On the other hand, the size of R and hence the time needed to solve (P1) at each iteration is bounded by some polynomial in n.

Next consider the case that a given data set D contains "outliers", and no hyperplane can separate positive/negative examples given in D. In the SVM approach, this situation can be handled by considering "soft margin", that is, by relaxing constraints with slack variables. Precisely, we consider the following generalization of (P1).

Max Soft Margin (P2)

$$\begin{aligned} \text{min. } & \frac{1}{2}\|\boldsymbol{w}\|^2 - (\theta_+ - \theta_-) + K \cdot \sum_i \xi_i \\ & \text{w.r.t. } \boldsymbol{w} = (w_1, ..., w_n),\ \theta_+,\ \theta_-, \text{ and } \xi_1, ..., \xi_m \\ \text{s.t. } & \boldsymbol{w} \cdot \boldsymbol{x}_i \ \geq\ \theta_+ - \xi_i \quad \text{if } y_i = 1, \quad \boldsymbol{w} \cdot \boldsymbol{x}_i \ \leq\ \theta_- + \xi_i \quad \text{if } y_i = -1, \\ & \text{and } \ \xi_i \ \geq\ 0. \end{aligned}$$

Here variables $\xi_i \geq 0$ are newly introduced, which express the penalty of $\boldsymbol{x}_i$ being misclassified by a hyperplane. That is, examples can be misclassified by a solution in this formulation, but the penalty $\boldsymbol{x}_i$ is added to the cost if $\boldsymbol{x}_i$ is misclassified. Intuitively, both the margin from the separating hyperplane and the cost of misclassifications are taken into account in this problem (P2). At this point, we can formally define our notion of "outliers". An *outlier* is an example $\boldsymbol{x}_i$ that is misclassified (in other words, with $\xi_i > 0$) by the solution of (P2). Note here that we use a parameter $K < 1$ for adjusting the degree of influence from misclassified examples, and that the solution of (P2) and the set of outliers depend on the choice of K.

Now our problem is to solve (P2). We assume that the parameter K is appropriately chosen [6]. We may still use the algorithm of Figure 4. The point we need to consider is only the choice of δ, the combinatorial dimension of (P2). Since (P2) is defined with $n+m+2$ variables, δ is bounded by $n+m+1$ by the straightforward generalization of the argument for (P1). But this bound is too large; in fact, the number of examples required for R becomes much larger than the original number m of examples! Fortunately, however, we can show [3] that δ is indeed bounded by $n+\ell+1$, where ℓ is the number of outliers. (The proof uses the geometrical interpretation of (P2) given by Bennett and Bredensteiner [5].) Then by using the bound of Theorem 5, we can conclude the following: the algorithm needs to solve (P2) roughly $O((n+\ell)\ln m)$ times, and the time needed to solve (P2) at each time is bounded by some polynomial in $n+\ell$.

Comments on Related Work

The present form of SVM was first proposed by Cortes and Vapnik [10]. Many algorithms and implementation techniques have been developed for training SVMs efficiently; see, e.g., [35,6]. This is because solving QP for training SVMs is costly. Among speed-up techniques, those called "subset selection" [35] have been used as effective heuristics from the early stage of the SVM research. Roughly speaking, a *subset selection* is to divide the original QP problem into small pieces, thereby reducing the size of each QP problem. Well known subset selection techniques are chunking, decomposition, and sequential minimal optimization (SMO in short). In particular, SMO has become popular because it outperforms the others in several experiments. (See, e.g., [10,32,11] for the detail.) Though the performance of these subset selection techniques has been extensively examined, no theoretical guarantee has been given on the efficiency of algorithms based on these techniques. On the other hand, we can theoretically guarantee the efficiency of our algorithm based on random sampling. Compared with existing heuristics, our algorithm may not be efficient as it, but it can be used with the other heuristics to obtain an yet faster algorithm. Furthermore, using our weighting scheme, it may be possible to identify some of the outliers in earlier stages, thereby making the problem easier to solve by removing them.

The idea of using random sampling to design efficient randomized algorithms was first introduced by Clarkson [8]. This approach has been used for solving various combinatorial optimization problems. Indeed a similar idea has been used [2] to design an efficient randomized algorithm for QP. More recently, Gärtner

and Welzl [18] introduced a general framework for discussing such randomized sampling techniques. Our algorithm of Figurre 4 and its analysis for (P1) are immediate from their general argument.

References

1. N. Abe and H. Mamitsuka, Query learning strategies using boosting and bagging, in *Proc. the 15th Int'l Conf. on Machine Learning* (ICML'00), 1–9, 1998.
2. I. Adler and R. Shamir, A randomized scheme for speeding up algorithms for linear and convex programming with high constraints-to-variable ratio, *Math. Programming* 61, 39—52, 1993.
3. J. Balcázar, Y. Dai, and O. Watanabe, Provably fast training algorithms for support vector machines, in *Proc. the first IEEE Int'l Conf. on Data Mining*, to appear.
4. J. Balcázar, Y. Dai, and O. Watanabe, Random sampling techniques for training support vector machines: For primal-form maximal-margin classifiers, in *Proc. the 12th Int'l Conf. on Algorithmic Learning Theory* (ALT'01), to appear.
5. K.P. Bennett and E.J. Bredensteiner, Duality and geometry in SVM classifiers, in *Proc. the 17th Int'l Conf. on Machine Learning* (ICML'2000), 57–64, 2000.
6. P.S. Bradley, O.L. Mangasarian, and D.R. Musicant, Optimization methods in massive datasets, in *Handbook of Massive Datasets* (J. Abello , P.M. Pardalos, and M.G.C. Resende, eds.), Kluwer Academic Pub., to appear.
7. L. Breiman, Pasting small votes for classification in large databases and on-line, *Machine Learning* 36, 85–103, 1999.
8. K.L. Clarkson, Las Vegas algorithms for linear and integer programming, *J.ACM* 42, 488—499, 1995.
9. M. Collins, R.E. Schapire, and Y. Singer, Logistic regression, AdaBoost and Bregman Distance, in *Proc. the 13th Annual Conf. on Comput. Learning Theory* (COLT'00), 158–169, 2000.
10. C. Cortes and V. Vapnik, Support-vector networks, *Machine Learning* 20, 273–297, 1995.
11. N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge Univ. Press, 2000.
12. P. Dagum, R. Karp, M. Luby, and S. Ross, An optimal algorithm for monte carlo estimation, *SIAM J. Comput.* 29(5), 1484–1496, 2000.
13. T.G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization, *Machine Learning* 32, 1–22, 1998.
14. C. Domingo, R. Gavaldà, and O. Watanabe, Practical algorithms for on-line selection, in *Proc. the first Intl. Conf. on Discovery Science* (DS'98), Lecture Notes in AI 1532, 150–161, 1998.
15. C. Domingo, R. Gavaldà, and O. Watanabe, Adaptive sampling methods for scaling up knowledge discovery algorithms, in *Proc. the 2nd Intl. Conf. on Discovery Science* (DS'99), Lecture Notes in AI, 172–183, 1999. (The final version will appear in *J. Knowledge Discovery and Data Mining*.)
16. C. Domingo and O. Watanabe, MadaBoost: A modification of AdaBoost, in *Proc. the 13th Annual Conf. on Comput. Learning Theory* (COLT'00), 180–189, 2000.
17. C. Domingo and O. Watanabe, Scaling up a boosting-based learner via adaptive sampling, in *Proc. of Knowledge Discovery and Data Mining* (PAKDD'00), Lecture Notes in AI 1805, 317–328, 2000.

18. B. Gärtner and E. Welzl, A simple sampling lemma: Analysis and applications in geometric optimization, *Discr. Comput. Geometry*, to appear. (Also available from `http://www.inf.ethz.ch/personal/gaertner/publications.html`.)
19. W. Feller, *An Introduction to Probability Theory and its Applications* (Third Edition), John Wiley & Sons, 1968.
20. Y. Freund, Boosting a weak learning algorithm by majority, *Information and Computation* 121(2), 256–285, 1995.
21. Y. Freund, An adaptive version of the boost by majority algorithm, in *Proc. the 12th Annual Conf. on Comput. Learning Theory* (COLT'99), 102–113, 1999.
22. J. Friedman, T. Hastie, and R. Tibshirani, Additive logistic regression: a statistical view of boosting, Technical Report, 1998.
23. Y. Freund and R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, *J. Comput. Syst. Sci.* 55(1), 119–139, 1997.
24. B.K. Ghosh and P.K. Sen eds., *Handbook of Sequential Analysis*, Marcel Dekker, 1991.
25. R. Greiner, PALO: a probabilistic hill-climbing algorithm, *Artificial Intelligence* 84, 177–204, 1996.
26. P. Haas and A. Swami, Sequential sampling, procedures for query size estimation, *IBM Research Report* RJ 9101(80915), 1992.
27. M. Kearns, Efficient noise-tolerant learning from statistical queries, in *Proc. the 25th Annual ACM Sympos. on Theory of Comput.* (STOC'93), 392–401, 1993.
28. R.J. Lipton, J.F. Naughton, D.A. Schneider, and S. Seshadri, Efficient sampling strategies for relational database operations, *Theoret. Comput. Sci.* 116, pp.195–226, 1993.
29. R.J. Lipton and J.F. Naughton, Query size estimation by adaptive sampling, *J. Comput. and Syst. Sci.* 51, 18–25, 1995.
30. J.F. Lynch, Analysis and application of adaptive sampling, in *Proc. the 19th ACM Sympos. on Principles of Database Systems* (PODS'99), 260–267, 1999.
31. O. Maron and A. Moore, Hoeffding races: accelerating model selection search for classification and function approximation, in *Proc. Advances in Neural Information Process. Systems* (NIPS'94), 59–66, 1994.
32. J. Platt, Fast training of support vector machines using sequential minimal optimization, in *Advances in Kernel Methods – Support Vector Learning* (B. Scholkopf, C.J.C. Burges, and A.J. Smola, eds.), MIT Press, 185–208, 1999.
33. R.E. Schapire, The strength of weak learnability, *Machine Learning* 5(2), 197–227, 1990.
34. T. Scheffer and S. Wrobel, A sequential sampling algorithm for a general class of utility criteria, in *Proc. the 6th ACM Intl. Conf. on Knowledge Discovery and Data Mining* (KDD'00), 2000.
35. A.J. Smola and B. Scholkopf, A tutorial on support vector regression, NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, Univ. London, 1998.
36. A. Wald, *Sequential Analysis*, John Wiley & Sons, 1947.

On the Approximability of Interactive Knapsack Problems

Isto Aho*

Dept. of Computer and Information Sciences,
33014 University of Tampere, Finland
tyisah@cs.uta.fi

Abstract. We show that the interactive knapsack heuristic optimization problem is APX-hard. Moreover, we discuss the relationship between the interactive knapsack heuristic optimization problem and some other knapsack problems.

1 Introduction

The interactive knapsack (IK) model is a generalization of the model behind the generalized assignment problem. An instance of the interactive knapsack problem contains several knapsacks connected somehow together: inserting an item in one of the knapsacks affects also to the other knapsacks (in a way to be described below). Most reasonable problems using the IK model are strongly NP-complete. One possible approach to avoid computational difficulties is to restrict the problems to use one item at a time leading to the interactive knapsack heuristic optimization (IKHO) problems. Unfortunately, some IKHO problems are NP-complete, too. (See [1].)

In the IK model we have an ordered group of knapsacks, also called a knapsack array. When inserting an item j into a knapsack i, it gets copied into knapsacks $i+1, \ldots, i+c$ ($c \geq 0$). The inserted item and its copies together are called a *clone*. Moreover, an item can also "radiate", i.e. some portions of the item can be copied into knapsacks $i-u, \ldots, i-1$ and $i+c+1, \ldots, i+c+u$ ($u \geq 0$). This behavior (clone and radiation) can model, for example, the controls made in the electricity management, like load clipping (see [1,2,3]).

We suppose a familiarity with the basics of complexity theory as given, e.g. in [8,12].

We are able to prove that IKHO is APX-hard. Moreover, we establish a connection between IKHO and the multi-dimensional 0-1 knapsack problem. This suggests the existence of new applications of IKHO in query optimization and in related areas, where the multi-dimensional 0-1 knapsack problem is applied [5].

At the end of this section we formulate IKHO handling one item at a time. We allow the same item to be inserted several times into the knapsack array

* Author is grateful to Erkki Mäkinen for his time and comments.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 152–159, 2001.

but not into the same knapsack. Similarly, in load clipping, we can usually make several controls with the same utility. Other possible applications are described in [1].

In the 0-1 knapsack problem [11] we want to maximize $\sum_{j=1}^{n} p_j x_j$ such that $\sum_{j=1}^{n} w_j x_j \leq b$. We have one knapsack, whose capacity is b ($b > 0$). Profits p and weights w are n-vectors of positive integers and x is a binary vector ($x \in \{0,1\}^n$). The 0-1 knapsack problem is NP-complete but it has a fully polynomial time approximation scheme (see [12], pp. 305–307).

The generalized assignment problem (GAP) [11] maximizes $\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij}$ subject to $\sum_{j=1}^{n} w_{ij} x_{ij} \leq b_i$ $(i = 1, \ldots, m)$, $\sum_{i=1}^{m} x_{ij} \leq 1$ $(j = 1, \ldots, n)$, and $x_{ij} = 0$ or 1, $(i = 1, \ldots, m,\ j = 1, \ldots, n)$. In GAP the profit and weight of an item depend on the knapsack. Notice that all values p_{ij}, w_{ij} and b_i are positive integers. Martello and Toth [11] call this problem LEGAP; in their terminology, GAP is supposed to have condition $\sum_{i=1}^{m} x_{ij} = 1$ instead of the inequality used above.

GAP is APX-hard and it can be 2-approximated. The APX-hardness is shown for a very restricted case of GAP [6]. This restricted special case of GAP is used in Sections 2 and 3.

The multi-dimensional knapsack problem (MDKP) [4] is similar to GAP. However, this time the selection of an item means that we add an amount to every knapsack. Hence, we select only items (and not the knapsack at the same time, as in GAP). In MDKP we are to maximize $\sum_{j=1}^{n} p_j x_j$ subject to $\sum_{j=1}^{n} w_{ij} x_j \leq b_i$ $(i = 1, \ldots, m)$, and $x_j = 0$ or 1 $(i = 1, \ldots, m,\ j = 1, \ldots, n)$. Lin [10] gives a survey on some well-known non-standard knapsack problems (including MDKP).

MDKP has a PTAS, if m is fixed [7]. Chekuri and Khanna [5] show that MDKP is hard to approximate within a factor of $\Omega\big(m^{1/(B+1)-\epsilon}\big)$ for every fixed $B = \min_i b_i$. Srinivasan [13] shows how to obtain in polynomial time $\Omega(t^{B/(B+1)})$ solutions, where $t = \Omega(z^*/m^{1/B})$ and z^* is the optimal solution. We obtain better approximations as B (the smallest knapsack) increases.

MDKP is closely connected to some integer programming problems. The packing integer program (PIP) [5] coincides with MDKP [5], but we could call it the positive max 0-1 integer program as well, because all w_{ij}, p_j and b_i consist of positive elements.

In Section 2 we present an approximation preserving transformation between GAP and IKHO. Moreover, we show that IKHO equals MDKP. In Section 3 we characterize the transformation between GAP and MDKP.

In IKHO we have $x_i \in \{0,1\}$, profits $p_i \in \mathbb{N}$, weights $w_i \in \mathbb{N}$, lengths of clones c or $c_i \in \mathbb{N}$, lengths of radiations u or $u_i \in \mathbb{N}$, and functions $I_i : \mathbb{Z} \to \mathbb{Q}$, for knapsacks $i = 1, \ldots, m$. Moreover, we have a capacity $b_i \in \mathbb{N}$, for each knapsack i, and a positive integer K. Interaction I_i of knapsack i maps the distance from i to a rational number. By using c (clone) and u (radiation) we mean that I_i equals one, for knapsacks $i, \ldots, i+c$, and is arbitrary for knapsacks $i-u, \ldots, i-1$ and $i+c+1, \ldots, i+c+u$. For all other knapsacks, I_i is zero.

(Fixed) IKHO problem is to

$$\max \sum_{i=1}^{m} x_i \sum_{k=i-u}^{i+c+u} I_i(k)p_k \tag{1}$$

$$\text{subject to } \sum_{i=1}^{m} w_i I_i(l) x_i \leq b_l, \tag{2}$$

$$x_k = 0, \text{ for } i < k \leq i + c, \text{ when } x_i = 1, \tag{3}$$

$$x_i = 0 \text{ or } 1, \text{ and} \tag{4}$$

$$\sum_{i=1}^{m} x_i \leq K, \tag{5}$$

where $l = 1, \ldots, m$ in (2) and $i = 1, \ldots, m$ in (3) and (4).

If $K = O(1)$ or $u = 0$, the above problem is solvable in polynomial time [1]. When $K = O(m)$ and $u = m$, it becomes NP-complete [1]. This can be seen by a transformation from the 0-1 knapsack problem. Thus, the questions about the existence of PTAS and FPTAS algorithms for IKHO are relevant. The NP-completeness of IKHO is an open question, when $u = O(\log(m))$.

We see easily that the knapsack problem build up by the radiation uses at most u different profits and weights. Thus, if $u = O(m)$ the problem remains NP-complete (m/c must be $O(m)$, of course). As noted, other values for u may or may not imply polynomial time algorithms.

2 Transformation from GAP into IKHO

We reduce a highly restricted version of GAP to IKHO. As shown in [6], GAP is APX-hard even on the instances of the following form, for all positive δ:

- $p_{ij} = 1$, for all j and i,
- $w_{ij} = 1$ or $w_{ij} = 1 + \delta$, for all j and i, and
- $b_i = 3$, for all i.

In what follows, this problem is referred to as the restricted GAP.

The APX-hardness of the restricted GAP means that there exists $\epsilon > 0$ such that it is NP-hard to decide whether an instance has an assignment of $3m$ items or if each assignment has value at most $3m(1 - \epsilon)$ (see [6]). In the sequel, we suppose that $u = m$.

Theorem 1. *IKHO is APX-hard.*

Proof. Given an instance of the restricted GAP (having m knapsacks and n items), we create an instance of IKHO as follows.

If $x_{ij} = 1$ in GAP, then $x'_{j_i} = 1$ in IKHO. In IKHO we have knapsacks $j_1, \ldots, j_m$ for each item j, and knapsacks $b'_1, \ldots, b'_m$ similar to knapsacks in GAP.

Interaction I is defined in such a way that knapsacks $j_1, \ldots, j_i, \ldots, j_m$ will be full when item j_i is chosen (i.e. when we set $x'_{j_i} = 1$). Hence, item j can be put at most once into the IKHO array. Moreover, interaction for knapsack b'_i is such that the weight w_{ij} of GAP will be put into knapsack b'_i. Other knapsacks are left intact. We need only radiation (i.e. $c = 0$) defined as:

$$I_{j_i}(k) = \begin{cases} 1+z, & \text{when } k = b'_i, \\ 1, & \text{when } k \in [j_1, j_{i-1}] \cup [j_{i+1}, j_m], \\ 0, & \text{otherwise,} \end{cases}$$

where $z = 0$, if $w_{ij} = 1$, and $z = \delta$, if $w_{ij} = 1 + \delta$.

The capacity of knapsacks $b'_1, \ldots, b'_m$ is 3, while the capacity of knapsacks $j_1, \ldots, j_m$ is 1, for each item j. Profit is $1/(m+1)$, for each nm items, because an item with interaction involves $m+1$ knapsacks. Hence, the profit given by radiation is $\sum_{k=1}^{m+1} 1/(m+1)$ and it equals 1, for each item. We set $w_{j_i} = 1$, for each item and knapsacks $1_1, \ldots, n_m$. The weight is 4, for knapsacks $b'_1, \ldots, b'_m$, i.e. we cannot set $x_{b'_i} = 1$, for any i.

Three items can be put into a knapsack i in GAP if and only if three items can be fitted into a knapsack b'_i through interaction. If GAP has a full assignment of value $3m$, the corresponding IKHO instance has the same value and knapsacks b'_i are full. Otherwise, in GAP the value is at most $3m(1-\epsilon)$ as well as in IKHO. Each item in GAP can be assigned only once and IKHO behaves similarly. □

Hence, a PTAS for IKHO would contradict the fact that the restricted GAP is APX-hard. Figure 1 shows the knapsack structure used in Theorem 1. It corresponds to a situation where we set $x_{11} = 1$ in GAP, i.e. we put the first item into the first knapsack. In IKHO we set $x_{1_1} = 1$ and radiation takes care that knapsacks $1_2, \ldots, 1_m$ are full so that the first item of GAP cannot be put into any other knapsack any more. Moreover, to handle the normal GAP knapsack restriction, the interaction puts to the first corresponding knapsack b_1 in IKHO the same amount than in GAP. In this example, weight w_{11} in GAP is $1 + \delta$.

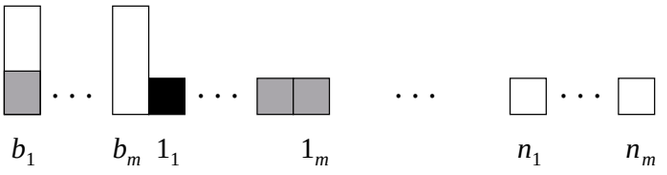


Fig. 1. The knapsack structure used in Theorem 1.

Theorem 1 considers the special case of IKHO where $c = 0$, $u = m$ and $K = m$. We can also reformulate IKHO so that it equals a special case of MDKP. Thus, we can use the algorithms designed for MDKP [7,13] to solve this special case of IKHO.

First, define $p'_i = \sum_{k=i-u}^{i+c+u} I_i(k)p_k$ $(i = 1, \ldots, m)$. This changes our objective function (1) to be $\sum_{i=1}^{m} x_i p'_i$. Second, define $w'_{ik} = w_i I_i(k)$, for $i, k = 1, \ldots, m$. As a result, condition (2) turns to be $\sum_{i=1}^{m} w'_{ik} x_i \leq b_k$ $(k = 1, \ldots, m)$. Again, an item in knapsack i means that we add a weight w'_{ik}, for each knapsack $k = 1, \ldots, m$.

Now conditions (3) and (5) are both fulfilled. We have converted our problem into

$$\max \sum_{i=1}^{m} p'_i x_i \tag{6}$$

$$\text{subject to } \sum_{i=1}^{m} w'_{ik} x_i \leq b_k, \quad k = 1, \ldots, m. \tag{7}$$

$$x_i = 0 \text{ or } 1, \quad i = 1, \ldots, m. \tag{8}$$

Condition (7) is equal to $Wx \leq b$, where W is an $m \times m$ matrix with positive elements. The above problem is a special case of MDKP containing m knapsacks and m items (as opposite to n items).

As pointed out, MDKP has a PTAS for fixed number of knapsacks but is otherwise hard to approximate. Even though IKHO seems to be easier than MDKP with unfixed number of knapsacks, Theorem 1 rules out the possibility of having a PTAS for IKHO. Moreover, Chekuri and Khanna [5] show also that MDKP is hard to approximate even when $m = \text{poly}(n)$ (recall that in our case $m = n$).

The cases of IKHO with some fixed $c > 0$ are at least as hard as the case with $c = 0$. The number of items is not any more m, but rather $m/(c+1)$. However, restriction (3) is non-linear.

Since some of the algorithms for MDKP are based on linear programming relaxation [13] and as the efficiency depends on the number of knapsacks (to the number of linear restrictions), restriction (3) should be efficiently transformed into a linear form. Because $c > 0$ is fixed, we can do it as follows.

By restricting $x_i + x_j \leq 1$ we can ensure that at most one of x_i or x_j equals one. The m restrictions of form (3) are transformed into cm restrictions of the form

$$x_k + x_{k+i} \leq 1, \quad \text{where } i = 1, \ldots, c \text{ and } k = 1, \ldots, m. \tag{9}$$

If $x_k = 1$, then the following c knapsacks x_i $(i = k+1, \ldots, k+c)$ fulfill the condition $x_i = 0$. This does not change the hardness of approximating IKHO by using MDKP.

Hence, we have shown

Theorem 2. *For IKHO, we can obtain solutions of value $\Omega(t^{B/(B+1)})$, where $t = \Omega(z^*/m^{1/B})$, z^* is the optimal solution, and B is the size of the smallest knapsack.*

Theorem 2 holds also for the case, where the clone length depends on the knapsack into which we put an item. Thus, in the problem setting (1)–(5) we can change every c to be c_i.

Actually, we can reduce MDKP to IKHO as well. Let W' be the integer weight matrix and p' integer profit vector of MDKP. Define $w_i = 1$ and $I_i(k) = w'_{ik}$.

Now, we have to find profits p_i such that $p'_i = \sum_{k=i-u}^{i+c+u} I_i(k)p_k$. We can set $p = W'^{-1}p'$, if we use a matrix pseudo-inverse (see, e.g. [9]). Calculating pseudo-inverse (up to some predefined accuracy) takes polynomial time. Elements p_i are rational numbers but the sum $\sum_{k=i-u}^{i+c+u} I_i(k)p_k$ is near to p'_i and can be rounded to an integer. Finally, set $K = m$, $c = 0$ and $u = m$.

When $x'_i = 1$ in MDKP, $x_i = 1$ in IKHO. Profits will be the same (after obvious rounding) and knapsack restrictions are handled equally in both problems. Thus IKHO is no easier than MDKP.

Theorem 3. *Unless NP=ZPP, IKHO is hard to approximate to within a factor of* $\Omega(m^{1/(\lfloor B \rfloor + 1)-\epsilon})$*, for any fixed* $\epsilon > 0$*, where* $B = \min_i b_i$ *is fixed.*

Proof. See the discussion above and the results of [5]. □

While Theorem 2 gives us methods to solve IKHO within a known precision in polynomial time, Theorem 3 sharpens the result of Theorem 1. After Theorems 1 and 3 it is natural to discuss the nature of transformation from GAP into MDKP.

3 Transformation from GAP into MDKP

A direct transformation from the restricted GAP into MDKP is also possible by using a similar construction than the one used in the proof of Theorem 1. By using

$$x = (x_{1_1} \ \cdots \ x_{1_m} \ \cdots \ x_{n_1} \ \cdots \ x_{n_m})^T$$

(T stands for the matrix transpose) and by adding

$$\underbrace{0 \ \cdots \ 0}_{(j-1)m} \ \underbrace{1 \ \cdots \ 1}_{m \text{ ones}} \ 0 \ \cdots \ 0,$$

for each item j in W, where W stands for the weight matrix, we can ensure that only one of the values $x_{j_1}, \ldots, x_{j_m}$ can be one. Ones occur in those positions in b, which correspond to the rows W described above. GAP gives other rows directly:

$$0 \ \cdots \ 0 \ w_{i1} \ 0 \ \cdots \ 0 \ w_{i2} \ 0 \ \cdots \ 0 \ w_{in} \ 0 \ \cdots \ 0,$$

for each knapsack i. Notice that the first non-zero values take place at positions i mod $(m+1)$ and the next (rest) non-zeros occur after $m-1$ zeros. The size of W is $(m+n) \times nm$. Vectors p and b are defined in the obvious manner. Again, the profit gap between an instance of the restricted GAP and MDKP is the same.

To summarize, we have

$$W = \begin{bmatrix}
w_{11} & 0 & \cdots & 0 & w_{12} & 0 & \cdots & \cdots & 0 \cdots 0 & w_{1n} & 0 & \cdots & 0 \\
0 & w_{21} & 0 & \cdots & 0 & w_{22} & 0 & \cdots & 0 \cdots 0 & 0 & w_{2n} & \cdots & 0 \\
 & & \ddots & & & & \ddots & & \ddots & & & \ddots & \\
0 & \cdots & 0 & w_{m1} & 0 & \cdots & 0 & w_{m2} & 0 \cdots 0 & 0 & \cdots & 0 & w_{mn} \\
1 & \cdots & 1 & 1 & 0 & \cdots & 0 & 0 & 0 \cdots 0 & 0 & \cdots & \cdots & 0 \\
0 & \cdots & 0 & 0 & 1 & \cdots & 1 & 1 & 0 \cdots 0 & 0 & \cdots & \cdots & 0 \\
 & & & & & & & & \ddots & & & & \\
0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \cdots 0 & 1 & 1 & \cdots & 1
\end{bmatrix},$$

and

$$b = (b_1 \ \cdots \ b_i \ \cdots \ b_m \ \underbrace{1 \ \cdots \ 1}_{n})^T,$$

where $b_i = 3$ $(i = 1, \ldots, m)$ in the case of the restricted GAP, and

$$p = (p_{1_1} \ \cdots \ p_{1_m} \ \cdots \ p_{n_1} \ \cdots \ p_{n_m})^T,$$

where $p_i = 1$ $(i = 1_1, \ldots, n_m)$ for the restricted GAP. (Hence, MDKP is APX-hard in the general case, as already known.)

If we had a PTAS for MDKP, we could use it to solve the restricted GAP. Moreover, as the above restriction works also for the general GAP (with positive values only), the PTAS could be used also for GAP. As noted, PTAS [7] applies only to the cases, where m is fixed. In our transformation an $m \times n$ matrix of an instance of GAP is transformed into an $(m + n) \times mn$ matrix of an instance of MDKP. The transformation of GAP into MDKP introduces new knapsacks for each item, and hence, the number of knapsacks in MDKP will not be fixed. Further, we can conclude the following theorem.

Theorem 4. *Unless N=NP, there cannot be an approximation preserving transformation of the restricted GAP into MDKP, in which the difference between the numbers of knapsacks is at most* $O(1)$.

Otherwise, we could use the PTAS of MDKP with fixed number of knapsacks directly as a PTAS for the restricted GAP, which in turn, is APX-hard.

4 Conclusion

We have shown that fixed IKHO is APX-hard and equal to MDKP so that the properties of MDKP apply to IKHO. Further, we characterized the relationship between GAP and MDKP.

A version of IKHO having variable clone lengths is obtained from (1)–(5) by changing c to be c_i. We also have to modify restrictions: replace restriction (4) with

$$x_i = \begin{cases} 1, & \text{when } c_i \geq 0, \\ 0, & \text{otherwise,} \end{cases}$$

and add restrictions

$$c_i \geq -1, \qquad c_i \text{ integer.}$$

In addition, the interaction function I takes c_i to its second argument, i.e. we use $I_i(k, c_i)$. Applications similar to load clipping typically give ranges for the admitted values of c_i, for each i.

The variable clone length IKHO is harder than the one with fixed clone lengths. The transformation given in Section 2 does not work anymore, because the decision of the length of a clone settles also the number of restrictions (9). We will work on the open problem of having a reasonable approximation algorithm for IKHO with variable clone length c_i. It is also open whether the variable clone length IKHO is harder to approximate than fixed IKHO.

References

1. Aho, I.: Interactive knapsacks. Fundamenta Informaticae **44** (2000) 1–23
2. Aho, I.: Notes on the properties of dynamic programming used in direct load control scheduling. Technical Report A-2000-12, University of Tampere, Dept. of Computer and Information Sciences (2000) [Online]. Available: http://www.cs.uta.fi/reports/r2000.html
3. Aho, I., Klapuri, H., Saarinen, J., Mäkinen, E.: Optimal load clipping with time of use rates. International Journal of Electrical Power & Energy Systems **20(4)** (1998) 269–280
4. Chandra, A. K., Hirschberg, D. S., Wong, C. K.: Approximate algorithms for some generalized knapsack problems. Theoretical Computer Science **3** (1976) 293–304
5. Chekuri, C., Khanna, S.: On multi-dimensional packing problems. In Proc. of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (1999) 185–194
6. Chekuri, C., Khanna, S.: A PTAS for the multiple knapsack problem. In Proc. of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (2000) 213–222
7. Frieze, A.M., Clarke, M.R.B.: Approximation algorithms for the m-dimensional 0-1 knapsack problem: Worst-case and probabilistic analyses. European Journal of Operational Research **15** (1984) 100–109
8. Garey, M., Johnson, D.: Computers and Intractability, A Guide to the Theory of NP-completeness. W. H. Freeman (1979)
9. Gill, P., Murray, W., Wright, M.: Practical Optimization. Academic Press (1981)
10. Lin, E.: A bibliographical survey on some well-known non-standard knapsack problems. Information Systems and Operations Research **36(4)** (1998) 274–317
11. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons (1990)
12. Papadimitriou, C.: Computational Complexity. Addison-Wesley (1994)
13. Srinivasan, A.: Improved approximations of packing and covering problems. In Proc. of the 27th ACM Symposium on the Theory of Computing (1995) 268–276

Model Checking Communication Protocols

Pablo Argón[1], Giorgio Delzanno[2],
Supratik Mukhopadhyay[2], and Andreas Podelski[2]

[1] IRCyN
1 rue de la Noë, BP 92101
44321 Nantes (F)
argon@lan.ec-nantes.fr
[2] Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
{supratik|podelski}@mpi-sb.mpg.de

Abstract. Brand and Zafiropulo [1] introduced the model of communicating finite-state machines to represent a distributed system connected with FIFO channels. Several different communication protocols can be specified with this simple model. In this paper we address the problem of automatically validating protocols by verifying properties such as well-formedness and absence of deadlock. Our method is based on a representation of communicating finite-state machines in terms of logic programs. This leads to efficient verification algorithms based on the ground and non-ground semantics of logic programming.

1 Introduction

Formal methods of specification and analysis are being gradually introduced to handle the increasing complexity of communication protocols. In particular, in the last years many efforts have been spent in order to identify properties independent from the functionality of a specific protocol that can be effectively verified and that can help a designer to discover uncorrect behaviors of the specification. For this purposes, Brand and Zafiropulo [1] have introduced the model of communicating finite-state machines (CFSMs) to represent a distributed system connected with FIFO channels. Such communication model can be described by using queues. Many communication protocols can be specified within this simple model. In the same paper, they also defined a number of minimal properties that *well-formed* protocol are expected to satisfy. Our aim is to develop tools for computer-aided validation, based on algorithms that can detect violations of such minimal properties. Our methodology is based on a representation of CFSMs in logic programming. This allows us to apply a number of efficient and theoretically well-founded techniques that have previously been developed for the analysis of logic programs. In particular, as we will discuss in the paper, it is possible to use the non-ground semantics of logic programs in order to implicitly represent set of states. According this, we have built a prototype for a validation tool based on the fixpoint operators of logic programs. This tool (implemented

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 160–170, 2001.

in SICStus Prolog) is an extension of a prototype we used for the verification of infinite-states concurrent programs without explicit communication [3]. In that work we used constraints to represent the evolution of the system variables. Here we will use operations on channels (queues) as "constraints" to represent the executions of the protocols. As we will discuss in the present paper, we have used our prototype to verify *well-formedness* of protocols and to detect potential deadlocks. Our analysis is performed by assuming a given *bound* on the length of the communication channels.

Though, in this paper, we are mainly concerned with reachability analysis (backward), in an extended version of this paper we will consider model-checking for Protocol Validation Logic (PVL), a logic which has sufficient expressive power to express the well-formedness properties of protocols.

Based on such experience, in the conclusions we will discuss some ideas concerning a possible application of constraints logic programming for the validation of infinite-state communication models (e.g. systems with unbounded channels or systems with arbitrary messages).

The paper is organized as follows. We first recall the main definition introduced in [1]. We then set the formal grounds for using logic programming as a validation model for protocols. This connection allows us to reduce the verification problem to reasoning about SLD-derivations. Finally, we discuss the verification of a sample protocol. The conclusion provides directions for future work (e.g. validation of communicating infinite-state machines) and comparisons of our method with some of the other approaches in literature.

Related works. There exist other works relating logic programming and verification of protocols. In [5], Fribourg and Peixoto propose the model of Concurrent Constraint Automata (CCA) to specify and prove properties of (synchronous) protocols with unbounded number of data. CCA are represented as logic programs with arithmetic constraints. Differently from [5] we use constraints that represent operations over queues and we study reachability and safety properties of the considered protocols. In [7], XSB a logic programming language based on *tabling* is used to implement an efficient local model checker for a finite-state CCS-like value-passing language. Differently from [7] in this paper we focus on the relationships between the nonground semantics of logic programs encoding protocols and well-formedness of protocols. In [2], pushdown automata are represented by logic programs and their properties are discussed using set-based costraints techniques. Communication primitives are not considered in such an approach. In [1], communicating finite state machines were used for modelling protocols. The method relied on building a tree for each process, representing all its possible executions, and on studying the interrelationships between the processes. Their approach cannot be fully automated and hence, is not comparable to our method. Another approach is based on the PROMELA [6] validation model in which one can directly specify processes, message channels and state variables. Protocols written in PROMELA can be verified using the validator of SPIN [6]. The SPIN validator performs a forward reachability analysis: it can either do an exhaustive state space search, or can be used in *supertrace* mode

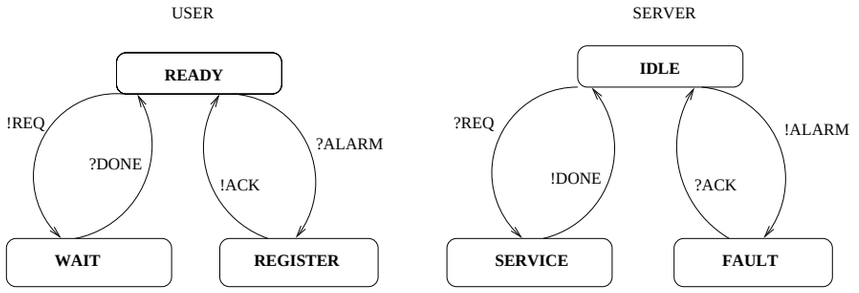


Fig. 1. Protocol *prot*.

[6], with a *bit state space* technique. SPIN explicitly represents the states of the protocol. In contrast, sets of protocol states are implicitly represented as non-ground atoms in our system. In various settings for model-checking, the implicit representation (e.g., via BDD s) has been crucial for feasibility.

2 Communicating Finite-State Machines

Communicating finite-state machines [1] have been introduced to represent a network of communicating processes. Each process is represented by a finite-state automaton. Each pair of communicating processes is assumed to be connected by a full-duplex, error free FIFO channel, i.e., communication is achieved by using a pair of queues for each pair of processes. Several different communication *protocols* can be represented within this formalism, as we will show in the rest of this section.

Protocols. Informally, a protocol specifies the possible evolution of the considered processes upon sending and receiving messages. To illustrate, consider the example in Figure 1. The process USER can pass from state **READY** to state **WAIT** sending a request (written !REQ) to process SERVER. The server, in turn, can enter in state **SERVICE** upon receipt of a request (written ?REQ), and so on. To simulate the full-duplex channels, there exist a pair of queues connecting the two processes. More formally, we have the following definition of protocols.

Definition 1 (Protocol [1]). *A protocol is a quadruple*

$$\langle \langle S_i \rangle_{i=1}^{N}, \langle o_i \rangle_{i=1}^{N}, \langle M_{ij} \rangle_{i,j=1}^{N}, succ \rangle$$

where

- N *is a positive integer.*
- $\langle S_i \rangle_{i=1}^{N}$ *are* N *disjoint finite sets* (S_i *represents the set of states for process* i*).*
- $o_i \in S_i$ *represents the initial state of process* i*.*

- $\langle M_{ij}\rangle_{i,j=1}^{N}$ *are* N^2 *disjoint finite sets with* M_{ii} *empty for all* i *(*M_{ij} *represents the messages that can be sent from process* i *to process* j*).*
- *succ is a family of partial function for each* i *and* j*, s.t.*
  - $succ(s_i, x) = s_i'$*,* $x \in M_{ij}$*, means that process* i *can move from state* s_i *to state* s_i' *sending a message* x *to process* j *(written as* $!x$ *when the indexes are clear from the context). The message* x *is enqueued in the queue of pending messages from process* i *to process* j*.*
  - $succ(s_i, x) = s_i'$*,* $x \in M_{ji}$*, means that process* i *can move from state* s_i *to state* s_i' *receiving a message* x *from process* j *(written as* $?x$*). The message* x *must be the top element of the queue containing the pending messages from process* j *to process* i*.*

As an example, in the Figure 1, $succ(\mathbf{READY}, !REQ) = \mathbf{WAIT}$, $succ(\mathbf{IDLE}, ?REQ) = \mathbf{SERVICE}$, and so on.

A *global* state is a pair $\langle S, C\rangle$ where S is a N-tuple $\langle s_1, s_2, \ldots, s_N\rangle \in \prod_N S_i$ and C is a N^2-tuple $\langle c_{11}, \ldots, c_{1N}, \ldots, c_{NN}\rangle$, each c_{ij} is a sequence of messages from M_{ij}. The message sequence c_{ij} represents the content of the channel from process i to process j. Let $\langle S^0, C^0\rangle = \langle\langle o_i\rangle_{i=1}^{N}, \langle\epsilon\rangle_{i,j=1,i\neq j}^{N}\rangle$ (where ϵ is the empty sequence of messages) be the initial state of the protocol. The possible executions are described by the reflexive and transitive closure of the relation $\longrightarrow$ which is defined as follows: $\langle S, C\rangle \longrightarrow \langle S', C'\rangle$ iff there exists i, k and x_{ik} such that one of the following two conditions hold.

- All the elements of $\langle S, C\rangle$ and $\langle S', C'\rangle$ are equal except $s_i' = succ(s_i, !x)$ for $x \in M_{ik}$ and $c_{ik}' = c_{ik}x_{ik}$.
- All the elements of $\langle S, C\rangle$ and $\langle S', C'\rangle$ are equal except $s_k' = succ(s_k, ?x)$ for $x \in M_{ik}$ and $c_{ik} = x_{ik}c_{ik}'$.

No assumptions are being made on the time a process spends in a state before sending a message and on the time a message spends in a queue before it is being delivered to its destination.

As anticipated in the introduction, in the following section we will reformulate the operational behaviour of CFSMs in terms of logic programs.

3 A Simulation in Logic Programming

The key point is to encode global states as atomic formulas. The successor functions of the CFSMs are then represented by definite clauses describing the state transitions and the modification of the queues depending on the type of communication. Our specification consists of two parts: a logic program P_{queue} which specifies the communication model, and a logic program P_{prot} which encodes the "successor" functions.

Communication model. Our aim is to consider the queue operations as special constraints added to the description of the transitions of the CFSMs. Let P_{queue} be a logic program specifying the basic operations on a FIFO channel (queue):

- $push(x, Q, Q')$: Q' is the queue that results by enqueuing x in Q;
- $pop(x, Q, Q')$: Q' is the queue that results by dequeuing x from Q;
- $empty(Q)$: Q is the empty queue.

Here we are not interested in a particular implementation of the three predicates above. We just require the implementation to be *non-directional*, i.e., all the arguments can be *input* as well as *output* arguments.

Encoding of protocols. As discussed in [2,3] a transition rule $s \to s'$ can be represented by a binary logic programs of the form $p(\tilde{s}) \leftarrow p(\tilde{s}')$ where p is nothing but a tuple-constructor and $\tilde{s}, \tilde{s}'$ represent the states s and s'. We can employ a similar idea to encode CFSMs, however, we will express communication by an additional invocation of a predicate in P_{queue}.

Formally, given a global state $\langle S, C\rangle$ s.t. $S = \langle s_1, s_2, \ldots, s_N\rangle$ and $C = \langle c_{11}, c_{12}, \ldots, c_{NN}\rangle$, we encode it as the atomic formula:

$$p(s_1, s_2, \ldots, s_N, \ldots, c_{11}, c_{12}, \ldots, c_{NN})$$

where p is a fixed predicate symbol. Consider a protocol *prot* (as in Def. 1)

$$\langle \langle S_i\rangle_{i=1}^N, \langle o_i\rangle_{i=1}^N, \langle M_{ij}\rangle_{i,j=1}^N, succ\rangle,$$

the associated logic program P_{prot} is defined by the following set of clauses:

- $p(S_1, \ldots, s_i, \ldots, Q_{ij}, \ldots, Q_{NN}) \leftarrow$
  $p(S_1, \ldots, s_i', \ldots, Q_{ij}', \ldots, Q_{NN}), push(x, Q_{ij}, Q_{ij}')$
  whenever $succ(s_i, !x) = s_i'$, $x \in M_{ij}$;
- $p(S_1, \ldots, s_i, \ldots, Q_{ji}, \ldots, Q_{NN}) \leftarrow$
  $p(S_1, \ldots, s_i', \ldots, Q_{ji}', \ldots, Q_{NN}), pop(x, Q_{ji}, Q_{ji}')$
  whenever $succ(s_i, ?x) = s_i'$, $x \in M_{ji}$,
- $init \leftarrow$
  $p(o_1, \ldots, o_n, Q_{11}, \ldots, Q_{NN}), empty(Q_{11}), \ldots, empty(Q_{NN})$

where capitalized identifiers represent free variables. As an example, the protocol of Figure 1 is encoded as the logic program of Figure 2. Now, given a ground SLD-derivation $\sigma = G_0, G_1, \ldots$ s.t. $G_0 = \leftarrow init$, let us define $\sigma^\bullet$ as the sequence

$init$	$\leftarrow p(ready, Qu, idle, Qs),$	$empty(Qu), empty(Qs).$
$p(ready, S, Qu, Qs)$	$\leftarrow p(wait, S, Qu, Qs1),$	$push(req, Qs, Qs1).$
$p(register, S, Qu, Qs)$	$\leftarrow p(ready, S, Qu, Qs1),$	$push(ack, Qs, Qs1).$
$p(wait, S, Qu, Qs)$	$\leftarrow p(ready, S, Qu1, Qs),$	$pop(done, Qu, Qu1).$
$p(ready, S, Qu, Qs)$	$\leftarrow p(register, S, Qu1, Qs),$	$pop(alarm, Qu, Qu1).$
$p(U, idle, Qu, Qs)$	$\leftarrow p(U, service, Qu, Qs1),$	$pop(req, Qs, Qs1).$
$p(U, fault, Qu, Qs)$	$\leftarrow p(U, idle, Qu, Qs1),$	$pop(ack, Qs, Qs1).$
$p(U, service, Qu, Qs)$	$\leftarrow p(U, idle, Qu1, Qs),$	$push(done, Qu, Qu1).$
$p(U, idle, Qu, Qs)$	$\leftarrow p(U, fault, Qu1, Qs),$	$push(alarm, Qu, Qu1).$

Fig. 2. The program P_{prot}.

$A_0, A_1, \ldots$ of atoms of the form $p(\tilde{t})$ obtained from σ by getting rid of the intermediate SLD-derivation steps over P_{queue}, i.e., by considering only the literals produced by resolution steps over the program P_{prot}. Then, the following result holds.

Theorem 1. *Let prot be a protocol and let P_{prot} be the encoding of prot presented above. Then, for each ground derivation σ in P_{prot} $\sigma^{\bullet}$ is a computation in prot, and, vice versa, given a computation σ in prot there exists a derivation ρ in P_{prot} s.t. $\rho^{\bullet} = \sigma$.*

This result allows us to relate the fixpoint semantics (i.e the least model) of a program with interesting properties of the encoded protocols.

4 Fixpoint Semantics and Verification

Let us recall some definitions of the fixpoint semantics of logic programs. Given a program P, let us denote by T_P the immmediate consequences operator associated to P defined over a collection I of ground atoms as follows:

$$\begin{array}{l} T_P(I) = \\ \{p(\tilde{d}) \in \mathcal{B} \mid p(\tilde{d}) \leftarrow b_1, \ldots, b_n \textit{ is a ground instance of } P, \\ \qquad\qquad b_i \in I \textit{ for } i: \ 1, \ldots, n \quad n \geq 0\}. \end{array}$$

Let $[\![P]\!]$ be the least fixpoint of T_P (i.e. the least model of P).

Now let us assume that the two programs P_{prot} and P_{queue} are defined over the same language (i.e. they are defined over the same constants and function symbols). As we we will explain later, we are mainly interested in the following problem:

> decide whether or not a given state $p(\tilde{t})$ is reachable from the initial state *init*.

This problem is undecidable for protocols with *unbounded* channels, whereas it becomes decidable whenever considering bounded channels only. We can characterize such a property by using the fixpoint semantics of P_{prot}.

Theorem 2. *A given state $p(\tilde{t})$ is reachable from the initial state init if and only if*

$$init \in [\![P_{prot} \cup [\![P_{queue}]\!] \cup \{p(\tilde{t})\}]\!].$$

The intuition is that we first compute the least model of the program P_{queue} in order to derive all the possible successful invocations of the *communication* primitives. Then, we use the obtained (potentially infinite) set of facts as an "oracle" to compute the least fixpoint of the program P_{prot}.

Such characterization can be made effective by considering the *non-ground* fixpoint semantics operator [4], defined over a collection I of *non-ground* atoms. Such an operator is defined as follows:

$$\begin{array}{l} S_P(I) = \{p(\tilde{s})\theta \mid p(\tilde{s}) \leftarrow b_1, \ldots, b_n \textit{ is a variant of } r \in P, \\ \qquad\qquad a_i \in I \textit{ for } i: 1, \ldots, n, \quad n \geq 0 \textit{ which share no variables}, \\ \qquad\qquad \theta = mgu((a_1, \ldots, a_n), (b_1, \ldots, b_n))\}. \end{array}$$

Using non-ground interpretations allows us to implicitly represent collection of states. Now, let us call $[\![P]\!]_s$ the least non-ground model of P (i.e. the least fixpoint of S_P). The following result holds: the set of ground instances of $[\![P]\!]_s$ is equal to $[\![P]\!]$. Thus, it is possible to reason both in terms of the ground and of the non-ground semantics. The non-ground characterization of the reachability problem becomes now:

$$init \in [\![P_{prot} \cup [\![P_{queue}]\!]_s \cup \{p(\tilde{t})\}]\!]_s.$$

Actually, since the least non-ground model $[\![P]\!]_s$ captures the answer-substitutions of P, such a fixpoint computation corresponds to a mixed *backward* and *forward* analysis of the program $P_{queue} \cup P_{prot} \cup \{p(\tilde{t})\}$: we compute the fixpoint of the operator $S_{P_{prot}}$ by executing the auxiliary predicates $pop, push$ and $empty$ in a top-down manner. In other words, in our prototype we use the following modified definition of S_P, where we use Q to denote an invocation of pop, $push$, or $empty$:

$$\begin{aligned} S^*_P(I) = \{p(\tilde{s})\sigma \mid\ & p(\tilde{s}) \leftarrow p(\tilde{t}), Q \textit{ is a variant of } r \in\ P, \\ & p(\tilde{t'}) \in I,\ \theta = mgu(p(\tilde{t}), p(\tilde{t'})) \\ & Q\theta \textit{ has a successful derivation with answer } \sigma\}. \end{aligned}$$

We prove the correctnss of this definition w.r.t. S_P in appendix A. The queue operations can be seen as a "special" form of constraints, and P_{queue} as the definition of an incomplete constraint solver for them.

5 Application to the Validation of Protocols

In the following we will discuss some important general properties of protocols [1]. Let us first introduce some terminology.

Well-formed protocols and stable tuples. A reception is a pair $\langle s, x\rangle$, where s is a state s and x is a message. A reception $\langle s, x\rangle$ is *specified* iff $succ(s, ?x)$ is defined. A reception $\langle s, x\rangle$ is *executable* iff there exists a reachable global state $\langle S, C\rangle$ where, for some i and k, $s\ =\ s_k$ and c_{ik} is of the form xY for some sequence Y. A protocol is said to be *well-formed* whenever each possible reception is executable if and only if it is specified. A protocol has a reception missing if a message x can arrive at a process when it is in state s, but the protocol does not specify which state should be entered upon receiving x in state s. A *N-stable tuple* is a global state in which all the channels are empty (i.e. there are no pending messages). Identifying non-specified/executable receptions and stable tuples can only be done for some classes of protocols. In particular, this can be done for protocols with bounded channels: we say that a channel from process i to process j is bounded by a constant h iff for every reachable global state $\langle S, C\rangle$, c_{ij} is a sequence of length at most h. A protocol is said to be *bounded* iff all its channels are bounded. If no such constant h exists the channel is unbounded. For a given protocol, the bound on its channels can be estimated with a method described in [1]. In the following we will restrict ourselves to consider bounded channels protocols.

Previous works in analysis of protocols [8] show that it is important for a protocol to be well-formed independently of its functionalities. Furthermore, it is important to identify stable N-tuples as they are useful for detecting synchronization losses.

Validation of protocols. Protocol validation [9] is aimed at finding any violation of the requirements of the specification. Our validation procedure is based on the logic programming model of CFSM we introduced in the previous sections.

As an example, let us consider again the protocol in Figure 1 and its encoding in logic programming in Figure 2. For this protocol it is possible to estimate an upper bound on the length of the queues during the executions. Thus, in the following discussion we limit ourselves to channels whose length is at most two.

First of all we would like to check whether *prot* is well-formed or not. Let us define a new program P_{dummy} obtained from P_{prot} by adding a *dummy* transition for each *unspecified* reception (e.g., we add the clause $p(ready, S, Qu, Qs) \leftarrow dummy, pop(ack, Qu, Qu1)$). If *dummy* is reached from *init*, i.e.,

$$init \in [\![P_{dummy} \cup [\![P_{queue}]\!] \cup \{dummy\}]\!],$$

then, there exists an executable reception which is not specified. Running the example on our prototype we have found two counterexamples which correspond to the following transitions:

$$p(wait, S, Qu, Qs) \leftarrow dummy, pop(alarm, Qu, Qu1).$$
$$p(U, fault, Qu, Qs) \leftarrow dummy, pop(req, Qs, Qs1).$$

Thus, the considered protocol is not well-formed (there exist(s) executable reception(s) which are not specified). Obviously, this result extends to the unbounded case. Let us remark an interesting aspect in the use of the non-ground semantics to verify the above property. Since we assume that the definition of *pop* is nondirectional the invocation of $pop(ack, Qu, Qu1)$ with Qu and $Qu1$ as free variables returns the more general pairs of queues (q, $q1$) such that *ack* is the top element of q and $q1$ is the result of a pop over q: in case we use lists of length at most two: $Qu = [ack], Qu1 = []$ and $Qu = [\_, ack], Qu1 = [\_]$ where _ is a variable.

Now, let us modify the protocol of Figure 1 as shown in Figure 3, i.e., by adding the following two clauses to the program in Figure 2:

$$p(wait, S, Qu, Qs) \leftarrow p(wait, S, Qu1, Qs), pop(alarm, Qu, Qu1).$$
$$p(U, fault, Qu, Qs) \leftarrow p(U, fault, Qu, Qs1), pop(req, Qs, Qs1).$$

i.e. we add *loops* to consume messages which cannot be received in the state *wait* and *fault*. To verify that the new protocol is well-formed we can use the following method. Let us define a new program P'_{dummy} obtained from P_{prot} by turning one of the clauses which specify a reception $succ(s, ?x) = s'$ into a *dummy* transition such that $succ(s, ?x) = dummy$. For instance, we define a program P'_{dummy} by turning the clause

$$p(wait, S, Qu, Qs) \leftarrow p(ready, S, Qu1, Qs), pop(done, Qu, Qu1)$$

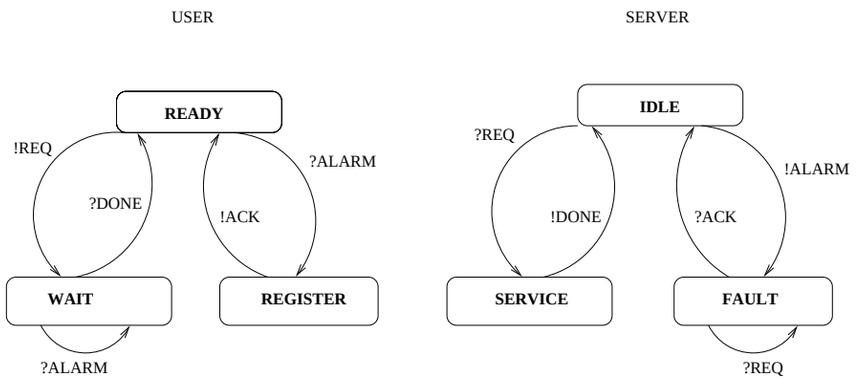


Fig. 3. Modified Protocol.

into the clause

$$p(wait, S, Qu, Qs) \leftarrow dummy, pop(done, Qu, Qu1).$$

Now, if *dummy* is reached from *init*, i.e.,

$$init \in [\![P'_{dummy} \cup [\![P_{queue}]\!] \cup \{dummy\}]\!],$$

it follows that the considered specified reception is executable. We have automatically verified such condition for each specified reception, thus proving that the new version is well-formed.

Let us now study reachability of stable tuples for the noew protocol. Following our encoding, a stable tuple can be represented by the following clause, say STABLE:

$$p(s_1, s_2, Qu, Qs) : -empty(Qu), empty(Qs)$$

where $s_1 \in \{ready, register, wait\}$, $s_2 \in \{fault, service, idle\}$. Such a tuple is reachable iff the following condition holds:

$$init \in [\![P_{prot} \cup [\![P_{queue}]\!] \cup \{stable\}]\!].$$

Executing the fix point computation shows that for the following pairs of states the corresponding stable tuple is reachable: $(wait, fault)$, $(register, fault)$, $(wait, service)$, $(ready, idle)$.

Note that the first tuple corresponds to a *deadlock* state: the two processes can only remain in states *wait* and $fault$ for ever, because no other messages can arrive. As a consequence, though well-formed, the considered protocol has a potential deadlocked execution.

6 Conclusions

The example in the previous section has already been treated by other verification methods [1].

The novelty of our approach is in the implicit representation of of set of reachable states of the protocol by non-ground atoms. This may be useful to reduce the state-space explosion. Furthermore, the encoding of protocols in logic programs makes the specification modular and flexible. For instance, we can easily modify the definition of P_{queue} in order to specify a different bound and a different type of communication. As a comparison, we have run the example on SPIN [6], detecting the deadlock state in 0.1s. We had the same execution time with our Sicstus Prolog prototype based on backward analysis. However, in SPIN we cannot submit assertions about queues (for instance to detect stable tuples) as we did in our prototype for validating the protocol. The above model can be extended in order to capture communicating infinite-state systems as we will briefly discuss hereinafter.

Channels with unbounded values. In previous work [3] we have successfully applied constraint logic programming to the verification of infinite-state concurrent systems with shared memory. Constraints are useful as symbolic representation of possibly infinite set of states of the computation. The same idea can be applied to the case of communicating machines if we extend the model of CFSMs and admit arbitrary messages to be exchanged by two processes. For instance, we could write a rule like:

$$p(P, idle, Qu, Qs) \leftarrow X > 0,\ p(P, service, Qu, Qs1),\ pop(req(X), Qs, Qs1).$$

to force the server to change state only upon receiving a positive integer. The combination of constraints, non-ground semantics, and use of forward/backward analysis might be useful to make the automatic analsyis of such complex systems effective.

Finally, let us give some more hints about potential optimizations of the validation procedure.

State explosion problem. An approach based on the exploration of the execution space of a protocol is likely to suffer from the state explosion problem even in case of bounded protocols. In case of protocol with very large queues one possible solution is the definition of an encoding of operations on queues in terms of constraints (e.g constraints over integer variables based on a positional representation of the queues elements). This is still argument of research since the encoding we have found requires non-linear constraints which are very expensive to handle. For large protocols an alternative testing method is the *random walk method* [10]. The advantage of this method is that it has minimal space requirements: to simulate a step of random walk, we need to store only the current state and a description of the actions. As a future work, we would like to incorporate the possibility of random partial state exploration in our tool, whenever exhaustive search fails due to state explosion.

Model Checking for Protocol Validation Logic. As stated in the introduction, in the extended version of this paper, we are going to introduce a Protocol Validation Logic (PVL), which is sufficiently expressive to model the well-formedness

properties of a protocol. This logic is based on the Modal Mu-Calculus and allows us to do local model-checking. The advantage is that, the state space is built on-the-fly and that the whole state space need not be explored. We defer the presentation of the logic due to lack of space.

References

1. D. Brand and P. Zafiropulo. On communicating finite-state machines. *JACM*, 30(2):323–342, April 1983.
2. W. Charatonik and A. Podelski. Set-based analysis of reactive infinite-state systems. In B. Steffen, editor, *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer-Verlag, March-April 1998. To appear.
3. G. Delzanno and A. Podelski. Model checking infinite-state concurrent systems in clp. Technical Report MPI-I-98-2-012, Max Planck Institut fuer Informatik, Saarbruecken, July 1998.
4. M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behaviour of Logic Languages. *Theoretical Comput. Sci.*, 69(3):289–318, 1989.
5. L. Fribourg and M. V. Peixoto. Concurrent constraint automata. Technical report, Laboratoire d'Informatique Ecole Nationale Superieure, Paris, May 1993.
6. G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, November 1991.
7. Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification (CAV '97)*, LNCS 1254. Springer, Haifa, Israel, July 1997.
8. H. Rudin, C. H. West, and P. Zafiropulo. Automated protocol validation: One chain of development. In *Proc Computer Network Protocols*, pages F4–1–F4–6, February 1978.
9. C. H. West. General technique for communications protocol validation. *IBM Journal of Res. and Dev.*, 22(4):393–404, July 1978.
10. C. H. West. Protocol validation in complex systems. In *Proceedings of 8th ACM Symposium on Principles of Distributed Computing*, pages 303–312, 1989.

A Proofs

Given $P = P_{prot} \cup P_{queue}$, let I be a set of non-ground facts such that $[\![P_{queue}]\!]_s \subseteq I$. We want to show that the definition of S_P^* given in Section 2, is correct w.r.t. S_P, i.e., $S_P(I) = S_P^*(I)$. Let us apply the definition of S_P: given a variant $a \leftarrow b, q$ of a rule in P (q defined over the predicates in P_{queue}) and given $c, d \in I$ (which share no variables in common) such that $d \in [\![P_{queue}]\!]_s$, $a\sigma \in S_P(I)$ whenever there exists σ s.t. $\sigma = mgu((b, q), (c, d))$. This implies that $\sigma = mgu(q\sigma', d\sigma')$ where $\sigma' = mgu(b, c)$. Since, by hypothesis, $var(d) \cap var(b, q, c) = \emptyset$, it follows that $d\sigma' = d$ and $\sigma = mgu(q\sigma', d)$. Thus, by a property of the S-semantics [4], σ is a computed answer substitution for $P_{queue} \cup \{\leftarrow q\sigma'\}$, i.e., $a\sigma \in S_P^*(I)$. The other direction can be proved in a similar way.

Pipelined Decomposable BSP Computers
(Extended Abstract)*

Martin Beran

Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, 118 00 Praha 1, the Czech Republic
beran@ms.mff.cuni.cz

Abstract. The class of weak parallel machines $\mathcal{C}_{\text{weak}}$ is interesting because it contains some realistic parallel machine models, especially suitable for pipelined computations. We prove that a modification of the bulk synchronous parallel (BSP) machine model, called decomposable BSP (dBSP), belongs to the class of weak parallel machines if its computational power is restricted properly. We will also correct some earlier results about pipelined parallel Turing machines, regarding their membership in the class $\mathcal{C}_{\text{weak}}$.

1 Introduction

The bulk synchronous parallel (BSP) model, introduced by Valiant [10], is an example of the so-called bridging models of parallel computers. A good bridging model should allow portable parallel algorithms to be developed easily and also implemented on the really existing computers efficiently. The BSP model fulfils these requirements – BSP algorithms have been designed and analyzed [4,5,7,8], and BSP implementations exist [3,6]. Hence, the BSP model is a realistic model of parallel computation. The decomposable BSP model (dBSP) is an extension of the BSP model [1]. It enables exploitation of communication locality of parallel algorithms in order to achieve an additional speedup.

Models of computation can be assigned into machine classes according to their computational power. The first machine class [9] contains the Turing machine (TM) and other sequential models, e.g., RAM. Given an algorithm, its time complexity differs only up to a polynomial factor when executed on different models from the first class. The second machine class [12] is the class of massively parallel computers, e.g., PRAM. The time complexity of a problem on computers from the second class is polynomially related to the space complexity of the Turing machine. Machines from the second class offer an exponential speedup in comparison with the first class but, unfortunately, they are infeasible if physical laws of nature are taken into account. Hence we would like to have a class of parallel computers computing faster than members of the first machine class, yet being realistic, i.e., physically feasible. One such class is the class of weak parallel machines $\mathcal{C}_{\text{weak}}$, defined by Wiedermann in [13]. Members of class

* This research was partially supported by the GA ČR grant No. 201/00/1489.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 171–180, 2001.

$\mathcal{C}_{\text{weak}}$ provide fast pipelined computation, i.e., fast solution of many instances of the same problem. A typical feature of weak parallel models is restricted communication. For example, in a single step of a parallel Turing machine, information can be transferred from a tape cell to its immediate neighbour only. In dBSP, any communication pattern is allowed, but local communication (limited to small clusters of processors) runs faster and is therefore preferred. This characteristic – communication locality – is exhibited by many really existing parallel computers. This observation supports the hypothesis that weak parallelism of $\mathcal{C}_{\text{weak}}$ members could be a good characterization of realistic parallel computers.

In this paper, we will answer the question whether dBSP computers belong to the class of weak parallel machines. In Sect. 2, we will define the class $\mathcal{C}_{\text{weak}}$ and present a representative weak parallel machine – the pipelined parallel Turing machine (PPTM). We will correct some claims from [13] about the relation of PPTMs and class $\mathcal{C}_{\text{weak}}$. Section 3 contains an analysis of the membership of the dBSP model in the class of weak parallel machines. Due to space constraints, some details are omitted in this paper, but can be found in the author's doctoral thesis, available as [2].

2 Class of Weak Parallel Machines

In this section, we will define the class of weak parallel machines and study pipelined Turing machines. This provides a base for a definition and analysis of pipelined dBSP computers in the following section.

A pipelined computation processes a sequence of N inputs (instances) of the same problem. We will usually assume that all inputs are of the same size n. *Time complexity* of a pipelined computation $T(n)$ is the time needed to process one input, i.e., the time between the first symbol of input i is read and the output of problem instance i is produced. *Space complexity* is the total memory consumed by all instances. *Period* is the time between beginnings of reading two subsequent inputs or (equivalently) between ends of writing two subsequent outputs. All three complexity measures are defined as depending only on size n of a single input, not on the number of inputs N. Efficient pipelined computers belong to the class of weak parallel machines [13]. Its definition resembles the definition of the second machine class [12] of parallel computers.

Definition 1. *The* class of weak parallel machines $\mathcal{C}_{\text{weak}}$ *contains machines with their periods $P(n)$ polynomially equivalent to the space complexity $S(n)$ of a deterministic sequential Turing machine solving the same problem, i.e., $P(n) = O(S^k(n))$ and $S(n) = O(P^l(n))$ for some positive constants k and l.*

A (non-pipelined) 1-dimensional 1-tape *parallel Turing Machine (PTM)* [13] is based on a nondeterministic sequential Turing machine (NTM). The computation starts with only one processor (the head with the control unit realizing the transition relation is called the processor). If the NTM would do a nondeterministic choice of performing one of instructions $i_1, i_2, \ldots, i_b$, the PTM creates $b-1$ new processors and each of b processors executes a different instruction from

the alternatives. Then all the processors work independently, but they share the same tape. The processors run synchronously, performing one step per time unit. It is forbidden for several processors to write simultaneously different symbols into the same tape cell.

A *pipelined parallel Turing machine (PPTM)*, defined in [13], has, in addition to the working tape, a two-dimensional read-only input tape, and one-dimensional write-only output tape. The i-th input word w_i is written from the beginning of the i-th row of the input tape. The machine prints its i-th output to the i-th cell of the output tape. The input is read in order $w_1, w_2, \ldots$ and the output is printed in the same order. The number of steps made by the PPTM between reading the first symbol of w_i and printing the i-th output depends only on the length n of w_i. The PPTM halts after printing the last output.

The PPTM is *uniform* if and only if it eventually starts cycling (repeating the same configuration) with period $P(n)$ on a sufficiently long sequence of identical inputs.

The following two lemmas providing mutual simulation algorithms between sequential and pipelined parallel TMs can be found in [13].

Lemma 1. *A Turing machine computing in time $T(n)$ and space $S(n)$ can be simulated by a PPTM with period $P(n) = O(S(n))$, time $O(T(n))$ and space $O(T(n))$.*

The idea is to store several instances (each occupying $S(n)$ cells) on the working tape. In each parallel step, one step of the sequential algorithm is simulated for each instance and the tape content is shifted by one cell towards the end of the tape.

Lemma 2. *Let $\mathcal{M}$ be a uniform PPTM computing with period $P(n)$. Then there exists a sequential Turing machine $\mathcal{M}'$ with a separate input tape computing in space $S(n) = O(P(n))$, which gets a configuration of $\mathcal{M}$ on its input tape and checks that $\mathcal{M}$ returns into the same configuration after $P(n)$ steps.*

The simulation algorithm uses the fact that the content of cell i after $P(n)$ steps depends only on the initial state of cells $i - P(n), \ldots, i + P(n)$. Thus only $O(P(n))$ cells has to be stored on the working tape of $\mathcal{M}'$ at any given time.

Lemmas 1 and 2 seem to support the claim of [13] that a uniform PPTM is a member of class $\mathcal{C}_{\text{weak}}$. But we refute this claim by the following theorem and its corollary.

Theorem 1. *Every sequential Turing machine (TM) algorithm that halts on each input can be simulated by a uniform PPTM algorithm with period $P(n) = O(n)$.*

Proof. The PPTM simulation algorithm consists two phases. In the precomputation phase, results of TM computations are computed and stored on the working tape for all possible $2^{O(n)}$ inputs of length n. During the computation phase, a simple table lookup is performed for each input. An input is compared (sequentially) to each table element and if the input matches, the corresponding output is used. A single comparison of one input takes linear time and there are exponentially many possible inputs, but exponentially many comparisons (on different inputs) are done in parallel. Hence, the period is $O(n)$. □

Corollary 1. *The uniform PPTM is not a model belonging to the class of weak parallel machines.*

Proof. It suffices to prove that there is a uniform PPTM computing with period $P(n)$ such that it cannot be simulated by a Turing machine in space $O(P^k(n))$ for any constant $k > 0$. Consider a problem with TM space complexity $S(n) = \omega(n^k)$ for all $k > 0$. It means that there is no TM which solves the problem in space less than $S(n)$. A TM algorithm for this problem can be simulated by a PPTM with period $P(n) = O(n)$ according to Theorem 1. A simulation of the PPTM on the TM in space $O(P^k(n)) = O(n^k)$ yields a contradiction to the assumption about $S(n)$. □

Now it is clear that the power of a uniform PPTM must be restricted to become a weak parallel machine. We define two such restrictions: limited and strictly pipelined PTMs.

Definition 2 (Limited PPTM). *A uniform PPTM with period $P(n)$ is called* limited PPTM *if and only if the computation cycle*[1] *contains configuration C, such that there is a sequential Turing machine algorithm working in space $S(n) = O(P^k(n))$ for some constant $k > 0$, which gradually prints the cells of C from left to right*[2] *to its output tape, given the input of size n.*

Theorem 2. *Limited PPTM is a member of the class of weak parallel machines.*

Proof. A simulation of a space $S(n)$ sequential computation on a PPTM with period $P(n) = O(S(n))$ is provided by Lemma 1. The algorithm is limited, because the PPTM tape contains configurations of a space bounded sequential machine. Therefore parts of the tape corresponding to individual instances can be generated by the sequential algorithm in space $O(S(n))$.

The reverse simulation uses the algorithm from Lemma 2 to check that the PPTM returns into the same configuration after a period. The special configuration C (required to exist by the definition of the limited PPTM) is tested, because it can be generated in space $S(n) = O(P^k(n))$. If a new cell is needed by the cycle testing algorithm, it is obtained by running the C generating algorithm until one new cell is printed. □

Definition 3 (Strictly pipelined PTM). *A uniform PPTM is called strictly pipelined, if and only if its working tape can be partitioned into subsets of cells pertinent to individual instances (each cell is pertinent to at most one instance) and no information is ever shared by several instances (passed from one instance to another).*

The above definition is only informal, see [2] for more precise formal and quite technical definition.

Theorem 3. *Strictly pipelined PPTM is a member of the class of weak parallel machines.*

[1] which the computation is required to enter due to uniformity

[2] assuming the working tape starts in the left and stretches to the right

Proof. During one cycle (induced by uniformity), a new input is read and computation of a new instance begins. A new instance does not know how many instances have already started before, therefore it must enter the cycle. Otherwise the cycling behavior would not be guaranteed. During one period, i.e., $P(n)$ steps, the instance can allocate up to $O(P(n))$ tape cells. As cycling is required, these cells must be freed again in subsequent $O(P(n))$ steps for usage by the next instance. At the same time, only $O(P(n))$ new cells can be allocated. The instance cannot utilize more than $O(P(n))$ cells at any time and hence the computation of one instance can be simulated by a nonpipelined PTM in space $O(P(n))$. Transformation to a sequential TM algorithm working in space $S(n) = O(P^k(n))$ for some constant $k > 0$ can be found in [13].

The reverse simulation of a space $S(n)$ Turing machine on the pipelined PTM with period $P(n) = O(S(n))$, as described in Lemma 1, satisfies the restriction of the strictly pipelined PTM. □

3 Pipelined Decomposable BSP

The *Bulk Synchronous Parallel Computer* [8,10] consists of p processors with local memories. Every processor is essentially a RAM. The processors can communicate by sending messages via a router (some communication and synchronization device). The computation runs in supersteps, i.e., the processors work asynchronously, but are periodically synchronized by a barrier. A superstep consists of three phases: computation, communication, and synchronization. In the computation phase, the processors compute with locally held data. The communication phase consists of a realization of so-called h-relation, i.e., processors send point-to-point messages to other processors so that no processor sends nor receives more than h messages. Data sent in one superstep are available at their destinations from the beginning of the next superstep. In the final phase of each superstep, all the processors perform a barrier synchronization.

Performance of the router is given by two parameters: g (the ratio of the time needed to send or receive one message to the time of one elementary computational operation – the inverse of the communication throughput) and l (the communication latency and the synchronization overhead). Both g and l are non-decreasing functions of p. If a BSP computation consists of s supersteps and the i-th superstep is composed of w_i computational steps in every processor and of h_i-relation, then the time complexity of the computation is defined by $T = \sum_{i=1}^{s}(w_i + h_i g + l) = W + Hg + sl$, where $W = \sum_{i=1}^{s} w_i$ and $H = \sum_{i=1}^{s} h_i$. We always assume the logarithmic cost of individual instructions.

A *Decomposable Bulk Synchronous Parallel Computer* with p processors and communication parameters g and l – denoted $dBSP(p, g, l)$ – is a $BSP(p, g, l)$ computer with some modifications and additional instructions `split` and `join`.

During a superstep, a processor can issue instruction `split(i)`, where $i \in \{0, \ldots, p-1\}$. If a processor calls `split`, then all other processors must call `split` exactly once in the same superstep. Beginning from the next superstep, the machine is partitioned into clusters (submachines) $C_1, \ldots, C_{cl}$, where cl is

the number of different values of i in instructions `split(i)`. Processors which specified the same i in the split instruction belong to the same cluster, while different values of i imply different clusters. Communication is restricted to processors in the same cluster. Sending a message from a processor in one cluster to a processor in another cluster is forbidden. Cluster C_i can be further recursively decomposed using instruction `split(j)`, which assigns the calling processor to subcluster $C_{i,j}$.

Instruction `join` called by a processor cancels the last level of decomposition which involved the calling processor. All the processors in all the sibling – originated from the same `split` operation – clusters must call `join` exactly once in the same superstep. Only one level of join is allowed in a single superstep. After a join, the machine can be decomposed again.

The *time complexity* of a dBSP computation is $T = \sum_{i=1}^{s}(w_i + h_i g(p_i) + l(p_i)) = W + \sum_{i=1}^{s}(h_i g(p_i) + l(p_i))$, where p_i is the size (number of processors) of the largest non-partitioned cluster existing in superstep s. An important property of dBSP is that partitioning into small clusters causes small values of $g(p_i)$ and $l(p_i)$ and thus faster communication.

A PPTM is constructed from a non-pipelined PTM by adding input and output tapes capable of holding many inputs and outputs. Similarly, input and output arrays of registers can be added to the dBSP model and pipelined dBSP is obtained. A *pipelined dBSP computer (pipe-dBSP)* is a dBSP computer augmented with a pair of two-dimensional arrays I and O of read-only input and write-only output registers. The j-th value of the i-th input word w_i can be read from register $I_{i,j}$ and the j-th value of the i-th output is written to register $O_{i,j}$. Special registers *i_sel* and *o_sel* are used to choose the particular input or output, i.e., only $I_{i\_sel,j}$ and $O_{o\_sel,j}$ can be accessed. Although logarithmic cost is used, time of an I/O instruction depends only on the value of j, not on *i_sel* or *o_sel*. The initial value of registers *i_sel* and *o_sel* is 0 and the only operation permitted for them is incrementing their values by one. The increment is performed in unit time. This provides access to inputs in order $w_1, w_2, w_3, \ldots$ with reading later inputs (with large indices) as fast as reading w_1. Only the first $O(n^k)$ processors may access the I/O registers, where $k > 0$ is a constant.

Individual instructions of a BSP computer can take different numbers of time units and a BSP machine computes – and thus can read input or write output – only during a part of each superstep. Moreover, we want to be able to comprise several periods in a single superstep. Thus we slightly relax the definition of time and period of a pipelined computation.

Definition 4. *A computation of a pipelined machine on inputs of length n runs in* time $T(n)$ *and with* period $P(n)$ *if and only if the N-th output is printed after at most $T(n) + (N-1)P(n)$ time units since the beginning of the computation.*

We also relax the notion of uniformity. We say that a pipelined dBSP machine is *uniform* either if it is uniform in the original sense (see p. 173), i.e., cycling with period $P(n)$, or if it originated from such a cycling algorithm by packing several periods into a single superstep[3].

[3] This transformation only spares synchronization time, which is $l(p)$ per superstep.

Pipelined dBSP is a powerful model, because an analog of Th. 1 and Cor. 1 can be proved for it. The proofs are omitted, because they exploit the same technique of building a table of all possible input/output pairs in a precomputation phase and simple table lookups during the rest of the computation.

Theorem 4. *Every sequential RAM algorithm that halts on each input can be simulated by a pipelined dBSP computer with period* $P(n) = O(n^2)$.

Corollary 2. *A pipelined dBSP computer is not a member of the class of weak parallel machines.*

No parallelism is required, only a single processor suffices to obtain the fast pipelined algorithm. The period is $O(n^2)$, not $O(n)$ as in the PPTM version, because n input bits can be distributed in $O(n)$ input registers. Contents of these registers must be multiplied together to get an index to the lookup table. The multiplication takes time $O(n^2)$ in the worst case.

Similarity of Theorems 1 and 4 indicate that some restriction of the pipelined dBSP model would be necessary in order to become a weak parallel machine. We define limited and strictly pipelined dBSP like we did for PPTMs in Sect. 2. Then we prove that these restricted dBSPs belong to the class of weak parallel machines.

Restriction of a limited pipe-dBSP machine is based on bounded sequential space needed for computing a configuration from the cycle required by uniformity. Thus, a space-efficient sequential simulation is made possible.

Definition 5 (Limited pipe-dBSP). *A pipe-dBSP machine with period* $P(n)$ *is called a limited pipe-dBSP machine if and only if it is uniform and its computation cycle contains a configuration* C*, such that there is a RAM algorithm working in space* $S(n) = O(P^k(n))$ *for some constant* $k > 0$*, which computes the contents of the* i*-th register of the* j*-th processor in configuration* C*, given the input of size* n *and numbers* i*,* j*.*

Theorem 5. *Limited pipe-dBSP with* $l(p) = O(p^k)$ *for some* $k > 0$ *is a member of class* $\mathcal{C}_{\text{weak}}$.

Proof. Limited pipe-dBSP computation with period $P(n) = O(S^k(n))$:
Consider a sequential RAM algorithm with time complexity $T(n)$ and space complexity $S(n)$. Each of $p = T(n)/S(n)$ dBSP processors stores $q = T^k(n)$ instances in its local memory. A superstep comprises computing the next $T(n)/p$ steps of the sequential algorithm for all qp unfinished instances. Then every processor sends its instances to the next processor. The oldest q instances are finished and their output written to the output registers. Simultaneously, q new inputs are read from the input registers.
Every processor uses $O(qS(n))$ registers. The maximum number of bits in an address is therefore $\log(qS(n)) = \log q + \log S(n)$ instead of $\log S(n)$ in the RAM algorithm. Hence an instruction can be slowed down relatively to its RAM counterpart by factor $\log q$. Communication is done in two phases. The computer is twice repartitioned into pairs of processors. In the first phase,

each processor with even *pid* sends its data to the successive odd processor. Data from odd to even processors are sent in the second phase.

From time needed by a single computational superstep and its related communication supersteps, constituting together q periods $qP(n) = T(n)/p \cdot q \log q + 2l(p) + S(n)q \log q + \log p + qS(n)g(2) + 2l(2)$, and after substitutions for p and q, we obtain the length of the period $P(n) = O(S(n) \log T(n))$. A well known fact about RAM time and space complexities $T(n) = 2^{O(S(n))}$ yields $P(n) = O(S^2(n))$.

The algorithm cycles with cycle length $qP(n)$. Every configuration consists of RAM configurations in different stages of processing and therefore can be generated by a sequential algorithm in space $S(n)$. Hence the simulation can be performed by a limited pipe-dBSP.

Simulation of a limited pipe-dBSP computer on a RAM:

The RAM machine generates configuration C which exists in the cycle by definition. Then it simulates one period and checks that dBSP returns to C. The largest value manipulated – and thus the highest addressable register and the number of processors – during the period is bounded by $2^{O(P(n))}$ due to the logarithmic cost. A single dBSP period can be simulated by a PRAM in time $O(P^l(n))$ using additional p^2 processors for handling h-relations. The PRAM is a member of the second machine class [11] and thus its time ($O(P^l(n))$ in our case) is polynomially related to sequential space $O(P^{l'}(n))$. The corresponding simulation algorithm is started in configuration C, which can be generated in space $O(P^k(n))$. Hence the total space needed is $O((P(n))^{\max\{k,l'\}})$. □

Definition 6 (Strictly pipelined dBSP). *A pipelined dBSP computer with period $P(n)$ is strictly pipelined, if and only if its registers can be partitioned into subsets of registers pertinent to individual instances, no information is ever shared by several instances, and at most $O(P^k(n))$ work[4], for some constant $k > 0$, is done on registers pertinent to a single instance during a single period.*

Theorem 6. *Strictly pipelined dBSP with $l(p) = O(p^k)$ for some $k > 0$ is a member of class $\mathcal{C}_{\text{weak}}$.*

Proof. Strictly pipelined dBSP computation with period $P(n) = O(S^k(n))$:

The algorithm from the proof of Th. 5 can be used, because it is strictly pipelined. Each instance is processed in a separate subset of registers and no value computed by an instance is reused during computation of another instance.

Simulation of a strictly pipelined dBSP computer on a RAM:

The strictly pipelined dBSP machine starts cycling with period $P(n)$, because it is uniform. A new input word is read and computation of the new instance begins during the cycle. The new instance has no information about the other instances, thus it must enter the cycle to ensure cyclic behavior.

[4] total computational steps performed by all processors working on this instance

Limited work allows for allocation of space only up to $O(P^k(n))$ bits of memory. This memory must be made free for the next instance during the next period. Hence, total memory consumed by a single instance is $O(P^k(n))$. The simulation algorithm takes the input and progressively executes all periods (cycles) with this input until the output is produced. Simulation of a single period uses the same method as in the previous proof and utilizes $O(P^{k'}(n))$ RAM registers. As no information from the registers pertinent to other instances can be utilized, only $O(P^k(n))$ registers pertinent to the single instance being processed have to be stored from one cycle to the next. Other registers can be assumed to contain 0. □

4 Conclusion

Our goal was to analyze pipelined computations on the dBSP machine model. To do it, we have used a framework of weak parallel machines. We have presented a definition of class $\mathcal{C}_{\text{weak}}$. Pipelined parallel Turing machines – defined and analyzed in [13] – are already known candidates for being weak parallel machines. In paper [13], membership of PPTM in class $\mathcal{C}_{\text{weak}}$ was claimed. The proof was based on ideas mentioned in Lemmas 1 and 2. In this paper, we identified a problem in that proof: ability to simulate a single period in a space-efficient way does not yield simulation of the whole computation in small space. As we have shown in Cor. 1, a major part of computation can be performed in its initial phase, before the machine starts cycling. Hence, a uniform PPTM is not a weak parallel machine, but a suitably restricted PPTM becomes a member of $\mathcal{C}_{\text{weak}}$. We have defined two possible restrictions – limited and strictly pipelined PPTMs.

The second type of machine models studied in this paper are pipelined Decomposable BSP computers. The situation in this case is similar to PPTMs. A general dBSP machine is too powerful (even without any parallelism, i.e., with only one processor used) and therefore is not a weak parallel machine. Two modifications of the pipelined dBSP model – limited and strictly pipelined dBSPs – have been introduced and their membership in $\mathcal{C}_{\text{weak}}$ proved. They are based on the same ideas as the corresponding limited and strictly pipelined PPTMs. This result indicates that dBSP is a viable realistic model of parallel computation. We have used dBSP and not BSP machines, because the algorithm for simulation of $S(n)$-bounded sequential computation in period $P(n)$ needs fast communication. A dBSP machines can be partitioned into clusters of size 2, thus the time of an h-relation is $hg(2) + l(p)$ instead of much larger BSP time $hg(p) + l(p)$. Note that $h = T^k(n)S(n)$ is large in our case.

An important (but not surprising) observation is that if some data are computed once and then reused during a pipelined computation, then the total time needed to process all instances can be made significantly smaller. It could be interesting to study relations of pipelined computation and another recently emerging paradigm of computing – interactive machines. Like a pipelined computer, an interactive machine processes a (potentially infinitely) long sequence of

input data, produces corresponding output data, and is able to store information in its internal memory for later use.

References

1. M. Beran. Decomposable bulk synchronous parallel computers. In *Proceedings of SOFSEM '99*, *LNCS 1725*, pp. 349–359. Springer-Verlag, 1999.
2. M. Beran. Formalizing, analyzing, and extending the model of bulk synchronous parallel computer. Technical Report V-829, Institute of Computer Science, Academy of Sciences of the Czech Republic, December 2000. `http://www.cs.cas.cz/research/library/reports_800.shtml`
3. O. Bonorden, B. Juurlink, I. von Otte, and I. Rieping. The Paderborn University BSP (PUB) library - design, implementation and performance. In *Proceedings of 13th International Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP)*, San Juan, Puerto Rico, April 1999.
4. A. V. Gerbessiotis and C. J. Siniolakis. Primitive operations on the BSP model. Technical Report PRG-TR-23-96, Oxford University Computing Laboratory, Oxford, October 1996.
5. A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing*, 22:251–267, 1994.
6. J. M. D. Hill, W. F. McColl, D. C. Stefanescu, M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, and T. Tsantilas and R. Bisseling. BSPlib: The BSP programming library, 1998. BSPlib reference manual with ANSI C examples. `http://www.bsp-worldwide.org/implmnts/oxtool/`
7. B. H. H. Juurlink and H. A. G. Wijshoff. Communication primitives for BSP computers. *Information Processing Letters*, 58:303–310, 1996.
8. W. F. McColl. Bulk synchronous parallel computing. In John R. Davy and Peter M. Dew, editors, *Abstract Machine Models for Highly Parallel Computers*, pages 41–63. Oxford University Press, 1995.
9. C. Slot and P. van Emde-Boas. On tape versus core; an application of space efficient perfect hash function to the invariance of space. In *Proceedings of STOC'84*, pages 391–400, Washington D.C., 1984.
10. L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.
11. P. van Emde Boas. Machine models and simulations. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 1–66. Elsevier Science Publishers, Amsterdam, 1990.
12. P. van Emde Boas. The second machine class, model of parallelism. In J. van Leeuwen, J. K. Lenstra, and A. H. G. Rinnooy Kan, editors, *Parallel Computers and Computations, CWI Syllabus*, volume 9, pages 133–161. Centre for Mathematics and Computer Science, Amsterdam, 1985.
13. Jiří Wiedermann. Weak parallel machines: A new class of physically feasible parallel machine models. In I. M. Havel and V. Koubek, editors, *Mathematical Foundations of Computer Science 1992, 17th Int. Symposium (MFCS'92)*, *LNCS 629*, pp. 95–111. Springer-Verlag 1992.

Quantum versus Probabilistic One-Way Finite Automata with Counter

Richard Bonner[1], Rūsiņš Freivalds[2], and Maksim Kravtsev[2]

[1] Department of Mathematics and Physics,
Malardalen University, Vasteras, Sweden*.
rbr@mdh.se,
[2] Department of Computer Science,
University of Latvia, Raina bulv. 19, Riga, Latvia**.
rusins@cclu.lv
maksims@batsolf.lv

Abstract. The paper adds the one-counter one-way finite automaton [6] to the list of classical computing devices having quantum counterparts more powerful in some cases. Specifically, two languages are considered, the first is not recognizable by deterministic one-counter one-way finite automata, the second is not recognizable with bounded error by probabilistic one-counter one-way finite automata, but each recognizable with bounded error by a quantum one-counter one-way finite automaton. This result contrasts the case of one-way finite automata without counter, where it is known [5] that the quantum device is actually less powerful than its classical counterpart.

1 Introduction

As a general background for the present paper we refer to Gruska's monograph [4]. Analogues to well known, classical computing devices such as finite automata and Turing machines have been defined and studied, often with surprising results. Thus, for example, the quantum Turing machine introduced by Deutsch [3] proved to have the same computational power as the classical deterministic one. The 2-way quantum automata of Kondacs and Watrous [5], on the other hand, recognize all regular languages and some non-regular languages such as 0^n10^n, and are thus more powerful than their 2-way probabilistic counterparts. The same authors showed, however, that the opposite relation holds for the quantum one-way finite automata, which can recognize only a proper subset of regular languages. It cannot recognize for example the language $\{0,1\}^*1$.

In the paper we show that the situation changes again if the one-way automaton is supplied with a counter; the quantumization of automata with counter was recently considered by Kravtsev [6]. We presently offer two languages recognizable by a quantum one-counter one-way finite automaton, one of which fails to

* Supported by the ML2000 project sponsored by the Swedish Institute.
** Research supported by the Latvian Council of Science, grant 01.0354; European Commission , contract IST-1999-11234; Swedish Institute, project ML2000

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 181–190, 2001.

be recognizable deterministically and the other probabilistically with bounded error, by classical one-way one-counter automata.

2 Definitions

Automata. See Gruska [4] for general and Condacs and Watrous [5] for technical background material on quantum finite automata. The quantum automata with counter which we consider here, were considered by Kravtsev [6], which we also refer to for formal definitions. Here, we recall the general ideas of the model only.

To begin with, a *deterministic* one-way automaton with counter (1D1CA) is specified by a finite *input alphabet* Σ, a finite set Q of *states* with a singled out *initial state*, two disjoint sets of *accepting* and *rejecting* states Q_a and Q_r, a *counter* which is allowed to hold an arbitrary integer, and a *transition function* δ updating the state and the counter at each step of computation as input letters are read from the tape. The value of the function δ depends on the current state, the read letter and whether the counter is zero or non-zero, but not on the exact value of the counter. The counter is set to 0 at the beginning of computation; it is allowed to change at each update by at most one. The automaton processes each letter of the input word precisely once until the last letter of the word is reached. If the automaton is then in an accepting state, the word is considered accepted, while if in a rejecting state, the word is rejected. Formally δ is a mapping $Q \times \Sigma \times S \to Q \times D$, where the elements of $S = \{0, 1\}$ indicate whether the counter is 0 or not, and $D = \{-1, 0, 1\}$ contains the number by which the counter's value may change.

A *probabilistic* version of this type of automaton (1P1CA) is obtained in the usual way, essentially by considering the states of the deterministic case as point masses in a larger set of probability measures, to which the evolution of the automaton is then extended. Formally δ is now defined as a mapping $\delta\colon Q \times \Sigma \times S \times Q \times D \to R^+$, where $\delta(q, \sigma, s, q', d)$ describes probability of getting from state q and some value of the counter (zero when s=0 and nonzero when s=1) by reading letter σ , to state q' and change the value of the counter by d. δ should satisfy the following condition: $\sum_{q' \in Q, d \in D} \delta(q, \sigma, s, q', d) = 1$ for each $q \in Q$, $\sigma \in \Sigma$, $s \in \{0, 1\}$.

Finally, for the one-way *quantum* one-counter automaton (1Q1CA), the alphabet is supplemented by end markers $\vdash$ and $\dashv$,and the transition function $\delta\colon Q \times \Gamma \times S \times Q \times D \to C$, is complex-valued and assumed to satisfy certain well-formedness conditions. Here $\Gamma = \Sigma \cup \{\vdash, \dashv\}$. The definitions of the counter and the actions on it remain unchanged from the classical version. Language recognition works then roughly as follows. For each letter σ of a word extended by the end markers, two actions are performed:
1) a unitary operator U_σ is applied to the current state of automaton, and
2) the certain measurement is applied to the resulting state.
The measurement leads with certain probabilities that are determined by resulting state to one of three outcomes - the word is either rejected or accepted (and

computation stops in both cases) or the computation proceeds with the next letter.

More formally we describe the dynamics of a quantum automaton with counter as the evolution of a quantum system defined as follows:

Quantum state. The state of 1Q1CA is considered as normalized vector of the Hilbert space $l_2(Q \times \mathbb{Z})$ with basis vectors $|q,k\rangle$ corresponding to the possible configurations of the classical automaton - the combinations of state $q \in Q$ and counter value $k \in \mathbb{Z}$. So the state of a quantum automaton is a linear combination or superposition $\sum_{ik} a_{ik}|q_i,k\rangle$ of basis vectors, where the modules of complex amplitudes $a_{ik} \in C$ sum up to one $\sum_{ik} \|a_{ik}\|^2 = 1$.

Evolution. Operator U_σ is defined in terms of δ on a vector $|q,k\rangle$

$$U_\sigma |q,k\rangle = \sum_{q',d} \delta(q,\sigma,sign(k),q',d)\,|q',k+d\rangle\,,$$

where sign$(k) = 0$ if $k = 0$ and 1 otherwise. U_σ is extended by linearity to map any superposition of basis states. To ensure that U_σ is unitary δ should satisfy the certain conditions of well-formedness. In this paper we consider only the so called simple one-way one-counter quantum automata for which well-formedness conditions are as follows [6]: for each $\sigma \in \Gamma, s \in \{0,1\}$ there is a linear unitary operator $V_{\sigma,s}$ on the inner product space $l_2(Q)$ and a function $B\colon Q,\Gamma \to \{-1,0,1\}$ such as for each $q,q' \in Q$

$$\delta(q,\sigma,s,q',d) = \begin{cases} \langle q'|V_{\sigma,s}|q\rangle & \text{if } B(q',\sigma) = d \\ 0 & \text{else} \end{cases}$$

where $\langle q'|V_{\sigma,s}|q\rangle$ denotes the coefficient of $|q'\rangle$ in $V_{\sigma,s}\,|q\rangle$.
In other words δ is defined according to the following:
1) the set of finite unitary matrices, that map states from Q to states from Q, different for each input letter and zero or not zero counter value.
2) the change of the value of the counter is determined only by the read letter and the state from Q, that automaton moves in.

Measurement. An observation of the superposition $\sum_{ik} a_{ik}|q_i,k\rangle$ results in an acceptance of the word with probability $p_a = \sum_{ik} a_{ik}^2$ where $q_i \in Q_a$, rejection of the word with probability $p_r = \sum_{ik} a_{ik}^2$ where $q_i \in Q_r$. Computation halts in these cases. Otherwise the computation continues with the renormalized state $\sum_{ik} a'_{ik}|q_i,k\rangle$ where $q_i,q_j \in Q\backslash(Q_a \bigcup Q_r)$

The total probability of accepting a word is a sum of probabilities to accept the word at each step. The total probability to reject a word is a sum of probabilities to accept the word at each step.

Languages. Let Σ be a finite alphabet. For $S \subseteq \Sigma$ define the 'projection' map $\pi_S : \Sigma^* \to S^*$ which acts on words over Σ by forgetting all letters not in S. When S is given explicitly as $\{\sigma_1,...,\sigma_n\}$, we write $\pi_{\sigma_1,...,\sigma_n}$. Note, in particular, that the length $|\pi_\sigma(x)|$ counts the occurrences of a letter $\sigma \in \Sigma$ in a word $x \in \Sigma^*$.

The language L_1. Consider the alphabet $\Sigma = \{0, 1, 2, \flat, 3, 4, 5, \sharp\}$ with two special symbols $\flat$ and $\sharp$. Let $\Sigma^\flat = \{0, 1, 2, \flat\}$ and $\Sigma^\sharp = \{3, 4, 5, \sharp\}$. We define languages $L_1^\flat$, $L_2^\flat$, $L_3^\flat$ as follows

$$L_1^\flat = \{w \in \Sigma^* : \pi_{\Sigma^\flat}(w) = x\flat y, x \in \{0,1\}^*, y \in \{2\}^* \text{ and } |\pi_0(x)| = |\pi_1(x)|\}$$

$L_2^\flat = \{w \in \Sigma^* : \pi_{\Sigma^\flat}(w) = x\flat y, x \in \{0,1\}^*, y \in \{2\}^*$ and
$|\pi_0(x)| = |\pi_1(x)| + |\pi_2(y)| \quad |\pi_2(y)| > 0\}$
$L_3^\flat = \Sigma^*/(L_1^\flat \cup L_2^\flat)$
Similarly, for the alphabet $\Sigma^\sharp$

$$L_1^\sharp = \{w \in \Sigma^* : \pi_{\Sigma^\sharp}(w) = x\sharp y, x \in \{3,4\}^*, y \in \{5\}^* \text{and } |\pi_3(x)| = |\pi_4(x)|\}$$

$L_2^\sharp = \{w \in \Sigma^* : \pi_{\Sigma^\sharp}(w) = x\sharp y, x \in \{3,4\}^*, y \in \{5\}^*$ and
$|\pi_3(x)| = |\pi_4(x)| + |\pi_5(y)| \quad |\pi_5(y)| > 0\}$
$L_3^\sharp = \Sigma^*/(L_1^\sharp \cup L_2^\sharp)$
The language L_1 is formally defined as follows:

$$L_1 = (L_1^\flat \cap L_2^\sharp) \cup (L_2^\flat \cap L_1^\sharp). \tag{1}$$

The language L_2. Additional symbols α, β_1, β_2 are added to the alphabet of L_1. A word is in L_2, if it has the form

$$x_1\ \alpha y_1\ x_2\ \alpha y_2\ x_3\ \alpha y_3 \ldots\ \alpha y_{n-1}\ x_n\ \alpha \tag{2}$$

with $x_1, x_2, \ldots, x_n$ in L_1 and $y_i{=}\beta_1$ iff x_i is in $(L_1^\flat \cap L_2^\sharp)$, $y_i{=}\beta_2$ iff x_i is in $(L_2^\flat \cap L_1^\sharp)$.

3 Results

Theorem 1. *The language L_1 cannot be recognized by deterministic one-counter one-way automata, but it can be recognized with bounded error by a quantum one-counter one-way automaton.*

Proof. To prove the first claim, assume to the contrary that a deterministic one-counter one-way finite automaton recognizing L_1 does exist and has k states. Consider the words $x_{ij} = 0^i 3^j 1^i 4^{j-1} \flat \sharp 5$, where $j \le i \le n$. Clearly, all $x_{ij} \in L_1$. When the first part $0^i 3^j$ of the word x_{ij} has been read, the value of the counter is $i + j \le 2n$ at most, so the automaton can at this stage distinguish $2nk$ of the words x_{ij} at most that are $\frac{1}{2}n(n+1)$ in total. Thus, if n is large enough, two different words, $x_{i_1 j_1}$ and $x_{i_2 j_2}$, would, at this step of computation, share the same state and counter value. But then, clearly, the automaton would also accept the words $0^{i_1} 3^{j_1} 1^{i_2} 4^{j_2 - 1} \flat \sharp 5$ and $0^{i_2} 3^{j_2} 1^{i_1} 4^{j_1 - 1} \flat \sharp 5$ neither of which is in the language L_1.

We now describe a 1Q1CA which, as we subsequently show, recognizes L_1 with bounded error. In addition to an initial state, the automaton will have sixteen non-terminating states q_{ijk}, q'_{ij}, q''_{ij}, $i, j, k = 1, 2$, four accepting states

$a_1, \ldots, a_4$, and 8 rejecting states $r_1, \ldots, r_8$. As customary, we interpret invertible transformations of the set of basis states of the automaton as unitary operators in its quantum configuration space.

When the initial marker $\vdash$ comes in, the states q_{ij1} get amplitudes $(-1)^{i+j}\frac{1}{2}$, while all the remaining states get amplitude 0. When any of the symbols $0, 1, 3$, or 4 arrives, the state remains unchanged; the counter is changed only in the following cases: the symbols 0 and 3 *increase* the counter for the states q_{1jk} and q_{2jk}, respectively, while the symbols 1 and 4 *decrease* the counter for the states q_{1jk} and q_{2jk}, respectively.

The special symbol $\flat$ is ignored, if read in any of the states q_{2jk} or q''_{ij}; if read in the state q_{1jk} and the counter is 0, the state q'_{jk} follows, whereas the state q'_{jk^*} [1] follows if the counter is not 0; if read in the state q'_{ij}, the rejecting states $r_1, \ldots, r_4$ follow.

The special symbol $\sharp$ is ignored if read in any of the states q_{1jk} or q'_{ij}; if read in the state q_{2jk} and the counter is 0, the state q''_{jk} follows, while the state q''_{jk^*} follows if the counter is not 0; if read in the state q''_{ij}, the rejecting states $r_5, \ldots, r_8$ follow.

The symbol 2 is ignored, if read in any of the states q_{2jk}, q''_{ij}, or q'_{j1}; if read in state q_{1jk} the rejecting states $r_1, \ldots, r_4$ follow; if read in the state q'_{j2}, the state remains unchanged while the counter decreases.

The symbol 5 is ignored if read in any of the states q_{1jk}, q'_{ij}, or q''_{j1}; if read in the state q_{2jk} the rejecting states $r_1, \ldots, r_4$ follow; if read in the state q''_{j2}, the state remains unchanged while the counter decreases.

When the end marker $\dashv$ arrives, if the value of the counter is 0, a unitary transformation is applied as in the following transition table (here and in following tables first column denotes the initial states, first row denotes resulting states, elements of the table are amplitudes):

	a_1	a_2	r_1	r_2	a_3	a_4	r_3	r_4
q'_{11}	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0
q'_{12}	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$	0	0	0	0
q'_{21}	0	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
q'_{22}	0	0	0	0	$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$
q''_{11}	$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$	0	0	0	0
q''_{12}	$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0
q''_{21}	0	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$
q''_{22}	0	0	0	0	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$

To complete the description of the automaton we should extend the transitions given above as follows:

- Each non terminating state, in the case that there is no transformation given above for some letter and zero or non zero counter value separately, should be mapped to some rejecting state. This can be easily accomplished without violating unitarity by adding some more rejecting states.

[1] If e is an element of a two-element set (here {1,2}), we write e^* to denote the other of the two elements.

- The other transitions that are not described above should be specified arbitrary, to ensure the unitarity of the matrix describing transformation for each letter and counter value.

It remains to verify that the automaton described above indeed recognizes the language L_1 with bounded error. First we need to compute the non-zero amplitudes for the automaton's states when a word $x \in \Sigma^*$ has been processed, right before the end marker $\hookleftarrow$. We do this for each of the eight cases as $x \in L_i^\flat \cap L_j^\sharp$, $i, j = 1, 2, 3$. We look first at the cases $i, j = 1, 2$. The value of the counter is 0 in all these cases. Note that in the case when x is an empty word, it has the same distribution of amplitudes as in $x \in L_1^\flat \cap L_1^\sharp$, so we will consider these two cases together:

	q'_{11}	q'_{12}	q'_{21}	q'_{22}	q''_{11}	q''_{12}	q''_{21}	q''_{22}
$x \in L_1^\flat \cap L_1^\sharp$	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	$-\frac{1}{2}$	0	$\frac{1}{2}$	0
$x \in L_1^\flat \cap L_2^\sharp$	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	$-\frac{1}{2}$	0	$\frac{1}{2}$
$x \in L_2^\flat \cap L_1^\sharp$	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	$-\frac{1}{2}$	0	$\frac{1}{2}$	0
$x \in L_2^\flat \cap L_2^\sharp$	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	$-\frac{1}{2}$	0	$\frac{1}{2}$

The following table shows non-zero amplitudes after the end marker $\hookleftarrow$ has been processed:

	a_1	a_2	r_1	r_2	a_3	a_4	r_3	r_4
$x \in L_1^\flat \cap L_1^\sharp$	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	$-\frac{1}{2}$	$-\frac{1}{2}$
$x \in L_1^\flat \cap L_2^\sharp$	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	$-\frac{1}{2}$	$-\frac{1}{2}$	0
$x \in L_2^\flat \cap L_1^\sharp$	$-\frac{1}{2}$	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	$-\frac{1}{2}$	0
$x \in L_2^\flat \cap L_2^\sharp$	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	$-\frac{1}{2}$	$\frac{1}{2}$

Hence, after the measurement, the accepting probability for $x \in L_1 = (L_1^\flat \cap L_2^\sharp) \cup (L_2^\flat \cap L_1^\sharp)$ is equal to $\frac{3}{4}$, while for $x \in L_1^\flat \cap L_1^\sharp$ or $x \in L_2^\flat \cap L_2^\sharp$ it is equal to $\frac{1}{4}$; the corresponding rejecting probabilities are complementary.

It remains to check the cases when i or j in $L_i^\flat \cap L_j^\sharp$ is equal to three. The amplitudes for the states corresponding to non-terminating states and zero value of the counter just before the end marker $\hookleftarrow$ arrives are then as follows:

	q'_{11}	q'_{12}	q'_{21}	q'_{22}	q''_{11}	q''_{12}	q''_{21}	q''_{22}
$x \in L_1^\flat \cap L_3^\sharp$	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	0	0	0
$x \in L_2^\flat \cap L_3^\sharp$	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	0	0
$x \in L_3^\flat \cap L_1^\sharp$	0	0	0	0	$-\frac{1}{2}$	0	$\frac{1}{2}$	0
$x \in L_3^\flat \cap L_2^\sharp$	0	0	0	0	0	$-\frac{1}{2}$	0	$\frac{1}{2}$

There are also non zero amplitudes for states with values of the counter not equal to 0 with norm $\frac{1}{\sqrt{2}}$, they will cause a rejecting states to be observed.

In the remaining case $x \in L_3^\flat \cap L_3^\sharp$ whether the word is rejected before end marker or all states with zero value of the counter have zero amplitudes.

Hence, after the end marker has been processed, we have the following terminating amplitudes:

	a_1	a_2	r_1	r_2	a_3	a_4	r_3	r_4
$x \in L_1^\flat \cap L_3^\sharp$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$
$x \in L_2^\flat \cap L_3^\sharp$	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$
$x \in L_3^\flat \cap L_1^\sharp$	$-\frac{1}{4}$	$\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$
$x \in L_3^\flat \cap L_2^\sharp$	$\frac{1}{4}$	$\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$	$-\frac{1}{4}$	$-\frac{1}{4}$	$\frac{1}{4}$

and in the case $x \in L_3^\flat \cap L_3^\sharp$ all non-terminating amplitudes are still 0. The probability of acceptance of x in any of these four cases is equal to $\frac{1}{4}$, while in the case $x \in L_3^\flat \cap L_3^\sharp$ it is zero; the rejecting probabilities are complementary.

To sum it up, the automaton accepts all words in the language L_1 with probability $\frac{3}{4}$, and rejects all words not in L_1 with probability at least $\frac{3}{4}$.

Theorem 2. *The language L_2 cannot be recognized with bounded error by probabilistic one-counter one-way finite automata, but it can be recognized with probability $\frac{3}{4}$ by a quantum one-counter one-way finite automaton.*

Proof. For the first statement, we first note that the language L_1 cannot be recognized with probability 1 by a probabilistic one-way one-counter finite automaton. Indeed, assuming the contrary and simulating the probabilistic automaton by a deterministic one which take the first of available choices of probabilistic automaton with positive probability at any time (our automaton reads one input symbol at a time and moves to the next symbol, so we can build such simulation), would bring us into contradiction with the first part of Theorem 1. The impossibility of recognizing L_2 by a probabilistic automaton with a bounded error now follows, since the subwords $x_i \in L_1$ of a word x in L_2 can be taken in arbitrarily large numbers, and every x_i is processed with a positive error.

For the second part of the theorem, we extend the construction of the quantum automaton described in the proof of Theorem 1. We should add transformations for the symbols $\alpha, \beta_1, \beta_2,$. We need four other non-terminating states q_i for it.

The transformation for the α is described as follows:

	q_1	q_2	r_1	r_2	r_3	q_3	q_4	r_4
q'_{11}	$\frac{1}{\sqrt{2}}$	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0
q'_{12}	0	$-\frac{1}{\sqrt{2}}$	$\frac{1}{2}$	$-\frac{1}{2}$	0	0	0	0
q'_{21}	0	0	0	0	$\frac{1}{2}$	0	$\frac{1}{\sqrt{2}}$	$\frac{1}{2}$
q'_{22}	0	0	0	0	$\frac{1}{2}$	$-\frac{1}{\sqrt{2}}$	0	$-\frac{1}{2}$
q''_{11}	0	$\frac{1}{\sqrt{2}}$	$\frac{1}{2}$	$-\frac{1}{2}$	0	0	0	0
q''_{12}	$-\frac{1}{\sqrt{2}}$	0	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0
q''_{21}	0	0	0	0	$\frac{1}{2}$	$\frac{1}{\sqrt{2}}$	0	$-\frac{1}{2}$
q''_{22}	0	0	0	0	$\frac{1}{2}$	0	$-\frac{1}{\sqrt{2}}$	$\frac{1}{2}$

The transformation for states $q1, q_2, q3, q_4$ and zero value of the counter for both letters β_1 and β_2, can be written as follows (the first row shows resulting states for β_1 and the second for β_2):

(β_1)	q_{111}	q_{121}	q_{211}	q_{221}	r_1	r_2	r_3	r_4
(β_2)	r_1	r_2	r_3	r_4	q_{111}	q_{121}	q_{211}	q_{221}
q_1	$\frac{1}{\sqrt{2}}$	0	0	$\frac{1}{\sqrt{2}}$	0	0	0	0
q_2	0	0	0	0	0	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	0
q_3	0	0	0	0	$\frac{1}{\sqrt{2}}$	0	0	$\frac{1}{\sqrt{2}}$
q_4	0	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	0	0	0	0	0

Finally we describe the transformation for the end marker $\hookleftarrow$ for states $q1$, q_2, q_3, q_4 and zero value of the counter as follows:

	a_1	a_2	a_3	r_1
q_1	1	0	0	0
q_2	0	1	0	0
q_3	0	0	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$
q_4	0	0	$\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$

To complete the description of the automaton we should extend the described transformations for each letter and zero and non zero value of the counter as in the automaton of Theorem 1.

It remains to verify that the automaton described above recognizes the language L_2 with probability $\frac{3}{4}$.

While processing x_1 the automaton acts as described for L_1, so when first α comes the distribution of amplitudes is exactly the same as described in proof of L_1 before $\hookleftarrow$. After applying transformation for α the automaton gets the amplitudes for states with zero value of the counter that are shown in the second column of the following table:

	q_1	q_2	r_1	r_2	r_3	q_3	q_4	r_4	p_r	q_1	q_2	q_3	q_4
$x_1 \in L_1^\flat \cap L_1^\sharp$	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	0	$\frac{1}{2}$	0	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$
$x_1 \in L_1^\flat \cap L_2^\sharp$	$\frac{1}{\sqrt{2}}$	0	0	0	0	0	$-\frac{1}{\sqrt{2}}$	0	0	$\frac{1}{\sqrt{2}}$	0	0	$-\frac{1}{\sqrt{2}}$
$x_1 \in L_2^\flat \cap L_1^\sharp$	0	$-\frac{1}{\sqrt{2}}$	0	0	0	$\frac{1}{\sqrt{2}}$	0	0	0	0	$-\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	0
$x_1 \in L_2^\flat \cap L_2^\sharp$	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	0	$-\frac{1}{2}$	0	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$

The third column shows the total probability of rejection after processing $x_1\alpha$. So we see that only in the cases $x_1 \in L_1^\flat \cap L_2^\sharp$ or $x_1 \in L_2^\flat \cap L_1^\sharp$ the probability of rejection is 0. When $x_1 \in L_1^\flat \cap L_1^\sharp$ or $x_1 \in L_2^\flat \cap L_2^\sharp$ the probability of rejection is $\frac{1}{2}$.

Similar computation for cases when i or j in $L_i^\flat \cap L_j^\sharp$ is equal to 3 give the probability of rejection at least $\frac{3}{4}$, so these cases are not considered further.

Note that during the measurement that follows α if automaton do not terminates according to definition of the measurement non zero amplitudes remain

only for non terminating states and vector is normalized. So the amplitudes after the measurement for this case are shown in the last column.

If β_1 or $\beta 2$ come after α, the amplitudes become as follows:

(β_1)	q_{111}	q_{121}	q_{211}	q_{221}	r_1	r_2	r_3	r_4
(β_2)	r_1	r_2	r_3	r_4	q_{111}	q_{121}	q_{211}	q_{221}
$x_1 \in L_1^\flat \cap L_1^\sharp$	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$\frac{1}{2\sqrt{2}}$	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$\frac{1}{2\sqrt{2}}$
$x_1 \in L_1^\flat \cap L_2^\sharp$	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0
$x_1 \in L_2^\flat \cap L_1^\sharp$	0	0	0	0	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$
$x_1 \in L_2^\flat \cap L_2^\sharp$	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$\frac{1}{2\sqrt{2}}$	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	$\frac{1}{2\sqrt{2}}$

So if x_1 is in $(L_1^\flat \cap L_2^\sharp)$ and β_1 is read or x_1 is in $(L_2^\flat \cap L_1^\sharp)$ and β_2 is read, we get the same amplitude distribution as after initial marker $\not\vdash$.

If x_1 is in $(L_1^\flat \cap L_2^\sharp)$ and β_2 is read or x_1 is in $(L_2^\flat \cap L_1^\sharp)$ and β_1 is read, the word is rejected with probability 1.

If x_1 is in $(L_1^\flat \cap L_1^\sharp)$ or x_1 is in $(L_2^\flat \cap L_2^\sharp)$ then $\frac{1}{2}$ of the remaining amplitudes is in rejecting states. So thus after α the probability of rejection for such a word is already $\frac{1}{2}$, the total probability of rejection becomes $\frac{3}{4}$

We should also check cases when β_1 or β_2 occur in other position than after α. Due to the construction of automaton the word is rejected immediately.

So we have seen that after processing $x_1 \alpha y_1$ the automaton is in the same quantum state as after reading initial marker $\not\vdash$ in the cases x_1 is in $(L_1^\flat \cap L_2^\sharp)$ and $y_1{=}\beta_1$ or x_1 is in $(L_2^\flat \cap L_1^\sharp)$ and $y_1{=}\ \beta_2$, otherwise the word is rejected with probability at least $\frac{3}{4}$. So the computation on $x_i \alpha y_i$ will be the same as for $x_1 \alpha y_1$ if the previous part of the word corresponds to the conditions of the language, otherwise the word will be rejected with probability at least $\frac{3}{4}$.

Finally, we need to show the processing of the end marker. Note that it should come after α, otherwise the word is rejected. Assume that end marker comes after x_1.

The resulting amplitudes are following:

	a_1	a_2	a_3	r_1
$x_1 \in L_1^\flat \cap L_1^\sharp$	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	0	$-\frac{1}{2}$
$x_1 \in L_1^\flat \cap L_2^\sharp$	$\frac{1}{\sqrt{2}}$	0	$-\frac{1}{2}$	$\frac{1}{2}$
$x_1 \in L_2^\flat \cap L_1^\sharp$	0	$-\frac{1}{\sqrt{2}}$	$\frac{1}{2}$	$\frac{1}{2}$
$x_1 \in L_1^\flat \cap L_1^\sharp$	$\frac{1}{2\sqrt{2}}$	$-\frac{1}{2\sqrt{2}}$	0	$-\frac{1}{2}$

So the word is accepted with probability $\frac{3}{4}$ if the $x_1 \in L_1^\flat \cap L_2^\sharp$ or $x_1 \in L_2^\flat \cap L_1^\sharp$ cases. The word is rejected with probability $\frac{1}{2}$ already after processing α in the cases $x_1 \in L_1^\flat \cap L_1^\sharp$ or $x_1 \in L_2^\flat \cap L_2^\sharp$ and thus the total probability of rejection in these cases is $\frac{3}{4}$.

References

1. Ambainis, A., and R. Freivalds: 1-way quantum finite automata: strengths, weaknesses and generalizations. Proc. 39^{th} FOCS (1998) 332 – 341
2. Ambainis, A., and J. Watrous: Two-way finite automata with quantum and classical states. http://xxx.lanl.gov/abs/cs.CC/9911009
3. Deutsch, D.: Quantum theory, the Church-Turing principle and the universal quantum computer. Proc. Royal Society London, A400 (1989) 96–117
4. Gruska, J.: Quantum Computing , McGraw Hill (1999)
5. Kondacs, A., and J. Watrous: On the power of quantum finite state automata. Proc. 38^{th} FOCS (1997) 66–75
6. Kravtsev, M.: Quantum Finite One-Counter Automata. In: Proc. 26^{th} SOFSEM (1999), LNCS 1725, Springer-Verlag, 431–440. http://xxx.lanl.gov/abs/quant-ph/9905092

How to Employ Reverse Search in Distributed Single Source Shortest Paths*

Luboš Brim, Ivana Černá, Pavel Krčál, and Radek Pelánek

Department of Computer Science, Faculty of Informatics
Masaryk University Brno, Czech Republic
{brim,cerna,xkrcal,xpelanek}@fi.muni.cz

Abstract. A distributed algorithm for the single source shortest path problem for directed graphs with arbitrary edge lengths is proposed. The new algorithm is based on relaxations and uses reverse search for inspecting edges and thus avoids using any additional data structures. At the same time the algorithm uses a novel way to recognize a reachable negative-length cycle in the graph which facilitates the scalability of the algorithm.

1 Introduction

The single source shortest paths problem is a key component of many applications and lots of effective sequential algorithms are proposed for its solution (for an excellent survey see [3]). However, in many applications graphs are too massive to fit completely inside the computer's internal memory. The resulting input/output communication between fast internal memory and slower external memory (such as disks) can be a major performance bottleneck.

In particular, in LTL model checking application (see Section 6) the graph is typically extremely large. In order to optimize the space complexity of the computation the graph is generated *on-the-fly*. Successors of a vertex are determined dynamically and consequently there is no need to store any information about edges permanently. Therefore neither the techniques used in external memory algorithms (we do not know any properties of the examined graph in advance) nor the parallel algorithms based on adjacency matrix graph representation are applicable.

The approach we have been looking upon is to increase the computational power (especially the amount of randomly accessed memory) by building a powerful parallel computer as a network of cheap workstations with disjoint memory which communicate via message passing.

With respect to the intended application even in the distributed environment the space requirements are the main limiting factors. Therefore we have been looking for a distributed algorithm compatible with other space-saving techniques (e.g. *on-the-fly* technique or *partial order* technique). Our distributed

* This work has been partially supported by the Grant Agency of Czech Republic grant No. 201/00/1023.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 191–200, 2001.

algorithm is therefore based on the relaxation of graph's edges [5]. Distributed relaxation-based algorithms are known only for special settings of single source shortest paths problem. For general digraphs with non-negative edge lengths parallel algorithms are presented in [6,7,9]. For special cases of graphs, like planar digraphs [10], graphs with separator decomposition [4] or graphs with small tree-width [2], more efficient algorithms are known. Yet none of these algorithms is applicable to general digraphs with potential negative length cycle.

The most notable features of our proposed distributed algorithm are *reverse search* and *walk to root* approaches. The *reverse search* method is known to be an exceedingly space efficient technique [1,8]. Data structures of the proposed algorithm can be naturally used by the *reverse search* and it is possible to reduce the memory requirements which would be otherwise induced by structures used for traversing graph (such as a queue or a stack). This could save up to one third of memory which is practically significant.

Walk to root is a strategy how to detect the presence of a negative length cycle in the input graph. The cycle is looked for in the graph of parent pointers maintained by the method. The parent graph cycles, however, can appear and disappear. The aim is to detect a cycle as soon as possible and at the same time not to increase the time complexity of underlying relaxation algorithm significantly. To that end we introduce a solution which allows to amortize the time complexity of cycle detection over the complexity of relaxation.

2 Problem Definition and General Method

Let (G, s, l) be a given triple, where $G = (V, E)$ is a directed graph, $l : E \to R$ is a *length function* mapping edges to real-valued lengths and $s \in V$ is the source vertex. We denote $n =\mid V \mid$ and $m =\mid E \mid$. The *length* $l(p)$ *of the path* p is the sum of the lengths of its constituent edges. We define the *shortest path length* from s to v by

$$\delta(s, v) = \begin{cases} \min\{l(p) \mid \ p \text{ is a path from } s \text{ to } v\} & \text{if there is such a path} \\ \infty & \text{otherwise} \end{cases}$$

A *shortest path* from vertex s to vertex v is then defined as any path p with length $l(p) = \delta(s, v)$. If the graph G contains no negative length cycles (*negative cycles*) reachable from source vertex s, then for all $v \in V$ the shortest path length remains well-defined and the graph is called *feasible*. The single source shortest paths (SSSP) problem is to determine whether the given graph is *feasible* and if so to compute $\delta(s, v)$ for all $v \in V$. For purposes of our algorithm we suppose that some linear ordering on vertices is given.

The general method for solving the SSSP problem is the *relaxation* method [3,5]. For every vertex v the method maintains its distance label $d(v)$ and parent vertex $p(v)$. The subgraph G_p of G induced by edges $(p(v), v)$ for all v such that $p(v) \neq nil$ is called the *parent graph*. The method starts by setting $d(s) = 0$ and $p(s) = nil$. At every step the method selects an edge (v, u) and *relaxes* it which

means that if $d(u) > d(v) + l(v,u)$ then it sets $d(u)$ to $d(v) + l(v,u)$ and sets $p(u)$ to v.

If no $d(v)$ can be improved by any relaxation then $d(v) = \delta(s,v)$ for all $v \in V$ and G_p determines the shortest paths. Different strategies for selecting an edge to be relaxed lead to different algorithms. For graphs where negative cycles could exist the relaxation method must be modified to recognize the *unfeasibility* of the graph. As in the case of relaxation various strategies are used to detect negative cycles [3]. However, not all of them are suitable for our purposes – they are either uncompetitive (as for example time-out strategy) or they are not suitable for distribution (such as the admissible graph search which uses hardly parallelizable DFS or level-based strategy which employs global data structures). For our version of distributed SSSP we have used the *walk to root* strategy.

The sequential *walk to root* strategy can be described as follows. Suppose the relaxation operation applies to an edge (v,u) (i.e. $d(u) > d(v) + l(v,u)$) and the parent graph G_p is acyclic. This operation creates a cycle in G_p if and only if u is an ancestor of v in the current parent graph. This can be detected by following the parent pointers from v to s. If the vertex u lies on this path then there is a negative cycle; otherwise the relaxation operation does not create a cycle.

The *walk to root* method gives immediate cycle detection and can be easily combined with the relaxation method. However, since the path to the root can be long, it increases the cost of applying the relaxation operation to an edge to $\mathcal{O}(n)$. We can use amortization to pay the cost of checking G_p for cycles. Since the cost of such a search is $\mathcal{O}(n)$, the search is performed only after the underlying shortest paths algorithm performs $\Omega(n)$ work. The running time is thus increased only by a constant factor. However, to preserve the correctness the behavior of *walk to root* has to be significantly modified. The amortization is used in the distributed algorithm and is described in detail in Section 5.

3 Reverse Search

Reverse search is originally a technique for generating large sets of discrete objects [1,8]. Reverse search can be viewed as a depth-first graph traversal that requires neither stack nor node marks to be stored explicitly – all necessary information can be recomputed. Such recomputations are naturally time-consuming, but when traversing extremely large graphs, the actual problem is not the time but the memory requirements.

In its basic form the reverse search can be viewed as the traversal of a spanning tree, called the *reverse search tree*. We are given a *local search function* f and an *optimum vertex* v^*. For every vertex v, repeated application of f has to generate a path from v to v^*. The set of these paths defines the *reverse search tree* with the root v^*. A reverse search is initiated at v^* and only edges of the reverse search tree are traversed.

In the context of the SSSP problem we want to traverse the graph G. The parent graph G_p corresponds to the reverse search tree. The optimum vertex v^* corresponds to the source vertex s and the local search function f to the

parent function p. The correspondence is not exact since $p(v)$ can change during the computation whereas original search function is fixed. Consequently some vertices can be visited more than once. This is in fact the desired behavior for our application. Moreover, if there is a negative cycle in the graph G then a cycle in G_p will occur and G_p will not be a spanning tree. In such a situation we are not interested in the shortest distances and the way in which the graph is traversed is not important anymore. We just need to detect such a situation and this is delegated to the cycle detection strategy.

```
proc Reverse_search (s)
  p(s) := ⊥;
  v := s;
  while v ≠ ⊥ do
        Do_something (v);
        u := Get_successor (v, NULL);
        while u does not exist do
              last := v; v := p(v);
              u := Get_successor (v, last);
        od
        v := u;
  od
end
```

```
proc Call_recursively (v)
  Do_something (v);
  for each edge (v, w) ∈ E do
      if p(w) = v then
         Call_recursively (w)
      fi
  od
end
```

Fig. 1. Demonstration of the reverse search

Fig. 1 demonstrates the use of the reverse search within our algorithm. Both procedures *Call_recursively(v)* and *Reverse_search(v)* traverse the subtree of v in the same manner and perform some operation on its children. But *Call_recursively* uses a stack whereas *Reverse_search* uses the parent edges for the traversal. The function *Get_successor(v, w)* returns the first successor u of v which is greater than w with respect to the ordering on the vertices and $p(u) = v$. If no such successor exists an appropriate announcement is returned.

4 Sequential SSSP Algorithm with Reverse Search

We present the sequential algorithm (Fig. 2) and prove its correctness and complexity first. This algorithm forms the base of the distributed algorithm presented in the subsequent section.

The *Trace* procedure visits vertices in the graph (we say that a vertex is *visited* if it is the value of the variable v). The procedure terminates either when a negative cycle is detected or when the traversal of the graph is completed.

The *RGS* function combines the relaxation of an edge as introduced in Section 2 and the *Get_successor* function from Section 3. It finds the next vertex u whose label can be improved. The change of $p(u)$ can create a cycle in G_p and therefore the *WTR* procedure is started to detect this possibility. If the change is safe the values $d(u)$ and $p(u)$ are updated and u is returned.

In what follows the correctness of the algorithm is stated. Due to the space limits the proofs are only sketched.

```
1  proc Trace (s)
2    p(s) := ⊥; v := s;
3    while v ≠ ⊥ do
4          u := RGS (v, NULL);
5          while u does not exist do
6                last := v; v := p(v);
7                u := RGS (v, last); od
8          v := u; od
9  end

1  proc RGS (v, last){Relax and Get Successor}
2    u := successor of v greater than last;
3    while d(u) ≤ d(v) + l(u, v) do
4          u := next successor of v; od
5    if u exists then
6                WTR (v, u);
7                d(u) := d(v) + l(u, v); p(u) := v;
8                return u;
9          else return u does not exist; fi
10 end

1  proc WTR (at, looking_for){Walk To Root}
2    while at ≠ s and at ≠ looking_for do at := p(at); od
3    if at = looking_for then negative cycle detected fi
4  end
```

Fig. 2. Pseudo-code of the sequential algorithm

Lemma 1. *Let G contains no negative cycle reachable from the source vertex s. Then G_p forms a rooted tree with root s and $d(v) \geq \delta(s, v)$ for all $v \in V$ at any time during the computation. Moreover, once $d(v) = \delta(s, v)$ it never changes.*

Proof: The proof is principally the same as for other relaxation methods [5].

Lemma 2. *After every change of the value $d(v)$ the algorithm visits the vertex v.*

Proof: Follows directly from the algorithm.

Lemma 3. *Let G contains no negative cycle reachable from the source vertex s. Every time a vertex w is visited the sequence* S *of the assignments on line 6 of the procedure* Trace *will eventually be executed for this vertex. Until this happens $p(w)$ is not changed.*

Proof: The value $p(w)$ cannot be changed because G has no negative cycle and due to Lemma 1 the parent graph G_p does not have any cycle. Let $h(w)$ denotes the depth of w in G_p. We prove the lemma by backward induction (from n to 0) with respect to $h(w)$. For the basis we have $h(w) = n$, w has no child and therefore *RGS(w,NULL)* returns *u does not exist* and the sequence S is executed immediately. For the inductive step we assume that the lemma holds for each v such that $h(v) \geq k$ and let $h(w) = k - 1$, $\{a_1, a_2, \ldots, a_r\} = \{u \mid (w, u) \in E\}$. Since $h(a_i) = k$ for all $i \in \{1, \ldots, r\}$, we can use the induction hypothesis for each a_i and show that the value of the variable u in *RGS* is equal to a_i exactly once. Therefore *RGS* returns *u does not exist* for w after a finite number of steps and the sequence S is executed. ■

Theorem 1 (Correctness of the sequential algorithm). *If G has no negative cycle reachable from the source s then the sequential algorithm terminates with $d(v) = \delta(s, v)$ for all $v \in V$ and G_p forms a shortest-paths tree rooted at s. If G has a negative cycle, its existence is reported.*

Proof: Let us at first suppose that there is no negative cycle. Lemma 3 applied to the source vertex s gives the termination of the algorithm. Let $v \in V$ and $< v_0, v_1, \ldots, v_k >, s = v_0, v = v_k$ is a shortest path from s to v. We show that $d(v_i) = \delta(s, v_i)$ for all $i \in \{0, \ldots k\}$ by induction on i and therefore $d(v) = \delta(s, v)$. For the basis $d(v_0) = d(s) = \delta(s, s) = 0$ by Lemma 1. From the induction hypothesis we have $d(v_i) = \delta(s, v_i)$. The value $d(v_i)$ was set to $\delta(s, v_i)$ at some moment during the computation. From Lemma 2 vertex v_i is visited afterwards and the edge (v_i, v_{i+1}) is relaxed. Due to Lemma 1, $d(v_{i+1}) \geq \delta(s, v_{i+1}) = \delta(s, v_i) + l(v_i, v_{i+1}) = d(v_i) + l(v_i, v_{i+1})$ is true before the relaxation and therefore $d(v_{i+1}) = d(v_i) + l(v_i, v_{i+1}) = \delta(s, v_i) + l(v_i, v_{i+1}) = \delta(s, v_{i+1})$ holds after the relaxation. By Lemma 1 this equality is maintained afterwards.

For all vertices v, u with $v = p(u)$ we have $d(u) = d(v) + l(v, u)$. This follows directly from line 7 of the *RGS* procedure. After the termination $d(v) = \delta(s, v)$ and therefore G_p forms a shortest paths tree.

On the other side, if there is a negative cycle in G, then the relaxation process alone would run forever and would create a cycle in G_p. The cycle is detected because before any change of $p(v)$ *WTR* tests whether this change does not create a cycle in G_p. ■

Let us suppose that edges have integer lengths and let $C = \max\{|\ l(u, v)\ | : (u, v) \in E\}$.

Theorem 2. *The worst time complexity of the sequential algorithm is $\mathcal{O}(Cn^4)$.*

Proof: Each shortest path consists of at most $n - 1$ edges and $-C(n - 1) \leq \delta(s, v) \leq C(n - 1)$ holds for all $v \in V$. Each vertex v is visited only after $d(v)$ is lowered. Therefore each vertex is visited at most $\mathcal{O}(Cn)$ times. Each visit consists of updating at most n successors and an update can take $\mathcal{O}(n)$ time (due to the *walk to root*). Together we have $\mathcal{O}(Cn^3)$ bound for total visiting time of each vertex and $\mathcal{O}(Cn^4)$ bound for the algorithm. ■

We stress that the use of the *walk to root* in this algorithm is not unavoidable and the algorithm can be easily modified to detect a cycle without the *walk to root* and run in $\mathcal{O}(Cn^3)$ time. The *walk to root* has been used to make the presentation of the distributed algorithm (where the walk to root is essential) clearer.

5 Distributed Algorithm

For the distributed algorithm we suppose that the set of vertices is divided into disjoint subsets. The distribution is determined by the function *owner* which assigns every vertex v to a processor i. Processor i is responsible for the subgraf determined by the owned subset of vertices. Good partition of vertices among

processors is important because it has direct impact on communication complexity and thus on run-time of the program. We do not discuss it here because it is itself quite a difficult problem and depends on the concrete application.

The main idea of the distributed algorithm (Fig. 3) can be summarized as follows. The computation is initialized by the processor which owns the source vertex by calling $Trace(s, \perp)$ and is expanded to other processors as soon as the traversal visits the "border" vertices. Each processor visits vertices basically in the same manner as the sequential algorithm does.

While relaxation can be performed in parallel, the realization of *walk to root* requires more careful treatment. Even if adding the edge initiating the *walk to root* does not create a cycle in the parent graph, the parent graph can contain a cycle on the way to root created in the meantime by some other processor. The *walk to root* we used in the sequential algorithm would stay in this cycle forever. Amortization of walk brings similar problems. We propose a modification of the *walk to root* which solves both problems.

Each processor maintains a *counter* of started *WTR* procedures. The *WTR* procedure marks each node through which it proceeds by the name of the vertex where the walk has been started (*origin*) and by the current value of the processor *counter* (*stamp*). When the walk reaches a vertex that is already marked with the same *origin* and *stamp* a negative cycle is detected and the computation is terminated. In distributed environment it is possible to start more than one walk concurrently and it may happen that the walk reaches a vertex that is already marked by some other mark. In that case we use the ordering on vertices to decide whether to finish the walk or to overwrite the previous mark and continue. In the case that the walk has been finished (i.e. it has reached the root or a vertex marked by higher *origin*, line 9 of *WTR*) we need to remove its marks. This is done by the *REM* (REmove Marks) procedure which follows the path in the parent graph starting from the *origin* in the same manner as *WTR* does. The values $p(v)$ of marked vertices are not changed (line 6 of *RGS*) and therefore the *REM* procedure can find and remove the marks. However, due to possible overwriting of walks, it is possible that the *REM* procedure does not remove all marks. Note that these marks will be removed by some other *REM* procedure eventually. The correctness of cycle detection is guaranteed as for the cycle detection the equality of both the *origin* and *stamp* is required.

The modifications of *walk to root* enforces the *Trace* procedure to stop when it reaches a marked vertex and to wait till the vertex becomes unmarked. Moreover, *walk to root* is not called during each relaxation step (*WTR_amortization* condition becomes true every n-th time it is called).

Whenever a processor has to process a vertex (during traversing or *walk to root*) it checks whether the vertex belongs to its own subgraph. If the vertex is local, the processor continues locally otherwise a message is sent to the owner of the vertex. The algorithm periodically checks incoming messages (line 4 of *Trace*). When a request to update parameters of a vertex u arrives, the processor compares the current value $d(u)$ with the received one. If the received value is lower than the current one then the request is placed into the local *queue*.

```
1  proc Main
2    while  not finished  do
3           req := pop(queue);
4           if req.length = d(req.vertex) then Trace (req.vertex, req.father);  fi
5    od
6  end
```

```
1  proc Trace (v, father)
2    p(v) := father;
3    while  v ≠ father  do
4           Handle_messages;
5           u := RGS(v, NULL);
6           while u does not exist  do
7                  last := v;  v := p(v);
8                  u := RGS(v, last);  od
9           v := u;
10   od
11 end
```

```
1  proc RGS (v, last) {Relax and Get Successor}
2    u :=  successor of v greater than last;
3    while u exists do
4           if u is local then
5                                 if d(u)  >  d(v) + l(u, v)  then
6                                   if mark(u) then  wait;  fi
7                                   p(u) := v;
8                                   d(u) := d(v) + l(u, v);
9                                   if WTR_amortization then WTR([u, stamp], u);  inc(stamp);  fi
10                                  return u;
12                                fi
13                          else send_message(owner(u), "update u, v, d(u) + l(u, v)");
14          fi
15          u :=  next successor of v;
16   od
17   return u does not exist;
18 end
```

```
1  proc WTR ([origin, stamp], at) {Walk To Root}
2    done := false;
3    while ¬done do
4           if at is local
5             then
6                   if mark(at) = [origin, stamp] →
7                     send_message(Manager, "negative cycle found");
8                     terminate
9                   [] (at = source) ∨ (mark(at) > [origin, stamp]) →
10                    if origin is local
11                      then REM([origin, stamp], origin)
12                       else send_message(owner(origin),
13                                    "start REM([origin, stamp], origin))" fi
14                    done := true;
15                  [] (mark(at) = nil) ∨ (mark(at) < [origin, stamp]) →
16                    mark(at) := [origin, stamp];
17                    at := p(at)
18                  fi
19              else send_message(owner(at), "start WTR([origin, stamp], at)");
20                   done := true
21          fi
22   od
23 end
```

Fig. 3. Pseudo-code of the distributed algorithm

Anytime the traversal ends the next request from the *queue* is popped and a new traversal is started.

Another type of message is a request to continue in the *walk to root* (resp. in removing marks), which is immediately satisfied by executing the *WTR* (resp. *REM*) procedure.

The distributed algorithm terminates when all local *queues* of all processors are empty and there are no pending messages or when a negative cycle is detected. A *manager* process is used to detect the termination and to finish the algorithm by sending a *termination* signal to all processors.

Theorem 3 (Correctness and complexity of the distributed algorithm). *If G has no negative cycle reachable from the source s then the distributed algorithm terminates with $d(v) = \delta(s, v)$ for all $v \in V$ and G_p forms a shortest-paths tree rooted at s. If G has a negative cycle, its existence is reported.*

The worst time complexity of the algorithm is $\mathcal{O}(Cn^3)$.

Proof: The proof of the correctness of the distributed algorithm is technically more involved and due to the space limits is presented in the full version of the paper only. The basic ideas are the same as for the sequential case, especially in the case when G has no negative cycle. Proof of the correctness of the distributed *walk to root* strategy is based on the ordering on walks and on the fact that if G contains a reachable negative cycle then after a finite number of relaxation steps G_p always has a cycle.

Complexity is $\mathcal{O}(Cn^3)$ due to the amortization of the *walk to root*. ■

6 Experiments

We have implemented the distributed algorithm. The experiments have been performed on a cluster of seven workstations interconnected with a fast 100Mbps Ethernet using Message Passing Interface (MPI) library.

We have performed a series of practical experiments on particular types of graphs that represent the *LTL model checking problem*. The LTL model checking problem is defined as follows. Given a finite system and a LTL formula decide whether the given system satisfies the formula. This problem can be reduced to the problem of finding an *accepting cycles* in a directed graph [11] and has a linear sequential complexity. In practice however, the resulting graph is usually very large and the linear algorithm is based on depth-first search, which makes it hard to distribute. We have reduced the model checking problem to the SSSP problem with edge lengths 0, -1. Instead of looking for accepting cycles we detect negative cycles.

The experimental results clearly confirm that for LTL model checking our algorithm is able to verify systems that were beyond the scope of the sequential model checking algorithm.

Part of our experimental results is summarized in the table below. The table shows how the number of computers influences the computation time. Time is given in minutes, 'M' means that the computation failed due to low memory.

	Number of Computers						
No. of Vertices	1	2	3	4	5	6	7
94578	0:38	0:35	0:26	0:21	0:18	0:17	0:15
608185	5:13	4:19	3:04	2:26	2:03	1:49	1:35
777488	M	6:50	4:09	3:12	2:45	2:37	2:05
736400	M	M	M	6:19	4:52	4:39	4:25

7 Conclusions

We have proposed a distributed algorithm for the single source shortest paths problem for arbitrary directed graphs which can contain negative length cycles. The algorithm employs reverse search and uses one data structure for two purposes — computing the shortest paths and traversing the graph. A novel distributed variant of the walk to root negative cycle detection strategy is engaged. The algorithm is thus space-efficient and scalable.

Because of the wide variety of relaxation and cycle detection strategies there is plenty of space for future research. Although not all strategies are suitable for distributed solution, there are surely other possibilities besides the one proposed in this paper.

References

1. D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Appl. Math.*, 65:21–46, 1996.
2. S. Chaudhuri and C. D. Zaroliagis. Shortest path queries in digraphs of small treewidth. In *Automata, Languages and Programming*, pages 244–255, 1995.
3. B. V. Cherkassky and A. V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming, Springer-Verlag*, 85:277–311, 1999.
4. E. Cohen. Efficient parallel shortest-paths in digraphs with a separator decomposition. *Journal of Algorithms*, 21(2):331–357, 1996.
5. T. H. Cormen, Ch. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT, 1990.
6. A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. A parallelization of Dijkstra's shortest path algorithm. In *Proc. 23rd MFCS'98, Lecture Notes in Computer Science*, volume 1450, pages 722–731. Springer-Verlag, 1998.
7. U. Meyer and P. Sanders. Parallel shortest path for arbitrary graphs. In *6th International EURO-PAR Conference*. LNCS, 2000.
8. J. Nievergelt. Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power. In *SOFSEM 2000*, number 1963 in LNCS, pages 18–35. Springer, 2000.
9. K. Ramarao and S. Venkatesan. On finding and updating shortest paths distributively. *Journal of Algorithms*, 13:235–257, 1992.
10. J. Traff and C.D. Zaroliagis. A simple parallel algorithm for the single-source shortest path problem on planar digraphs. In *Parallel algorithms for irregularly structured problems*, volume 1117 of *LNCS*, pages 183–194. Springer, 1996.
11. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *1st Symp. on Logic in Computer Science, LICS'86*, pages 332–344. Computer Society Press, 1986.

Multi-agent Systems as Concurrent Constraint Processes*

Luboš Brim[1], David Gilbert[1], Jean-Marie Jacquet[2], and Mojmír Křetínský[3]

[1] Dept.of Comp.Sci., City University, London, U.K.
drg@soi.city.ac.uk
[2] Dept.of Comp.Sci., University of Namur, Namur, Belgium
jmj@info.fundp.ac.be
[3] Faculty of Informatics, Masaryk University Brno, Brno, Czech Republic
{brim,mojmir}@fi.muni.cz

Abstract. We present a language `Scc` for a specification of the direct exchange and/or the global sharing of information in multi-agent systems. `Scc` is based on concurrent constraint programming paradigm which we modify in such a way that agents can (i) maintain its local private store, (ii) share (read/write) the information in the global store and (iii) communicate with other agents (via multi-party or hand-shake). To justify our proposal we compare `Scc` to a recently proposed language for the exchange of information in multi-agent systems. Also we provide an operational semantics of `Scc`. The full semantic treatment is sketched only and done elsewhere.

1 Introduction

Multi-agent system is a system composed of several autonomous agents that operate in a distributed environment which they can perceive, reason about as well as can affect by performing actions. In the current research of multi-agent systems, a major topic is the development of a standardised agent communication language for the exchange of information. Several languages have been proposed, e.g. [7,9,12,4]. Recently de Boer et al.([4]) have also (for the first time) introduced a formal semantic theory for the exchange of information in the multi-agent systems. Their approach uses principles of concurrent constraint programming (CCP) to model the local behaviour of agents while the communication is modelled by a standard process algebraic hand-shake approach.

Our proposal is based on CCP paradigm only, however semantics of mechanism for updating and testing the (local/global) store(s) are changed.

CCP has inherited two of main features of concurrent logic programming, namely the asynchronous character of the communication and the monotonic update of the store. During last years some work have been done to lift both features. On the one hand, de Boer et al. [5] has proposed non-monotonic updates

* This work has been partially supported by the Grant Agency of Czech Republic grant No. 201/00/0400.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 201–210, 2001.

of the store and have studied compositional and fully abstract semantics for them. On the other hand, Saraswat proposed a synchronisation mechanism in ([10]). Briefly, his proposal and many related ones (e.g. [6]) are based on a coding of an explicit operator to achieve synchrony. This should be contrasted with the classical concurrent constraint framework in which asynchronous communication is simply obtained by the blocking of ask primitives when information on the store is not complete enough to entail the asked constraints. Following these lines, a natural alternative to obtain synchronous communication in CCP is to force ask and tell primitives to synchronise in some way. We elaborate further on this idea by proposing new versions of these primitives.

As in the classical CCP framework, our proposal makes use of tell and ask primitives. However, a new perspective is taken in that, to be reduced, any $tell(c)$ operation needs an $ask(c')$ partner. Restated in other terms, if the tell primitives are seen as producers of new information and ask primitives as consumers, the new primitives consist of lazy tell (or "just-in-time") producers forced to synchronise on their consumer asks. Stress is put on the novelty of information, i.e. on the fact that the told information should not be entailed by the current store. Consequently, any $tell(c)$ and $ask(c)$ operations whose constraint argument c is entailed by the current store can proceed autonomously.

The general scheme is enriched by permitting the synchronisation of more than two partners. Futher we allow some of the tell primitives *not to update* the store. These primitives are subsequently called fictitious and are denoted as $ftell$. They can be used to transmit the information which is not (yet) entailed by the (global/local) store – quite important possibility in distributed systems.

Comparing to the works cited above for synchronisation, we notice the advantage of our approach is that it permits the specification of on *what* information the synchronisation should be made, rather than with *whom*. Our synchronisation is thus more data-oriented as opposed to process-oriented (however still keeping possibility to specify the latter approach as a derived operator). An interesting consequence from a software engineering point of view is that in a specification of an agent (process) it is not necessary to know in advance with which other agents synchronisation should take place. Modularity is thus gained.

Our aim is not to present a new programming language but rather to introduce new variants of tell and ask primitives, to justify our proposal via a comparison with [4] (demonstrating expressiveness), and to present a semantics for them. To sum up our approach allows each agent (i) to maintain its local private store, (ii) to share (read/write) global information in its global stores hierarchy and (iii) to communicate (via multi-party or hand-shakes) with other agents. To achieve this we largely employ standard CCP constructs for hiding of local variables ($\exists_X$) and parameter passing (d_{xy}) as well – see Definition 1.

The rest of this paper is organised as follows. In Section 2 we present the syntax and an informal semantics of `Scc`. To justify our proposal we compare it with the language of [4] in Section 3, while in the Section 4 we give an operational (SOS rules and final result) semantics. We conclude by summarising full semantic treatment of `Scc` and by suggesting some future research directions.

2 Language Scc

This section presents the syntax and the informal semantics of the language underlying the Scc paradigm, also called Scc. As in [11], the constraint system underlying Scc language consists of any system of partial information that supports the entailment relation. We assume a given cylindric constraint system $\langle C, \vdash \rangle$ over a set of variables *Svar*, defined as usual from a simple constraint system $\langle D, \vdash \rangle$ as follows.

Definition 1. *Let Svar be a denumerable set of variables (denoted by $x, y, \ldots$) and let $\langle D, \vdash \rangle$ be a simple constraint system. Let $\mathcal{P}_F(D)$ denote the set of finite subsets of D. For each variable $x \in Svar$ a function $\exists_x : \mathcal{P}_F(D) \to \mathcal{P}_F(D)$ is defined such that for any $c, d \in \mathcal{P}_F(D)$ the conditions (E_1) to (E_4) are satified. Moreover, for each $x, y \in Svar$ the elements $d_{xy} \in D$ are* diagonal elements *iff they satisfy the conditions (E_5) to (E_7).*

(E_1) $c \vdash \exists_x(c)$
(E_2) $c \vdash d$ *implies* $\exists_x(c) \vdash \exists_x(d)$
(E_3) $\exists_x(c \wedge \exists_x(d)) \sim \exists_x(c) \wedge \exists_x(d)$
(E_4) $\exists_x(\exists_y(c)) \sim \exists_y(\exists_x(c))$
(E_5) $\emptyset \vdash d_{xx}$
(E_6) $\{d_{xy}\} \sim \exists z(\{d_{xz}, d_{zy}\})$ *whenever* $x \not\equiv y$
(E_7) $\{d_{xy}\} \wedge \exists x(c \wedge \{d_{xy}\}) \vdash c$

Then $\langle \mathcal{P}(D)_{/\sim}, \vdash \rangle$ is a cylindric constraint system *(over Svar). We denote $\exists_x(c)$ by $\exists_x c$, and for a set $X = \{x_1, \ldots, x_n\}$, we denote $\exists_{x_1} \ldots \exists_{x_n} c$ by $\exists_X c$.*

The language description is parametric with respect to $\langle C, \vdash \rangle$, and so are the semantic constructions presented.

We use G possibly subscripted to range over the set *Sgoal* (processes), $c, d, \ldots$ to range over basic constraints (i.e. constraints which are equivalent to a finite set of primitive constraints), and $X, Y, \ldots$ to range over subsets of *Svar*.

Processes $G \in Sgoal$ are defined by the following grammar

$$G ::= \Delta \mid \texttt{ask}(c) \mid \texttt{tell}(c) \mid \texttt{ftell}(c) \mid G; G \mid G + G \mid G \parallel G \mid \exists_X G \mid p(\boldsymbol{t})$$

Let us briefly discuss an informal meaning of our language constructs. A constant Δ denotes a successfully terminated process. The atomic constructs `ask(c)` and `tell(c)` act on a given store in the following way: as usual, given a constraint `c`, the process `ask(c)` succeeds if `c` is entailed by the store, otherwise it is suspended until it can succeed. However, the process `tell(c)`, of a more lazy nature than the classical one, succeeds only if `c` is (already) entailed by the store and in this case it does not modify the store, and suspends otherwise. It is resumed by a concurrently suspended `ask(d)` operation provided that the conjunction of `c` and of the store entails `d`. In that case, both the `tell` and the `ask` are *synchronously* resumed and the store is atomically augmented with the constraint `c` at the same time. The atomic construct `ftell`(c) behaves as `tell` with the exception that the store is not augmented with the constraint `c`.

The sequential composition $G_1; G_2$ and the nondeterministic choice $G_1 + G_2$ have standard meanings (the latter being a *global* as the selection of a component can be influenced by the store and by the environment of the process as well).

The parallel composition $G_1 \parallel G_2$ represents both the interleaving (merge) of the computation steps of the components involved (provided they can perform these steps independently of each other) and also synchronisation: this is the case of the `tell`, `ftell` and `ask` described above. Note that in the general case there can be a parallel composition of a finite sets of `tell`'s and `ftell`'s and a finite set of `ask`'s such that store and a conjunction of `tell` and `ftell` constraints entails `ask` constraints. In this case all the components synchronise. Sometimes this is called a *multi-party* synchronous communication.

The block construct $\exists_X G$ behaves like a process G with the variables in X considered as local. It hides the information about variables from X within the process G. Finally, $p(\boldsymbol{t})$ is a procedure call, where p is the name of a procedure and $\boldsymbol{t}$ is a list of actual parameters. Its meaning is given w.r.t. a set of procedure declarations or *program*; each such a declaration is a construct of the form $p(x_1, \ldots, x_n) :- G$, where $x_1, \ldots, x_n$ are distinct variables and G is a goal.

Finally, we note it is quite easy to recover the traditional concurrent constraint paradigm within our framework by the introduction of an asynchronous tell. This can be specified by providing, for each constraint to be told, a concurrent corresponding ask operation. Hence this derived operator atell (standing for an asynchronous tell) can be defined as

$$\texttt{atell}(c) : -\texttt{tell}(c) \parallel \texttt{ask}(c).$$

Note the simulation of our primitives by the old ones is not so straightforward and involve auxiliary tells and asks as well as the coding of a manager.

3 Specification of Multi-agent Systems in Scc

In [4] a multi-agent programming language (we will refer to it in this paper as MAL) has been introduced. In this section we show how the exchange of information in multi-agent systems can be defined in Scc. To this end we represent expressions from MAL in the framework of Scc. We want to justify that our language Scc can be seen as a formal multi-agent programming language as well. Furthermore, we show that some aspects of behavior of multi-agent systems which cannot be covered by MAL have their (simple) specifications in Scc. The definition of the language MAL is taken from [4].

In the following definitions we assume a given set *Chan* of communication channels, with typical elements α, and a set *Proc* of procedure identifiers, with typical elements p. We also suppose that the set of variables *Svar* is divided into two disjoint subsets $Svar = ChanVar \cup AgentVar$. Typical elements of *ChanVar* are w, typical elements of *AgentVar* are x, y. The variables from *ChanVar* will be used to model communication via channels, while *AgentVar* is the set of agent's variables. We also suppose that the agent's variables are split into *local* and *global* ones. This is because in *MAL* there is a *global* store that is distributed over the agents. Each agent has direct access only to its *private* store. Information in the private store about the *global* variables can be communicated to the other agents. The *local* variables of an agent cannot be referred to in communications.

Definition 2 (Basic actions). *Given a cylindrical constraint system* $\langle C, \vdash \rangle$ *the basic actions of the programming language MAL are defined as follows:*

$$a ::= \alpha!c \mid \alpha?c \mid ask(c) \mid tell(c)$$

The execution of the output action $\alpha!c$ consists of sending the information c along the channel α, which has to synchronize with a corresponding input $\alpha?d$, for some d with $c \vdash d$. In other words, the information c can be sent along a channel α only if some information entailed by c is requested. The execution of an input action $\alpha?d$, which consists of receiving the information c along the channel α, also has to synchronize with a corresponding output $\alpha!c$, for some c with $c \vdash d$. The execution of a basic action $ask(c)$ by an agent consists of checking whether the private store of the agent entails c. On the other hand, the execution of $tell(c)$ consist of adding c to the private store.

Representing Basic Actions in Scc
With each channel α we associate a variable w_α from *ChanVar*. The actions $ask(c)$ and $tell(c)$ behave equally in both languages, hence are represented by the same expressions. Sending information along a channel α is modeled by the Scc action $\texttt{ftell}(w_\alpha = true \wedge c)$. This action has to synchronize with the corresponding `ask` action. As `ftell` does not update information on the store, the corresponding `ask` must be sequentially followed by an asynchronous `tell` action which will actually store the information. Hence, receiving of information is modeled as a sequence $\texttt{ask}(w_\alpha = true \wedge c); \texttt{atell}(c)$. The representation of MAL basic actions in the `Scc` is summarized in the following table.

MAL	`Scc`
$ask(c)$	$\texttt{ask}(c)$
$tell(c)$	$\texttt{tell}(c)$
$\alpha!c$	$\texttt{ftell}(w_\alpha = true \wedge c)$
$\alpha?c$	$\texttt{ask}(w_\alpha = true \wedge c); \texttt{atell}(c)$

Definition 3 (Statements). *MAL agents (statement* S*) are defined as:*

$$S ::= a.S \mid S_1 + S_2 \mid S_1 \& S_2 \mid \exists_x S \mid p(\overline{x})$$

Statements are thus built up from the basic actions using the following standard programming constructs: action prefixing (denoted by "."), non-deterministic choice (denoted by "+"), internal parallelism (denoted by "&"), local variables (denoted by $\exists_x S$, which indicates that x is a local variable in S), and (recursive) procedure calls of the form $p(\bar{x})$, where $\bar{x}$ denotes a sequence of variables which constitute the actual parameters of the call.

Representing Statements in Scc
With the exeption of prefixing, all the statements are directly represented by the corresponding Scc expressions. Prefixing is modeled by sequential composition.

MAL	Scc
$a.S$	$a; S$
$S_1 + S_2$	$S_1 + S_2$
$S_1 \& S_2$	$S_1 \parallel S_2$
$\exists_x S$	$\exists_x S$
$p(\overline{x})$	$p(\overline{x})$

Definition 4 (Multi-agent systems). *A multi-agent system A of MAL is as*

$$A ::= < D, S, \sigma > | \; A_1 \parallel A_2 \mid \delta_H(A)$$

A basic agent in a multi-agent system is represented by a tuple $< D, S, c >$ consisting of a set D of procedure declarations of the form $p(\bar{x}) :- S$, where $\bar{x}$ denote the formal parameters of p and S denotes its body. The statement S in $< D, S, c >$ describes the behavior of the agent with respect to its private store c. The threads of S, i.e. the concurrently executing sub-statements of S, interact with each other via the private store of the basic agent by means of the actions $ask(d)$ and $tell(d)$. Additionally, a multi-agent system itself consists of a collection of concurrently operating agents that interact with each other only via a synchronous information-passing mechanism by means of the communication actions $\alpha!d$ and $\alpha?d$. (In [4] authors provide the parallel composition of agent systems only; the semantic treatment of sequential and the non-deterministic composition of agent systems is standard.)

Representing Multi-agent Systems in Scc
The parallel operator $\parallel$ is represented as the asynchronous parallel operator $\parallel$ of Scc. The operator $\delta_H(A)$ is represented as $\exists_{w_H} A$. The encapsulation is thus achieved by making the channels from H local. The communication among concurrently operating agents is achieved by *synchronous* communication mechanism of Scc. In particular, the pair `ask`, `ftell` allows to synchronously communicate information between two agents without storing information on the global store. On the other hand the pair `ask`, `tell` allows for multi-agent communication among several process and with storing the communicated information. We summarize the translation in the following table.

MAL	Scc
$< D, S, c >$	Scc program
$A_1 \parallel A_2$	$A_1 \parallel A_2$
$\delta_H(A)$	$\exists_{w_H}(A)$

Global Multi-agent Communication

In contrast to *MAL* our `Scc` language allows asynchronous and synchronous multi-agent communication. Besides a synchronous `ftell` action, a `Scc` agent can also perform `tell`, `ftell` and `ask` actions on the *global* store. If an agent uses `atell` then it just communicates some piece of information to all processes,

i.e. it makes information generally accessible. If an agent uses a synchronous `tell` action on the global store, then there must be at least one agent waiting for this information and the communication is synchronous in this case. However, as information is stored into the global store in this case it would be accessible to any agent. This more general way of transmitting information among agents, makes `Scc` more general and more flexible language for specification and implementation of the exchange of information in multi-agent systems as is *MAL*.

4 Operational Semantics $\mathcal{O}$ of `Scc`

Contexts
It turns out that it is possible to treat the sequential and parallel composition operators of `Scc` in a very similar way by introducing the auxiliary notion of *context*. Basically, a context consists of a partially ordered structure where place holders (subsequently referred to by $\Box$) have been inserted at a top-level place, i.e. a place not constrained by the previous execution of other atoms. Viewing goals as partially ordered structures too, the ask and tell primitives to be reduced are those which can be substituted by a place holder $\Box$ in a context. Furthermore, the goals resulting from the reductions are essentially obtained by substituting the place holder by the corresponding clause bodies or the Δ, depending upon whether an atom or a ask/tell primitive is considered.

Definition 5. *Contexts are functions inductively defined on goals as follows:*

1. *A nullary context is associated with any goal. It is represented by the goal and is defined as the constant mapping from Sgoal to this goal with the goal as value.*
2. *$\Box$ is a unary context that maps any goal to itself. For any goal G, this application is subsequently referred to as $\Box[G]$. Thus $\Box[G] = G$ for any goal.*
3. *If tc is an n-ary context and if G is a goal, then $(tc; G)$ is an n-ary context. Its application is defined as follows : for any goals $G_1, \ldots, G_n$,*

$$(tc; G)[G_1, \cdots, G_n] = (tc[G_1, \cdots, G_n]; G)$$

4. *If tc_1 and tc_2 are m-ary and n-ary contexts then $tc_1 \parallel tc_2$ is an (m+n)-ary context. Its application is defined as follows: for any goals $G_1, \ldots, G_{m+n}$,*

$$(tc_1 \parallel tc_2)[G_1, \cdots, G_{m+n}] = (tc_1[G_1, \cdots, G_m]) \parallel (tc_2[G_{m+1}, \cdots, G_{m+n}])$$

In what follows the goals are considered modulo syntactical congruence induced by associativity of ";", "$\|$" and "+", by commutativity of "$\|$" and "+", and Δ as the unit element. Also we will simplify the goals resulting from the application of contexts accordingly.

Transition System
The operational semantics of `Scc` is defined in Plotkin's style ([8]) by means of a transition system, which is itself defined by rules of the form

$$\frac{Assumptions}{Conclusion} \quad if \quad Conditions$$

where *Assumptions* and *Conditions* may possibly be absent. Configurations traditionally describe the statement to be computed and a state summing up the computations made so far. Rephrased in the `Scc` context, the configurations to be considered here comprise a goal to be reduced together with a store. In the following definition $Sstore$ denotes the set of stores.

Definition 6. *The transition relation $\rightarrow$ is defined as the smallest relation of $(Sgoal \times Sstore) \times (Sgoal \times Sstore)$ satisfying the rules[1] of Figure 1. We write $<G, \sigma> \rightarrow <G', \sigma'>$ rather than $(<G, \sigma>, <G', \sigma'>) \in \rightarrow$.*

$$\text{(T)} \quad <tc[sp_1, \cdots, sp_m], \sigma> \rightarrow <tc[\Delta, \cdots, \Delta], \tau>$$

$$if \begin{cases} \{sp_1, \cdots, sp_m\} = \{\ ask(a_1), \cdots, ask(a_p), \\ \qquad\qquad\qquad\quad tell(at_1), \cdots, tell(at_q), \\ \qquad\qquad\qquad\quad tell(rt_1), \cdots, tell(rt_r), \\ \qquad\qquad\qquad\quad ftell(af_1), \cdots, ftell(af_s), \\ \qquad\qquad\qquad\quad ftell(rf_1), \cdots, ftell(rf_t)\ \ \}, \\ \sigma \cup \{rt_1, \cdots, rt_r\} \cup \{rf_1, \cdots, rf_t\} \\ \quad \vdash \{a_1, \cdots, a_p\} \cup \{at_1, \cdots, at_q\} \cup \{af_1, \cdots, af_s\}, \\ \text{there is no strict subset } S \text{ of } \{rt_1, \cdots, rt_r\} \cup \{rf_1, \cdots, rf_t\} \\ \text{ such that } \sigma \cup S \vdash \{a_1, \cdots, a_p\} \cup \{at_1, \cdots, at_q\} \cup \{af_1, \cdots, af_s\}, \\ \tau = \sigma \cup \{rt_1, \cdots, rt_r\},\ \ m > 0 \end{cases}$$

Fig. 1. `Scc` transition rules for new versions of aks and tells

The Operational Semantics

Rules for the sequential and parallel composition operators are tackled by means of the notion of context within the rule (T).

The rule (T) defines reductions of `tell`, `ftell` and `ask` primitives. The primitives to be reduced, referred to as $sp_1, \ldots, sp_m$, are partitioned in five categories: (1) the `ask` primitives (the multi-set $\{ask(a_1), \cdots, ask(a_p)\}$),
the `tell` primitives split into (2) those which add information to the store (the multi-set $\{tell(rt_1), \cdots, tell(rt_r)\}$) and (3) those which do not (the multi-set $\{tell(at_1), \cdots, tell(at_q)\}$, i.e. already entailed), and
the fictitious primitives `ftell` split in a similar way into (4) the multi-sets $\{ftell(rf_1), \cdots, ftell(rf_t)\}$ and (5) $\{ftell(af_1), \cdots, ftell(af_s)\}$, respectively.

All these primitives are then simultaneously reduced to the empty goal Δ when information on the current store (σ) together with new information told $(rt_1, \ldots, rt_r, rf_1, \ldots, rf_t)$ entails information of the other primitives. The new

[1] Please note that due to lack of space we do not give the very standard rules for a nondeterministic choice, hiding and a procedure call in Figure 1

store consists in this case of the old store enriched by new information told. Note that this rule reflects the laziness feature of our tell primitives.

An `ask(c)` primitive for a constraint c entailed by the current store σ can be reduced alone following rule (T) by taking the unary context $\Box$, $m = 1$, $p = 1$, $q = 0$, $r = 0$, $s = 0$, $t = 0$. The axiom

$$<\texttt{ask}(c), \sigma> \rightarrow <\Delta, \sigma> \qquad \text{if} \quad \{\sigma \vdash c\} \tag{1}$$

results from rule (T) as the particular case.

A `tell(c)` primitive for a constraint c entailed by the current store σ can be reduced alone following rule (T) by taking the unary context $\Box$, $m = 1$, $p = 0$, $q = 1$, $r = 0$, $s = 0$, $t = 0$. The axiom

$$<\texttt{tell}(c), \sigma> \rightarrow <\Delta, \sigma> \qquad \text{if} \quad \{\sigma \vdash c\} \tag{2}$$

results from rule (T) as the particular case as well.

Other tell's and ask's need each other for reduction and reduce simultaneously. A minimality condition (see the side condition of (T)) is required to forbid outsider tell's to be reduced by taking advantage of a concurrent reduction.

To define the operational semantics we follow the logic programming tradition – it specifies the final store of the successful computations. It also indicates those stores corresponding to deadlock situations and distinguishes between two types: *failure* corresponding to the absence of suitable procedure declarations to reduce procedure calls and *suspension* corresponding to the absence of suitable data on the store or of concurrent processes that would allow tell and ask primitives to proceed, i.e. to suspended tell's and ask's. Note that, as illustrated by axioms (1) and (2) above, the two situations may be distinguished by a simple criterion: the existence of a store richer than the current one that would enable the computation to proceed. The following definition is based on this intuition. The symbols δ^+, δ^-, and δ^s are used to indicate the computations ending by a success, a failure, and a suspension, respectively.

Definition 7. *Operational semantics* $\mathcal{O} : Sgoal \rightarrow \mathcal{P}(Sstore \times \{\delta^+, \delta^s, \delta^-\})$ *is defined as the following function: for any goal* G,

$$\begin{aligned}
\mathcal{O}(G) = & \{\ <\tau, \delta^+> : <G, true> \rightarrow \cdots \rightarrow <\Delta, \tau>\ \} \\
& \cup \{\ <\tau, \delta^s> : <G, true> \rightarrow \cdots \rightarrow <G', \tau> \not\rightarrow,\ \textit{where } G' \neq \Delta \textit{ and} \\
& \qquad \textit{there are } \sigma', G'', \sigma'' \textit{ such that } <G', \sigma'> \rightarrow <G'', \sigma''>\} \\
& \cup \{\ <\tau, \delta^-> : <G, true> \rightarrow \cdots \rightarrow <G', \tau> \not\rightarrow,\ \textit{where } G' \neq \Delta \textit{ and} \\
& \qquad \textit{for any } \sigma', <G', \sigma'> \not\rightarrow\ \}
\end{aligned}$$

5 Conclusions

We have presented a language for a *specification* of the exchange and/or the global sharing of information in multi-agent systems. It is solely based on concurrent constraint programming paradigm with slightly modified test and updates operations. We have briefly compared it to the latest proposal within this area as given in [4].

Due to a lack of space we have presented an operational semantics only. Reader can easily verify it is not compositional. We refer to our studies in [2],

where a compositional semantics is given and proved to be correct with respect to the semantics $\mathcal{O}$ (it is based on 'hypothetical' steps which can be made by both concurrent agents and the state of global store rather than successive updates). In [3] an algebraic (failure) semantics is defined for a subset of `Scc` (only finite behaviours and without `ftell`s). The algebraic semantics is proved to be sound and complete with respect to a compositional operational semantics. Allowing handshake communications only in the mentioned subset, we proposed [1] a denotational semantics. We employed so-called testing techniques – this semantics uses monotonic sequences of labelled pairs of input-output states, possibly containing "hypothetical" gaps, and ending with marks reporting success or failure (to follow logic programming tradition). This semantics is proved to be correct with respect to the operational semantics and fully abstract as well.

Our future work aims at designing fully abstract semantics for the full version of `Scc`. Also we study possibilities of incorporating (discrete) real-time aspects.

References

1. L. Brim, D. Gilbert, J-M. Jacquet, and M. Křetínský. A fully abstract semantics for a version of synchronous concurrent constraint programming. Technical Report FIMU-RS-99-08, Faculty of Informatice, MU Brno, 1999.
2. L. Brim, D. Gilbert, J-M. Jacquet, and M. Křetínský. New Versions of Ask and Tell for Synchronous Communication in Concurrent Constraint Programming. Technical report, TCU/CS/1996/03, City University London, ISSN1364-4009, 1996.
3. L. Brim, D. Gilbert, J-M. Jacquet, and M. Křetínský. A Process Algebra for Synchronous Concurrent Constraint Programming. In *Proceedings of ALP96*, LNCS, pages 24–37. Springer-Verlag, 1996.
4. F. de Boer, R. van Eijk, M. van der Hoek, and Ch. Meyer. Failure semantics for the exchange of information in multi-agent systems. In *CONCUR: 11th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2000.
5. Frank S. de Boer, Joost N. Kok, Catuscia Palamidessi, and Jan J. M. M. Rutten. Non-monotonic concurrent constraint programming. In Dale Miller, editor, *Logic Programming - Proceedings of the 1993 International Symposium*, pages 315–334, Vancouver, Canada, 1993. The MIT Press.
6. M. Falaschi, G. Levi, and Catuscia Palamidessi. A Synchronization Logic: Axiomatics and Formal Semantics of Generalized Horn Clauses. *Information and Control*, 60:36–69, 1994.
7. T. Finin, D. McKay, R. Fritzson, and R. McEntire. KQML: An information and knowledge exchange protocol. In *Knowledge Building and Knowledge Sharing*. Ohmsha and IOS Press, 1994.
8. G. Plotkin. A structured approach to operational semantics. Technical report, Tech.Rep. DAIMI FN-19, Computer Science Dept., Aarhus University, 1981.
9. J-H. Rety. *Langages concurrents avec constraintes, communication par messages at distribution*. Phd thesis, University of Orleans, 1997.
10. Vijay Saraswat. *Concurrent Constraint Programming*. MIT Press, 1993.
11. Vijay Saraswat, Martin Rinard, and Prakash Panangaden. Semantic foundations of concurrent constrant programming. In *Proc. of the 18th POPL*. ACM, 1991.
12. M. Wooldridge. Verifiable semantics for agent communication languages. In *ICMAS'98*. IEEE Computer Press, 1998.

ADST: An Order Preserving Scalable Distributed Data Structure with Constant Access Costs

Adriano Di Pasquale[1] and Enrico Nardelli[1,2]

[1] Dipartimento di Matematica Pura ed Applicata, Univ. of L'Aquila,
Via Vetoio, Coppito, I-67010 L'Aquila, Italia.
{dipasqua,nardelli}@univaq.it
[2] Istituto di Analisi dei Sistemi ed Informatica, Consiglio Nazionale delle Ricerche,
Viale Manzoni 30, I-00185 Roma, Italia.

Abstract. Scalable Distributed Data Structures (SDDS) are access methods specifically designed to satisfy the high performance requirements of a distributed computing environment made up by a collection of computers connected through a high speed network. In this paper we propose an order preserving SDDS with a worst-case constant cost for exact-search queries and a worst-case logarithmic cost for update queries. Since our technique preserves the ordering between keys, it is also able to answer to range search queries with an optimal worst-case cost of $O(k)$ messages, where k is the number of servers covering the query range. Moreover, our structure has an amortized almost constant cost for any single-key query.
Hence, our proposal is the first solution combining the advantages of the constant worst-case access cost featured by hashing techniques (e.g. LH*) and of the optimal worst-case cost for range queries featured by order preserving techniques (e.g., RP* and DRT). Furthermore, recent proposals for ensuring high-availability to an SDDS can be easily combined with our basic technique. Therefore our solution is a theoretical achievement potentially attractive for network servers requiring both a fast response time and a high reliability.
Finally, our scheme can be easily generalized to manage k-dimensional points, while maintaining the same costs of the 1-dimensional case.

Keywords: scalable distributed data structure, message passing environment, multi-dimensional search.

1 Introduction

With the striking advances of communication technology, distributed computing environments become more and more relevant. This is particularly true for the technological framework known as *network computing*: a fast network interconnecting many powerful and low-priced workstations, creating a pool of perhaps terabytes of RAM and even more of disc space. This is a computing environment

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 211–222, 2001.

very apt to manage large amount of data and to provide high performances. In fact, the large amount of RAM collectively available combined with the speed of the network allow the so-called *RAM data management*, which can deliver performances not reachable using standard secondary memory.

A general paradigm to develop access methods in such distributed environments was proposed by Litwin, Neimat and Schneider [9]: *Scalable Distributed Data Structures* (SDDSs). The main goal of an access method based on the SDDS paradigm is the management of very large amount of data implementing efficiently standard operations (i.e. inserts, deletions, exact searches, range searches, etc.) and aiming at *scalability*, i.e. the capacity of the structure to keep the same level of performances while the number of managed objects changes and to avoid any form of bottleneck. In particular, a typical distributed structure made up by a set of data server and a unique server directory cannot be considered an SDDS.

The main measure of performance for a given operation in the SDDS paradigm is the number of point-to-point messages exchanged by the sites of the network to perform the operation. Neither the length of the path followed in the network by a message nor its size are relevant in the SDDS context. Note that, some variants of SDDS admit the use of multicast to perform range query.

There are several SDDS proposals in the literature: defining structures based on hashing techniques [3,9,12,15,16], on order preserving techniques [1,2,4,8,10], or for multi-dimensional data management techniques [11,14], and many others.

LH* [9] is the first SDDS that achieves worst-case constant cost for exact searches and insertions, namely 4 messages. It is based on the popular linear hashing technique. However, like other hashing schemes, while it achieves good performance for single-key operations, range searches are not performed efficiently. The same is true for any operation executed by means of a scan involving all the servers in the network.

On the contrary, order preserving structures achieve good performances for range searches and a reasonably low (i.e. logarithmic), but not constant, worst-case cost for single key operations. Among order preserving SDDSs, we recall RP*s [10], based on the B^+-tree technique and BDST [4], based on balanced binary search tree. Both these structures achieves logarithmic costs for single key operations in the worst-case. Structures in the DRT family [5,8] can guarantee only a linear bound in the worst-case, but provide very good performances in the amortized case [5]. Finally, Distributed B^+-tree [2] is the first order preserving structures with constant exact search worst-case cost, but at the price of a linear worst-case cost for insertion.

Here we further develop the technique presented in [2] with the major objective to keep logarithmic the worst-case cost of insertions. This allows to obtain the following results: (i) worst-case constant cost for exact searches and insertions that do not causes splits, namely 4 messages; (ii) worst-case logarithmic cost for insertions that causes splits; (iii) amortized almost constant cost for any single-key operations.

Therefore, this is the first order preserving SDDS proposal achieving single-key performances comparable with the LH*, while continuing to provide the good worst-case complexity for range searches typical of order preserving access methods, like RP* and DRT.

Our structure is also able to support deletions: these are not explicitly considered in previous proposals in the literature, but for BDST [4] and, to some degree, for LH* [12]. Moreover, the technique used in our access method can be applied to the distributed k-d tree [14], an SDDS for managing k-dimensional data, with similar results.

2 ADST

We now introduce our proposal for a distributed search tree, that can be seen as a variant of the systematic correction technique presented in [2]. We first present the basic technique and then discuss our variation.

Each server manages a unique *bucket* of keys. The bucket has a fixed capacity b. We define a server "to be in overflow" or "to go in overflow" when it manages b keys and one more key is assigned to it. When a server s goes in overflow it starts the *split* operation. After a split, s manages $\frac{b}{2}$ keys and $\frac{b}{2}+1$ keys are sent to a new server s_{new}. It is easy to prove the following property:

Lemma 1. *Let σ be a sequence of m intermixed insertions and exact searches. Then we may have at most $\lfloor \frac{m}{A} \rfloor$ splits, where $A = \frac{b}{2}$.*

Moreover, clients and servers have a local indexing structure, called *local tree*. This is needed to avoid clients and servers to make address errors. From a logical point of view the local tree is an incomplete collection of associations $\langle server, interval\ of\ keys \rangle$: for example, an association $\langle s, I(s) \rangle$ identifies a server s and the managed interval of keys $I(s)$.

For further details on buckets and local trees management see [2,8].

Let us consider a split of a server s with a new server s'. Given the leaf f associated to s, a split conceptually creates a new leaf f' and a new internal node v, father of the two leaves. This virtual node is associated to s or to s'. Which one is chosen is not important: we assume to associate it always with the new server, in this case s'. s stores s' in the list l of servers in the path from the leaf associated to itself and the root. s' initializes its corresponding list l' with a copy of the s' one (s' included).

Moreover if this was the first split of s, then s identifies s' as its *basic server* and stores it in a specific field. Please note that the interval $I(v)$ now corresponds to the *basic interval* of s.

After the split s sends a correction message containing the information about the split to s' and to the other servers in l. Each server receiving the message corrects its local tree. Each list l of a server s corresponds to the path from the leaf associated with s to the root.

This technique ensures that a server s_v associated to a node v knows the exact partition of the interval $I(v)$ of v and the exact associations of elements

of the partition and servers managing them. In other words the local tree of s_v contains all the associations $\langle s', I(s')\rangle$ identifying the partition of $I(v)$. Please note that in this case $I(v)$ corresponds to $I(lt(s_v))$.

This allows s_v to forward a request for a key belonging to $I(v)$ (i.e. a request for which s_v is logically pertinent) directly to the right server, without following the tree structure. In this distributed tree, rotations are not applied, then the association between a server and its basic server never changes.

Suppose a server s receives a requests for a key k. If it is pertinent for the requests ($k \in I(s)$) then it performs the request and answers to the client. Otherwise if it is logically pertinent for the requests ($k \in I(lt(s))$) then it finds in its local tree $lt(s)$ the pertinent server and forwards it the requests. Otherwise it forwards the requests to its basic server s'. We recall that $I(lt(s'))$ corresponds to the basic interval of s, then, as stated before, if the request for k is arrived to s, k has to belong to this interval. Then s' is certainly logically pertinent.

Therefore a request can be managed with at most 2 address errors and 4 messages.

The main idea of our proposal is to keep the path between any leaf and the root short, in order to reduce the cost of correction messages after a split. To obtain this we aggregate internal nodes of the distributed search tree obtained with the above described techniques in compound nodes, and apply the technique of the Distributed B$^+$-tree to the tree made up by compound nodes. For this reason we call our structure ADST (Aggregation in Distributed Search Tree).

Please note that the aggregation only happens at a logical level, in the sense that no additional structure has to be introduced. What happens in reality is simply that a server associated to a compound node maintains the same information maintained by the one associated to an internal node in the Distributed B$^+$-tree.

Each server s in ADST is conceptually associated to a leaf f. Then, as a leaf, s stores the list l of servers managing compound nodes in the path from f and the (compound) root of the ADST. If s has already split at least one time, then it stores also its *basic server* s'. In this case s' is a server that manages a compound node and such that $I(lt(s'))$ contains the *basic interval* of s.

Any server records in a field called *adjacent* the server managing the adjacent interval on its right. Moreover, if s manages also a compound node $va(s)$, then it also maintains a local tree, in addition to the other information.

2.1 Split Management

Let s be a server conceptually associated to a leaf f, and let the father compound node va^* of f be managed by a server s^*. Let us now consider a split of s with s_{new} as new server. The first operation performed by s is to send correction messages to each server in l.

Then, exactly like in the technique described for distributed B$^+$-tree [2], a new leaf f' and a new internal node v father of the two leaves are conceptually created. v is associated to s_{new}.

In ADST two situations are possible:

- The node v has to be *aggregated* with the compound node va^*. Then v is released, s does not change anything in its list l and s_{new} initializes its list l_{new} with a copy of l. If this was the first split of s, then s identifies s^* as its *basic server* and stores it in a specific field. Please note that the interval $I(lt(s^*))$ contains the *basic interval* of s.
- The node v has not to be *aggregated* with the compound node va^*. Then a new compound node va is created as a son of va^* aggregating the single internal node v. s_{new} is called to manage va. s changes its list l adding s_{new}. s_{new} initializes its list l_{new} with a copy of l. If this was the first split of s, then s identifies s_{new} as its *basic server* and stores it in a specific field. Please note that the interval $I(lt(s_{new}))$ is now exactly the *basic interval* of s.

The field *adjacent* of s_{new} is set with the value stored in the field of s. The field *adjacent* of s is set with s_{new} (see figure 1).

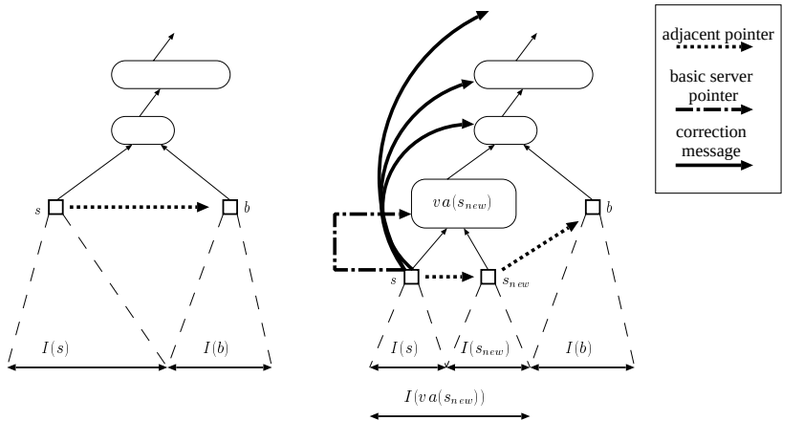


Fig. 1. Before (left) and after (right) the split of server s with s_{new} as new server. Intervals are modified accordingly. Correction messages are sent to server managing compound nodes stored in the list $s.l$ and *adjacent* pointers are modified. Since the aggregation policy decided to create a new compound node and s_{new} has to manage it, then s_{new} is added to the list $s.l$ of servers between the leaf s and the compound root nodes, s_{new} sets $s_{new}.l = s.l$. If this is the first split of s, then s sets s_{new} as its *basic server*

2.2 Aggregation Policy

The way to create compound nodes in the structure is called *aggregation policy*. We require that an aggregation policy creates compound nodes so that the height

of the tree made up by the compound nodes is logarithmic in the number of servers of the ADST. In such a way the cost of correcting the local trees after a split is logarithmic as well.

One can design several aggregation policies, satisfying the previous requirement. The one we use is the following.

(AP): To each compound node va a bound on the number of internal nodes $l(va)$ is associated. The bound of the root compound node ra is $l(ra) = 1$. If the compound node va' father of va has bound $l(va')$, then $l(va) = 2l(va') + 1$.

Suppose a server associated to a leaf son of va splits. If the bound $l(va)$ is not reached, va aggregates the new internal nodes v. Otherwise a new compound node has to be created as a son of va and aggregating v.

It is easy to prove the following:

Invariant 1 *Let va_0 be the compound root node. Let $va_0, va_1, ..., va_k$ be the compound nodes in the path between va_0 and a leaf. Then* $\#\text{internal\_nodes}(va_i) = l(va_i)$*, for each* $0 \leq i \leq k-1$*, and* $\#\text{internal\_nodes}(va_k) \leq l(va_k)$.

With reference to figure 2, we have for example: $I(a) = A$, $I(b) = B$, and so on; $a.adjacent = b$, $b.adjacent = c$, $c.adjacent = q$, and so on; $a.l = \{a\}$, $b.l = \{c, a\}$, $c.l = \{q, c, a\}$, and so on; $lt(f) = \{\langle d, D\rangle, \langle f, F\rangle, \langle g, G\rangle, \langle h, H\rangle, \langle i, I\rangle, \langle l, L\rangle, \langle o, O\rangle, \langle p, P\rangle, \langle m, M\rangle, \langle n, N\rangle\}$ and then $I(va(f)) = I(lt(f)) = D \cup F \cup G \cup H \cup I \cup L \cup O \cup P \cup M \cup N$; from the given sequence of splits, $a.basic\_server = a$, $b.basic\_server = c$, $c.basic\_server = c$, and so on.

Theorem 2. *Aggregation policy AP guarantees that the length of any path between a leaf and the compound root node is bounded by $h_a = k \leq \lfloor \log n \rfloor + 1$, where n is the number of internal nodes of the distributed tree.*

Proof. Let us consider a generic leaf f, and let $h_a = k$ be its height in the tree of compound nodes. Then there are k compound nodes $va_0, va_1, .., va_{k-1}$ in the path between f and the root compound node, and va_0 is just the compound root node. It follows directly from the definition of policy AP that: $\#internal\_nodes(va_0) = l(va_0) = 1$, $\#internal\_nodes(va_1) = l(va_1) = 2^1 + 1$, $\#internal\_nodes(va_2) = l(va_2) = 2^2 + 1$, ..., $\#internal\_nodes(va_{k-2}) = l(va_{k-2}) = 2^{k-2} + 1$ and $\#internal\_nodes(va_{k-1}) \geq 1$. Then the number n of internal nodes, in the case $k > 1$, is such that:

$$n \geq 1 + \sum_{i=1}^{k-2}(2^i + 1) + 1 = k - 2 + \frac{2^{k-1} - 1}{2 - 1} + 1 = k - 2 + 2^{k-1}$$

$$\Rightarrow n \geq 2^{k-1}, \forall k \geq 1$$

Then we have $h_a = k \leq \lfloor \log n \rfloor + 1$.

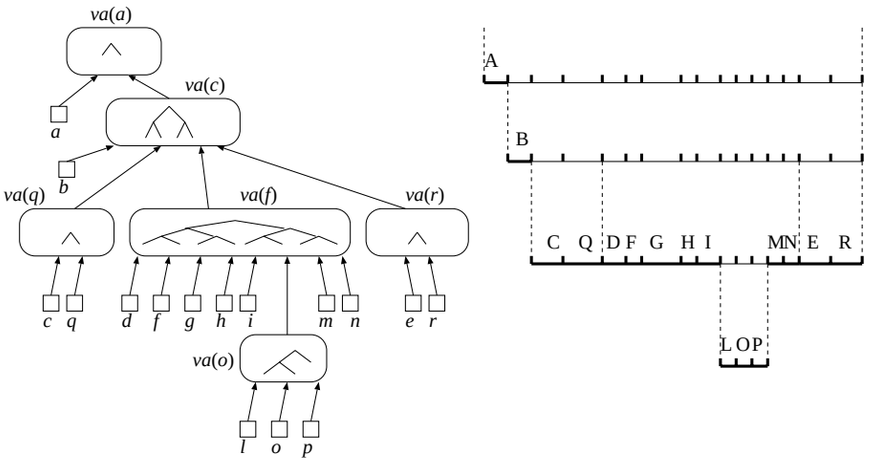


Fig. 2. An example of ADST with policy AP. Lower-case letters denote servers and associated leaves, upper-case letters denote intervals of data domain. The sequence of splits producing the structure is $a \to b \to c \to d \to e$, then $d \to f \to g \to h \to i \to l \to m \to n$, then $l \to o \to p$, then $c \to q$ and finally $e \to r$, meaning with $x \to y$ that the split of x creates the server y

We recall that for the binary tree we are considering, that is a tree where every node has 0 or two sons, the number of leaves, and then of servers, is $n = n' + 1$, where n' is the number of internal nodes. Substantially the previous theorem states that the cost of correcting local trees after a split is of $O(\log n)$ messages, where n is the number of servers in the ADST.

2.3 Access Protocols

We now analyze what happens when a client c has to perform a single-key request for a key k. We describe the case of exact search, insertions and deletion:

Exact Search: c looks for the pertinent server for k in its local tree, finds the server s, and sends it the request. If s is pertinent, it performs the request and sends the result to c.

Suppose s is not pertinent. If s does not manage a compound node, then it forwards the request to its *basic server* s'. We recall that $I(lt(s'))$ includes the basic interval of s, then, as stated before, if the request for k is arrived to s, k has to belong to this interval. Therefore s' is certainly logically pertinent: it looks for the pertinent server for k in its local tree and finds the server s''. Then s' forwards the request to s'', which performs the request and answers to c. In this case c receives the local tree of s' in the answer, so to update its local tree (see figure 3).

Suppose now that s manages a compound node. The way in which compound nodes are created ensures that $I(lt(s))$ includes the basic interval of s itself. Then

s has to be logically pertinent, hence it finds in $lt(s)$ the pertinent server and sends it the request. In this case c receives the local tree of s in the answer.

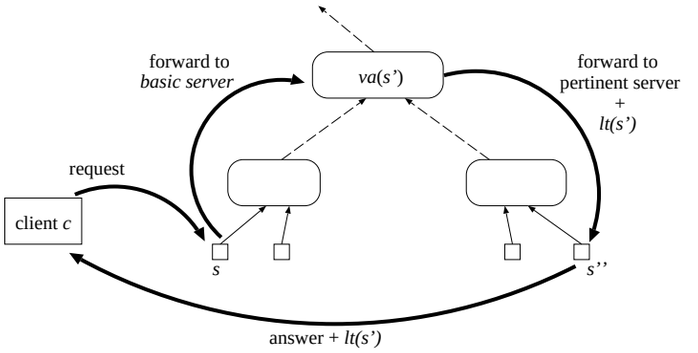


Fig. 3. Worst-case of the access protocol

Insertion: the protocol for exact search is performed in order to find the pertinent server s for k. Then s inserts k in its bucket. If this insertion causes s to go in overflow then a split is performed. After the split, correction messages are sent to the servers in the list l of s.

Deletion: the protocol for exact search is performed in order to find the pertinent server s for k. Then s deletes k from its bucket. If this deletion causes s to go in underflow then a *merge*, which is the opposite operation of a split, is performed.

We consider a server to be in underflow whenever it manages less than $\frac{b}{d}$ keys in the bucket, for a fixed constant d. The merge operation consists basically in releasing an existing server s which is in underflow. If s is not empty, it sends its remaining keys to another server s'. From now on $I(s')$ is enlarged by uniting it with $I(s)$. After the merge, correction messages are sent to the servers in list l of s. Moreover, an algorithm to preserve the invariant of policy AP is applied after a merge. A detailed presentation of this algorithm is quite long, and is left to the extended version of this paper. Please note that s' is only chosen if it is able to receive all the keys of s without going in overflow. If there is not such a server, a server s^*, whose interval is adjacent to $I(s)$, is chosen and it sends a proper number of keys to s in order to allow s to exit from the underflow state. $I(s)$ and $I(s^*)$ are modified accordingly.

Previous SDDSs, e.g LH*, RP*, DRT, etc., do not explicitly consider deletions. Hence, in order to compare ADST and previous SDDSs performances, we shall not analyze behavior of ADST under deletions.

2.4 Range Search

We now describe how a range search is performed in ADST.

The protocol for exact search is performed in order to find the server s pertinent for the leftmost value of the range. If the range is not completely covered by s, then s sends the request to server s' stored in its field *adjacent*. s' does the same. Following the adjacent pointers all the servers covering the range are reached and answer to the client. The operation stops whenever the server pertinent for the rightmost value of the range is reached, see figure 4.

The above algorithm is very simple and applies to the case when only the point-to-point protocol is available. Other variants can be considered, for example a client can send more than one request messages, if it discovers from its local tree that more than one server intersects the range of the query.

Usually whenever the range of a query is large, the multicast protocol is applied, if it is available in the considered technological framework. The same technique can be applied for ADST as well.

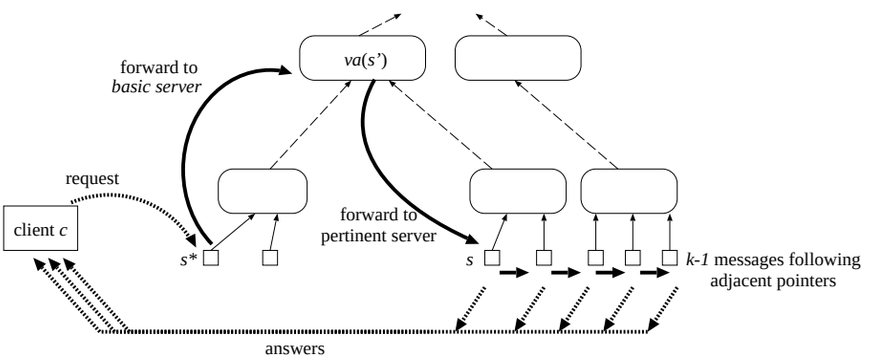


Fig. 4. Worst-case of the range search. The server s is pertinent for the leftmost value of the range

3 Complexity Analysis

In what follows we suppose to operate in an environment where clients work slowly. More precisely, we suppose that between two consecutive requests the involved servers have the time to complete all updates of their local trees: we call this a low concurrency state. Any communication complexity result in SDDS proposals is based on this assumption. In the extended version of this paper [6] we show how the complexity analysis of SDDSs is influenced by this assumption, and we give some ideas on how to operate in the case of fast working clients (also called high concurrency).

3.1 Communication Complexity

The basic performance parameter for an SDDS is the communication complexity, that is the number of messages needed to perform the requests of clients. We present the main results obtained with ADST for this parameter.

Theorem 3. *An exact search has in an ADST a worst-case cost of 4 messages. An insertion that does not cause a split and a deletion that does not cause a merge has also in an ADST a worst-case cost of 4 messages.*

Proof. Follows directly from the presented algorithms.

Theorem 4. *A split and the following corrections of local trees have in an ADST a worst-case cost of* $\log n + 5$.

Proof. Follows directly from the fact that a split costs 4 messages and from theorem 2.

Theorem 5. *A range search has in an ADST a worst-case cost of* $k + 1$ *messages, where* k *is the number of servers covering the range of query, without accounting for the single request message and the* k *response messages.*

Proof. Follows directly from the algorithm. Theorem 3 ensures that the search of server covering the leftmost limit of the range adds, without accounting for the requests and answer messages, 2 messages at the cost. Then we add other $k - 1$ messages to reach by following the *adjacent* pointers the remaining $k - 1$ servers covering the range.

As presented in sub-section 2.3, in ADST a merge is followed by the correction of local trees and by a restructuring of the tree made up by compound nodes in order to keep the invariant 1. In the extended version of this paper we prove the following theorem.

Theorem 6. *In ADST a merge costs* $O(\log n)$ *messages, accounting the local trees correction messages and the restructuring algorithms.*

The following theorems show the behavior of ADST in the amortized case.

Theorem 7. *A sequence of intermixed exact searches and insertions on ADST has an amortized cost of* $4 + \frac{2(\log n+5)}{b}$ *messages per operation, where* b *is the capacity of a bucket.*

Proof. Let us consider a sequence of m intermixed exact searches and insertions performed on ADST. From lemma 1 we have at most $\lfloor \frac{m}{A} \rfloor$ splits, where $A = \frac{b}{2}$. From theorems 3 and 4 the total number of message for the sequence is $C \leq 4m + \lfloor \frac{m}{A} \rfloor (\log n + 5) \leq m \left(4 + \frac{2(\log n+5)}{b}\right)$, hence the result holds.

Theorem 8. *Let* b *be the capacity of a bucket and* $\frac{b}{d}$ *be the merge threshold, for a fixed constant* $d > 2$. *Then a sequence of intermixed exact searches, insertions and deletions on ADST has an amortized cost of* $4 + \frac{2d}{d-2} \frac{O(\log n)}{b}$ *messages per operation.*

Proof. (Sketch). Let us consider a sequence of m intermixed exact searches, insertions, and deletions performed on ADST. Let $D = \frac{b}{2} - \frac{b}{d} = \frac{d-2}{2d}b$. For ease of presentation, we assume without loss of generality that $\lfloor \frac{m}{D} \rfloor = \frac{m}{D}$. From lemma 1, it is easy to verify that we can have at most $\lfloor \frac{m}{D} \rfloor$ operations among splits and merges. In the extended version we show that the worst-case for such a sequence is when, respect to a server, one split is followed by one merge, and vice-versa. From theorems 3, 4 and 6 the total number of message for the sequence is $C \leq 4m + \lfloor \frac{m}{D} \rfloor O(\log n) \leq m \left(4 + \frac{2d}{d-2} \frac{O(\log n)}{b}\right)$, hence the result holds.

We conclude showing a basic fact that holds in a realistic framework.

Assumption 9. *The number n of servers participating in an ADST is such that $\log n < kb$, where b is the capacity of a bucket and $k > 0$ is a constant.*

In fact, since in a realistic situation b is at least in the order of hundreds or thousands, then, assuming $k = 1$, it is true in practice that $n < 2^b$. Hence the assumption is realistically true. For the rest of the paper we therefore assume that: $\log n < b$.

Under the assumption 9 the results of theorems 7 and 8 show that in practice ADST has an amortized constant cost for any single-key operation.

4 Conclusions

We presented the ADST (Aggregation in Distributed Search Tree). This is the first order preserving SDDS, obtaining a constant single key query cost, like LH*, and at the same time an optimal cost for range queries. More precisely our structure features: (i) a cost of 4 messages for exact-search queries in the worst-case, (ii) a logarithmic cost for insert queries producing a split, (iii) an optimal cost for range searches, that is a range search can be answered with $O(k)$ messages, where k is the number of servers covering the query range, (iv) an amortized almost constant cost for any single-key query.

The internal load of a server is the typical one of order preserving SDDSs proposed till now. In particular, servers managing compound nodes may be also the ones managing buckets of the structure, like it is in DRT [8], or they may be dedicated servers, like in RP* [10]. The choice does not influence the correctness of ADST technique.

Moreover ADST is also able to manage deletions, and it is easily extendible to manage k-dimensional data, keeping the same results. Furthermore, ADST is an orthogonal technique with respect to techniques used to guarantee fault tolerance, in particular to the one in [13], that provides a high availability SDDS.

Hence our proposal is a theoretical achievement potentially attractive for distributed applications requiring high performances for single key and range queries, high availability and possibly the management of multi-dimensional data.

In [7] experimental comparisons exploring behavior of ADST in the average case and comparing it with existing structures are considered. The result is that

ADST is the best choice also in the average case. This makes ADST interesting, beyond the theoretical level, also from an application point of view.

Future work will also study the impact of using different aggregation policies. Any aggregation policies ensuring results of theorem 2 can be applied.

References

1. P. Bozanis, Y. Manolopoulos: DSL: Accomodating Skip Lists in the SDDS Model, *Workshop on Distributed Data and Structures (WDAS 2000)*, L'Aquila, June 2000.
2. Y. Breitbart, R. Vingralek: Addressing and Balancing Issues in Distributed B^+-Trees, *1st Workshop on Distributed Data and Structures (WDAS'98)*, 1998.
3. R.Devine: Design and implementation of DDH: a distributed dynamic hashing algorithm, *4th Int. Conf. on Foundations of Data Organization and Algorithms (FODO)*, Chicago, 1993.
4. A.Di Pasquale, E. Nardelli: Fully Dynamic Balanced and Distributed Search Trees with Logarithmic Costs, *Workshop on Distributed Data and Structures (WDAS'99)*, Princeton, NJ, May 1999.
5. A.Di Pasquale, E. Nardelli: Distributed searching of k-dimensional data with almost constant costs, *ADBIS 2000*, Prague, September 2000.
6. A.Di Pasquale, E. Nardelli: ADST: Aggregation in Distributed Search Trees, Technical Report 1/2001, *University of L'Aquila*, February 2001.
7. A.Di Pasquale, E. Nardelli: A Very Efficient Order Preserving Scalable Distributed Data Structure, accepted for pubblication at *DEXA 2001 Conference*.
8. B. Kröll, P. Widmayer: Distributing a search tree among a growing number of processor, in *ACM SIGMOD Int. Conf. on Management of Data*, pp 265-276 Minneapolis, MN, 1994.
9. W. Litwin, M.A. Neimat, D.A. Schneider: LH* - Linear hashing for distributed files, *ACM SIGMOD Int. Conf. on Management of Data*, Washington, D. C., 1993.
10. W. Litwin, M.A. Neimat, D.A. Schneider: RP* - A family of order-preserving scalable distributed data structure, in *20th Conf. on Very Large Data Bases*, Santiago, Chile, 1994.
11. W. Litwin, M.A. Neimat, D.A. Schneider: k-RP$^*_s$ - A High Performance Multi-Attribute Scalable Distributed Data Structure, in *4th International Conference on Parallel and Distributed Information System*, December 1996.
12. W. Litwin, M.A. Neimat, D.A. Schneider: LH* - A Scalable Distributed Data Structure, *ACM Trans. on Database Systems*, 21(4), 1996.
13. W. Litwin, T.J.E. Schwarz, S.J.: LH*$_{RS}$: a High-availability Scalable Distributed Data Structure using Reed Solomon Codes, *ACM SIGMOD Int. Conf. on Management of Data*, 1999.
14. E. Nardelli, F.Barillari, M. Pepe: Distributed Searching of Multi-Dimensional Data: a Performance Evaluation Study, *Journal of Parallel and Distributed Computation (JPDC)*, 49, 1998.
15. R.Vingralek, Y.Breitbart, G.Weikum: Distributed file organization with scalable cost/performance, *ACM SIGMOD Int. Conf. on Management of Data*, Minneapolis, MN, 1994.
16. R.Vingralek, Y.Breitbart, G.Weikum: SNOWBALL: Scalable Storage on Networks of Workstations with Balanced Load, *Distr. and Par. Databases*, 6, 2, 1998.

Approximative Learning of Regular Languages

Henning Fernau

Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
fernau@informatik.uni-tuebingen.de

Abstract. We show how appropriately chosen functions f which we call *distinguishing* can be used to make deterministic finite automata backward deterministic. These ideas have been exploited to design regular language classes called f-distinguishable which are identifiable in the limit from positive samples. Special cases of this approach are the k-reversible and terminal distinguishable languages as discussed in [1,3,5,15,16]. Here, we give new characterizations of these language classes. Moreover, we show that all regular languages can be approximated in the setting introduced by Kobayashi and Yokomori [12,13]. Finally, we prove that the class of all function-distinguishable languages is equal to the class of regular languages.

1 Introduction

Identification in the limit from positive samples, also known as *exact learning from text* as proposed by Gold [10], is one of the oldest yet most important models of grammatical inference. Since not all regular languages can be learned exactly from text, the characterization of *identifiable* subclasses of regular languages is a useful line of research, because the regular languages are a very basic language family.

In [6], we introduced the so-called function-distinguishable languages as a rich source of examples of identifiable language families. Among the language families which turn out to be special cases of our approach are the k-reversible languages [1] and the terminal-distinguishable languages [15,16], which belong, according to Gregor [11], to the most popular identifiable regular language classes. Moreover, we have shown [6] how to transfer the ideas underlying the well-known identifiable language classes of k-testable languages, k-piecewise testable languages and threshold testable languages to our setting. In a nutshell, an identification algorithm for f-distinguishable languages assigns to every finite set of samples $I_+ \subseteq T^*$ the smallest[1] f-distinguishable language containing I_+ by subsequently merging states which cause conflicts to the definition of f-distinguishable automata, starting with the simple prefix tree automaton accepting I_+.

[1] This is well-defined, since each class of f-distinguishable languages is closed under intersection, see Theorem 2.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 223–232, 2001.

In this paper, we firstly give a further useful characterization of function-distinguishable languages which has been also employed in other papers [4,7,8]. This also allows us to define a possible merging-state inference strategy in a concise manner. Then, we focus on questions of approximability of regular languages by function-distinguishable languages in the setting introduced by Kobayashi and Yokomori [13].

The paper is organized as follows: In Section 2, we provide the necessary background from formal language theory and in sections 3 and 4, we introduce the central concepts of the paper, namely the so-called distinguishing functions and the function distinguishable automata and languages. In Section 5, we discuss an alternative definition of the function canonical automata which we used as compact presentation in other papers. In Section 6, we show how to approximate arbitrary regular languages by using function-distinguishable languages, based on the notion of upper-best approximation in the limit introduced by Kobayashi and Yokomori in [12,13]. Section 7 concludes the paper, indicating practical applications of our method and extensions to non-regular language families.

An extended version of this paper is available as Technical Report WSI 2001–2.

2 General Definitions

Σ^* is the set of words over the alphabet Σ. Σ^k ($\Sigma^{<k}$) collects the words whose lengths are equal to (less than) k. λ denotes the empty word. $\mathrm{Pref}(L)$ is the set of prefixes of L and $u^{-1}L = \{ v \in \Sigma^* | uv \in L \}$ is the quotient of $L \subseteq \Sigma^*$ by u.

We assume that the reader knows that regular languages can be characterized by (deterministic) finite automata $A = (Q, T, \delta, q_0, Q_F)$, where Q is the state set, $\delta \subseteq Q \times T \times Q$ is the transition relation, $q_0 \in Q$ is the initial state and $Q_F \subseteq Q$ is the set of final states. As usual, δ^* denotes the extension of the transition relation to arbitrarily long input words. The language defined by an automaton A is written $L(A)$. An automaton is called *stripped* iff all states are accessible from the initial state and all states lead to some final state. Observe that the transition function of a stripped deterministic finite automaton is not total in general.

We denote the minimal deterministic automaton of the regular language L by $A(L)$. Recall that $A(L) = (Q, T, \delta, q_0, Q_F)$ can be described as follows: $Q = \{u^{-1}L | u \in \mathrm{Pref}(L)\}$, $q_0 = \lambda^{-1}L = L$; $Q_F = \{u^{-1}L | u \in L\}$; and $\delta(u^{-1}L, a) = (ua)^{-1}L$ with $u, ua \in \mathrm{Pref}(L)$, $a \in T$. According to our definition, any minimal deterministic automaton is stripped.

Furthermore, we need two automata constructions in the following:
The *product automaton* $A = A_1 \times A_2$ of two automata $A_i = (Q_i, T, \delta_i, q_{0,i}, Q_{F,i})$ for $i = 1, 2$ is defined as $A = (Q, T, \delta, q_0, Q_F)$ with $Q = Q_1 \times Q_2$, $q_0 = (q_{0,1}, q_{0,2})$, $Q_F = Q_{F,1} \times Q_{F,2}$, $((q_1, q_2), a, (q_1', q_2')) \in \delta$ iff $(q_1, a, q_1') \in \delta_1$ and $(q_2, a, q_2') \in \delta_2$.

A *partition* of a set S is a collection of pairwise disjoint nonempty subsets of S whose union is S. If π is a partition of S, then, for any element $s \in S$, there is a unique element of π containing s, which we denote $B(s, \pi)$ and call

the *block* of π containing s. A partition π is said to *refine* another partition π' iff every block of π' is a union of blocks of π. If π is any partition of the state set Q of the automaton $A = (Q, T, \delta, q_0, Q_F)$, then the *quotient automaton* $\pi^{-1}A = (\pi^{-1}Q, T, \delta', B(q_0, \pi), \pi^{-1}Q_F)$ is given by $\pi^{-1}\hat{Q} = \{\, B(q, \pi) \mid q \in \hat{Q} \,\}$ (for $\hat{Q} \subseteq Q$) and $(B_1, a, B_2) \in \delta'$ iff $\exists q_1 \in B_1 \exists q_2 \in B_2 : (q_1, a, q_2) \in \delta$.

3 Distinguishing Functions

In order to avoid cumbersome case discussions, let us fix now T as the input alphabet of the finite automata we are going to discuss.

Definition 1. *Let F be some finite set. A mapping $f : T^* \to F$ is called a* distinguishing function *if $f(w) = f(z)$ implies $f(wu) = f(zu)$ for all $u, w, z \in T^*$.*

In the literature, we can find the terminal function [16]

$$\mathrm{Ter}(x) = \{\, a \in T \mid \exists u, v \in T^* : uav = x \,\}$$

and, more generally, the k-terminal function [5]

$$\begin{aligned} \mathrm{Ter}_k(x) &= (\pi_k(x), \mu_k(x), \sigma_k(x)), \quad \text{where} \\ \mu_k(x) &= \{\, a \in T^{k+1} \mid \exists u, v \in T^* : uav = x \,\} \end{aligned}$$

and $\pi_k(x)$ [$\sigma_k(x)$] is the prefix [suffix] of length k of x if $x \notin T^{<k}$, and $\pi_k(x) = \sigma_k(x) = x$ if $x \in T^{<k}$. The example $f(x) = \sigma_k(x)$ leads to the k-reversible languages, confer [1,5]. In particular, the trivial distinguishing function, whose range is a singleton set, characterizes the 0-reversible languages. Other examples of distinguishing functions in the context of even linear languages can be found in [4,15].

Observe that every regular language R induces, via its Nerode equivalence classes, a distinguishing function f_R, where $f_R(w)$ maps w to the equivalence class containing w. Especially, T^* leads to a trivial distinguishing function $f_{T^*} : T^* \to \{q\}$, and the class of f_{T^*}-distinguishable languages coincides with the class of 0-reversible languages [1] over the alphabet T.

In some sense, these are the only distinguishing functions, since one can associate to every distinguishing function f a finite automaton $A_f = (F, T, \delta_f, f(\lambda), F)$ by setting $\delta_f(q, a) = f(wa)$, where $w \in f^{-1}(q)$ can be chosen arbitrarily, since f is a distinguishing function.

4 Function Distinguishable Languages

Here, we will formally introduce function distinguishable languages and discuss some formal language properties.

Definition 2. *Let $A = (Q, T, \delta, q_0, Q_F)$ be a finite automaton. Let $f : T^* \to F$ be a distinguishing function. A is called* f-distinguishable *if:*

1. *A is deterministic.*

2. *For all states $q \in Q$ and all $x, y \in T^*$ with $\delta^*(q_0, x) = \delta^*(q_0, y) = q$, we have $f(x) = f(y)$.*
   (In other words, for $q \in Q$, $f(q) := f(x)$ for some x with $\delta^*(q_0, x) = q$ is well-defined.)
3. *For all $q_1, q_2 \in Q$, $q_1 \neq q_2$, with either (a) $q_1, q_2 \in Q_F$ or (b) there exist $q_3 \in Q$ and $a \in T$ with $\delta(q_1, a) = \delta(q_2, a) = q_3$, we have $f(q_1) \neq f(q_2)$.*

A language is called *f-distinguishable* iff it can be accepted by an f-distinguishable automaton. The family of f-distinguishable languages is denoted by f-DL.

We need a suitable notion of a canonical automaton in the following.

Definition 3. *Let $f : T^* \to F$ be a distinguishing function and let $L \subseteq T^*$ be a regular set. Let $A(L, f)$ be the stripped subautomaton of the product automaton $A(L) \times A_f$, i.e., delete all states that are not accessible from the initial state or do not lead into a final state of $A(L) \times A_f$. $A(L, f)$ is called* f-canonical automaton *of L.*

Observe that the class f-DL formally fixes the alphabet of the languages by the range of f. As we have already seen by the examples for distinguishing functions listed above, f can oftenly defined for *all* alphabets. Taking this generic point of view, for example, Ter-DL is just the class of (reversals of) terminal distinguishable languages [4,16], where the alphabet is left unspecified.

For example, for each distinguishing function f, the associated automaton A_f is f-distinguishable. This simple observation leads us to:

Theorem 1. *A language is function-distinguishable iff it is regular.*

Proof. Let L be a regular language. Consider the canonical automaton A_L for L. It is quite easy to see that A_L is f_L-distinguishable. □

In other words, $\{\, f\text{-DL} \mid f \text{ is a distinguishing function} \,\}$ gives a finer classification of all regular languages. This finer classification is necessary, since it is well known that the class of all regular languages is not identifiable in the limit from positive data [10].

The following theorem generalizes the corresponding assertion for k-reversible languages as stated by Angluin [1].

Theorem 2. *For each distinguishing function f, f-DL is closed under intersection.*

Proof. The standard product automaton construction is applicable. □

To the contrary, f-DL is *not* closed under union nor complement in general, see [1]. According to Pin [14], the union closure of the 0-reversible languages is characterized by another class of regular languages which he calls reversible. He calls a language L *reversible* iff there is a finite automaton A accepting L such that A is deterministic and codeterministic but has possibly several initial and several accepting states. Sometimes, such automata are also called injective automata or permutation automata.

5 An Alternative Presentation

In [6], we developed a generic merging state algorithm for f-DL which paralleled the approach of Angluin for 0-reversible languages. More precisely, the algorithm, when given an input sample I_+, starts with the prefix tree acceptor $PTA(I_+)$ (as defined below). If $A_f(I_+)$ ($L_f(I_+)$, resp.) denotes the output automaton (output language, resp.) of the merging state inference algorithm when given I_+, then (disregarding automaton ismorphism) $A(L_f(I_+), f) = A_f(I_+)$, see [6]. In their works, Radhakrishnan and Nagaraja [16] do not start with the PTA of the given input data set I_+ but rather with a so-called "skeletal grammar" for the given input data set I_+, which corresponds to the "maximal canonical automaton" $MCA(I_+)$ in the framework of Dupont and Miclet [2]. Here, we describe a related algorithm for learning f-DL-languages. This way, we also yield an alternative characterization of f-DL.

Consider an input sample set $I_+ = \{w_1, \ldots, w_M\} \subseteq T^+$.[2] Let $w_i = a_{i1} \ldots a_{in_i}$, where $a_{ij} \in T$, $1 \leq i \leq M$, $1 \leq j \leq n_i$. The *skeletal automaton* for the sample set is defined as

$$\begin{aligned} A_S(I_+) &= (Q_S, T, \delta_S, Q_0, Q_f), \quad \text{where} \\ Q_S &= \{\, q_{ij} \mid 1 \leq i \leq M, 1 \leq j \leq n_i + 1 \,\}, \\ \delta_S &= \{\, (q_{ij}, a_{i,j}, q_{i,j+1}) \mid 1 \leq i \leq M, 1 \leq j \leq n_i \,\}, \\ Q_0 &= \{\, q_{i1} \mid 1 \leq i \leq M \,\} \quad \text{and} \\ Q_f &= \{\, q_{i,n_i+1} \mid 1 \leq i \leq M \,\}. \end{aligned}$$

Observe that we allow a *set* of initial states. The *frontier string* of q_{ij} is defined by $\mathrm{FS}(q_{ij}) = a_{ij} \ldots a_{in_i}$. The *head string* of q_{ij} is defined by the equation $\mathrm{HS}(q_{ij})\mathrm{FS}(q_{ij}) = w_i$, i.e., $\mathrm{HS}(q_{ij}) = a_{i1} \ldots a_{i,j-1}$. In other words, $\mathrm{HS}(q_{ij})$ is the unique string leading from an initial state into q_{ij}, and $\mathrm{FS}(q_{ij})$ is the unique string leading from q_{ij} into a final state.[3] Therefore, the skeletal automaton of a sample set simply spells all words of the sample set in a trivial fashion. Two things can be easily observed.

1. The state partition π of Q_S induced by $q \equiv q'$ iff $\mathrm{HS}(q) = \mathrm{HS}(q')$ yields the prefix tree acceptor, i.e., $PTA(I_+) = \pi^{-1}A_S(I_+)$.
2. Since there is only one word leading to any q, namely $\mathrm{HS}(q)$, $f(q) = f(\mathrm{HS}(q))$ can be uniquely defined.

Now, for $q_{ij}, q_{k\ell} \in Q_S$, define $q_{ij} \rightleftharpoons_f q_{k\ell}$ iff (1) $\mathrm{HS}(q_{ij}) = \mathrm{HS}(q_{k\ell})$ or (2) $\mathrm{FS}(q_{ij}) = \mathrm{FS}(q_{k\ell})$, as well as $f(q_{ij}) = f(q_{k\ell})$.

The following assertion is easily verified:

Lemma 1. *For each distinguishing function f and each sample set I_+, $\rightleftharpoons_f$ is a reflexive symmetric relation on the set Q_S of states of $A_S(I_+)$.*

[2] The inclusion of the empty word would introduce some unnecessary technalities.

[3] In order to overcome unnecessary technical complications, we underline here that we are dealing with a sample *set*, i.e., we do not consider repitions of sample words which are allowed in Gold's model in general.

In general, $\rightleftharpoons_f$ is not an equivalence relation on the state set of A_S, as the following example shows:

Example 1. Consider the trivial distinguishing function σ_0 and $I_+ = \{a, aa\}$. The skeletal automaton has the following state transitions: (q_{11}, a, q_{12}), (q_{21}, a, q_{22}) and (q_{22}, a, q_{23}). Since $\mathrm{HS}(q_{11}) = \mathrm{HS}(q_{21}) = \lambda$ and $\mathrm{HS}(q_{12}) = \mathrm{HS}(q_{22}) = a$, as well as $\mathrm{FS}(q_{12}) = \mathrm{FS}(q_{23}) = \lambda$, $\mathrm{FS}(q_{11}) = \mathrm{FS}(q_{22}) = a$ and $\mathrm{FS}(q_{21}) = aa$, all states in Q_S are σ_0-equivalent, but $q_{11} \not\rightleftharpoons_{\sigma_0} q_{12}$.

Therefore, we define $\equiv_f := (\rightleftharpoons_f)^+$, denoting in this way the transitive closure of the original relation. The following lemmma is again an easy exercise left to the reader.

Lemma 2. *For each distinguishing function f and each sample set I_+, $\equiv_f$ is an equivalence relation on the state set of $A_S(I_+)$.*

We consider now the automaton $\pi_f^{-1} A_S(I_+)$, where π_f is the partition induced by the equivalence relation $\equiv_f$. We like to show that $A_f(I_+) = \pi_f^{-1} A_S(I_+)$. As a preparatory stage, we prove:

Lemma 3. *For each distinguishing function f and each sample set I_+, $\pi_f^{-1} A_S(I_+)$ is an f-distinguishable automaton.*

Proof. We have to verify the three conditions posed upon f-distinguishable automata for $\pi_f^{-1} A_S(I_+)$. Let δ denote the transition relation of $\pi_f^{-1} A_S(I_+)$ and $\bar{q}_0$ its initial state. (We use barred state notations for states of $\pi_f^{-1} A_S(I_+)$ and non-barred notations for states of $A_S(I_+)$.)
ad 1.: Consider an input word w with $q_1, q_2 \in \delta^*(\bar{q}_0, w)$. Then, there are some $q_{ij} \in \bar{q}_1$ and $q_{k\ell} \in \bar{q}_2$ (recall that $\bar{q}_1, \bar{q}_2$ are both sets of states of $A_S(I_+)$) with $\mathrm{HS}(q_{ij}) = w$ and $\mathrm{HS}(q_{k\ell}) = w$. Hence, $q_{ij} \rightleftharpoons_f q_{k\ell}$, which means that $\bar{q}_1 = \bar{q}_2$, since $\bar{q}_1$ and $\bar{q}_2$ are equivalence classes of states of $A_S(I_+)$.
ad 2.: Observe that $f(q)$ is well-defined for every state q of $A_S(I_+)$. It is easy to check that if $q \rightleftharpoons_f q'$, then $f(q) = f(q')$. Since $q, q' \in \bar{q}$ iff $q \equiv_f q'$ iff $q \rightleftharpoons_f^+ q'$, $f(q) = f(q')$ immediately follows by the transitivity of equality.
ad 3.: It can be shown similar to point 1 (formally by induction). □

Theorem 3. *For each distinguishing function f and each sample set I_+, we have, up to isomorphism, $A_f(I_+) = \pi_f^{-1} A_S(I_+)$.*

Proof. According to [2], we can consider $\pi_f^{-1} A_S(I_+)$ as being obtained by a sequence of merging state steps, merging only two states at a time. Without loss of generality, such a sequence of mergings might start with "repairing" violations of the determinism requirement, so that we obtain $PTA(I_+)$ as an intermediate automaton. Similar to the reasoning in the previous lemma, the reader may verify that each of these merging steps can be justified also by the existence of conflicts in the merged states according to inference algorithm sketched in the introduction. Since we have shown the correctness of that inference algorithm in [6], the assertion of this theorem follows, as well. □

This argument justifies the presentation of certain subcases of function distinguishable languages as done in [4,7].

6 Approximation

Kobayashi and Yokomori introduced in [12,13] the notion of upper-best approximation in the limit of a target language with respect to the hypothesis space. They showed that regular languages can be upper-best approximated by k-reversible languages for any fixed k. Here, we shall prove that similar results are true for any class f-DL. In particular, this implies that, given any enumeration of an arbitrary regular language to some identification algorithm for f-DL, this algorithm will converge, yielding some well-defined result. Especially, the terminal distinguishable languages can be used to approximate all regular languages in a precise sense. This is interesting, since already Radhakrishnan and Nagaraja observed in [16] on an empirical basis that their algorithm converges for regular languages, but not for context-free languages. The approximation notion developed by Kobayashi and Yokomori gives a mathematical explanation of this empirical observation.

Firstly, we give the necessary definitions due to Kobayashi and Yokomori.

Let $\mathcal{L}$ be a language class and L be a language possibly outside $\mathcal{L}$. An *upper-best approximation $\bar{\mathcal{L}}L$ of L with respect to $\mathcal{L}$* is defined to be a language L_* containing L such that for any $L' \in \mathcal{L}$ with $L \subseteq L'$, $L_* \subseteq L'$ holds. If such an L_* does not exist, $\bar{\mathcal{L}}L$ is undefined.

Remark 1. If $\mathcal{L}$ is closed under intersection, then L_* is uniquely defined.

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two language classes. We say that $\mathcal{L}_1$ has the *upper-best approximation property (u.b.a.p.) with respect to* $\mathcal{L}_2$ iff, for every $L \in \mathcal{L}_2$, $\bar{\mathcal{L}}_1 L$ is defined.

Consider an inference machine I to which as input an arbitrary language $L \in \mathcal{L}$ may be enumerated (possibly with repetitions) in an arbitrary order, i.e., I receives an infinite input stream of words $E(1)$, $E(2)$, ..., where $E : \mathbb{N} \to L$ is an enumeration of L. We say that I *identifies an upper-best approximation of L in the limit (from positive data) by $\mathcal{L}$* if I reacts on an enumeration of L with an output device stream $D_i \in \mathcal{D}$ such that there is an $N(E)$ so that, for all $n \geq N(E)$, we have $D_n = D_{N(E)}$ and, moreover, the language defined by $D_{N(E)}$ equals $\bar{\mathcal{L}}L$. A language class $\mathcal{L}_1$ is called *upper-best approximately identifiable in the limit (from positive data) by $\mathcal{L}_2$* iff there exists an inference machine I which identifies an upper-best approximation of each $L \in \mathcal{L}_1$ in the limit (from positive data) by $\mathcal{L}_2$. Observe that this notion of identifiability coincides with Gold's classical notion of learning in the limit in the case when $\mathcal{L}_1 = \mathcal{L}_2$.

Consider a language class $\mathcal{L}$ and a language L from it. A finite subset $F \subseteq L$ is called a *characteristic sample* of L with respect to $\mathcal{L}$ iff, for any $L' \in \mathcal{L}$, $F \subseteq L'$ implies that $L \subseteq L'$.

Now, fix some distinguishing function f. We call a language $L \subseteq T^*$ *pseudo-f-distinguishable* iff, for all $u_1, u_2, v \in T^*$ with $f(u_1) = f(u_2)$, we have $u_1^{-1}L = u_2^{-1}L$ whenever $\{u_1 v, u_2 v\} \subseteq L$. By the characterization theorem derived in [6], $L \in f$-DL iff L is pseudo-f-distinguishable and regular.

Immediately from the definition, we may conclude:

Proposition 1. *Let $L_1 \subseteq L_2 \subseteq \ldots$ be any ascending sequence of pseudo-f-distinguishable languages. Then, $\bigcup_{i \geq 1} L_i$ is pseudo-f-distinguishable.* □

For brevity, we write $u_1 \equiv_{L,f} u_2$ iff $u_1^{-1}L = u_2^{-1}L$ and $f(u_1) = f(u_2)$.

Remark 2. If $L \subseteq T^*$ is a regular language and if $f : T^* \to F$ is some distinguishing function, then the number of equivalence classes of $\equiv_{L,f}$ equals the number of states of A_L (plus one) times $|F|$, and this is just the number of states of $A(L, f)$ (plus $|F|$).

Let $L \subseteq T^*$ be some language. For any integer i, we will define $R_f(i, L)$ as follows:

1. $R_f(0, L) = L$ and
2. $R_f(i, L) = R_f(i-1, L) \cup \{\, u_2 w \mid u_1 v, u_2 v, u_1 w \in R_f(i-1, L) \wedge f(u_1) = f(u_2) \,\}$ for $i \geq 1$.

Furthermore, set $R_f(L) = \bigcup_{i \geq 0} R_f(i, L)$.

Observe that, by definition, a language is pseudo-k-reversible [13] iff it is pseudo-σ_k-distinguishable. Moreover, the operator R_k introduced in [13] is written as R_{σ_k} in our notation.

Since R_f turns out to be a hull operator, the following statement is obvious.

Proposition 2. *For any language L and any distinguishing function f, $R_f(L)$ is the smallest pseudo-f-distinguishable language containing L.* □

Lemma 4. *Let $L \subseteq T^*$ be any language. If u_1 and u_2 are prefixes of L, then $u_1 \equiv_{L,f} u_2$ implies that $u_1^{-1}R_f(L) = u_2^{-1}R_f(L)$.*

Proof. Let u_1 and u_2 be prefixes of L with $u_1 \equiv_{L,f} u_2$. By definition of $\equiv_{L,f}$, $u_1^{-1}L = u_2^{-1}L \neq \emptyset$. Hence, there is a string v so that $\{u_1 v, u_2 v\} \subseteq L \subseteq R_f(L)$. Furthermore, by definition of $\equiv_{L,f}$, $f(u_1) = f(u_2)$. Since $R_f(L)$ is pseudo-f-distinguishable due to Proposition 2, $u_1^{-1}R_f(L) = u_2^{-1}R_f(L)$. □

Lemma 5. *Let $L \subseteq T^*$ be any language and let f be any distinguishing function. Then, for any prefix w_1 of $R_f(L)$, there exists a prefix w_2 of L with $w_1^{-1}R_f(L) = w_2^{-1}R_f(L)$.*

Proof. Since w_1 is a prefix of $R_f(L)$ iff w_1 is a prefix of $R_f(i, L)$ for some $i \geq 0$, it suffices to show the following claim by induction:

> Let $i \geq 0$. Then, for any prefix w_1 of $R_f(i, L)$, there exists a prefix w_2 of L with $w_1^{-1}R_f(L) = w_2^{-1}R_f(L)$.

Trivially, the claim is true when $i = 0$, since $R_f(0, L) = L$.

As induction hypothesis, assume that the claim is shown for $i = \ell$. Hence, we have to consider some $w_1 \in \mathrm{Pref}(R_f(\ell + 1, L)) \setminus \mathrm{Pref}(R_f(\ell, L))$ in the induction step. Consider some $w_1 z \in R_f(\ell + 1, L) \setminus R_f(\ell, L)$. This means that there are strings $u_1, v, w \in T^*$ with $\{u_1 v, u_2 v, u_1 w\} \subseteq R_f(\ell, L)$, $f(u_1) = f(u_2)$ and $u_2 w = w_1 z$. If $|u_2| \geq |w_1|$, w_1 is a prefix of $u_2 w \in R_f(\ell, L)$ in contrast to our assumption. Therefore, we have $w_1 = u_2 v'$ for some $v' \in T^+$. Since $R_f(L)$ is pseudeo-f-distinguishable and $\{u_1 v, u_2 v\} \subseteq R_f(L)$ as well as $f(u_1) = f(u_2)$, $u_1^{-1}R_f(L) = u_2^{-1}R_f(L)$, which yields $w_1^{-1}R_f(L) = (u_2 v')^{-1}R_f(L) = (u_1 v')^{-1}R_f(L)$. Since v' is a prefix of w, $u_1 v'$ is a prefix of $u_1 w \in R_f(\ell, L)$. By induction hypothesis, there is a prefix w_2 of L such that $w_2^{-1}R_f(L) = (u_1 v')^{-1}R_f(L) = w_1^{-1}R_f(L)$. □

By a reasoning completely analogous to [13], we may conclude:

Theorem 4. *For any distinguishing function f, the class f-DL has the u.b.a.p. with respect to the class of regular languages.* □

Observe that the number of states of $A_{R_f(L)}$ is closely related to the number of states of $A(L, f)$, see Remark 2.

Theorem 5. *For any distinguishing function f, the class of regular languages is upper-best approximately identifiable in the limit from positive data by f-DL.* □

7 Discussion

We have proposed a large collection of families of languages, each of which is identifiable in the limit from positive samples, hence extending previous works. We feel that deterministic methods yielding characterizable regular subclasses (such as the ones proposed in this paper) are quite important for practical applications, since they could be understood more precisely than mere heuristics, so that one can prove certain properties about the algorithms. Moreover, the approach of this paper allows one to make the bias (which each regular language identification algorithm necessarily has) explicit and transparent to the user: The bias consists in (1) the restriction to regular languages and (2) the choice of a particular distinguishing function f. Detailed comments in this direction can be found in [8].

We will provide a publicly accessible prototype learning algorithm for (each of the families) f-DL in the near future. A user can then firstly look for an appropriate f by making learning experiments with typical languages he expects to be representative for the languages in his particular application. If there are only few "typical languages" $L_1, \ldots, L_r$ in the beginning, one could also start with $f_{L_1} \times \cdots \times f_{L_r}$, where $f \times g$ is defined as $(f \times g)(x) = (f(x), g(x))$, see the proof of Theorem 1. After this "bias training phase", the user may then use the such-chosen learning algorithm (or better, an improved implementation for the specific choice of f) for his actual application.

Even if the particular class f-DL chosen by the user does not completely comprise all languages the identification machine IM will be confronted with, Theorem 5 suggests that, in the case that a regular language which does not lie in f-DL is enumerated to IM, some reasonable outcome will be produced in a reasonable time.

If the application suggests that the languages which are to be inferred are non-regular, methods such as those suggested in [15] can be transferred. This is done most easily by using the concept of *control languages* as undertaken in [3,4] or [17, Section 4] or by using the related concept of *permutations*, see [9].

Acknowledgment

We gratefully acknowledge discussions with S. Kobayashi.

References

1. D. Angluin. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29(3):741–765, 1982.
2. P. Dupont and L. Miclet. Inférence grammaticale régulière: fondements théoriques et principaux algorithmes. Technical Report RR-3449, INRIA, 1998.
3. H. Fernau. Learning of terminal distinguishable languages. Technical Report WSI–99–23, Universität Tübingen (Germany), Wilhelm-Schickard-Institut für Informatik, 1999. Short version published in the proceedings of AMAI 2000, see `http://rutcor.rutgers.edu/~amai/AcceptedCont.htm`.
4. H. Fernau. Identifying terminal distinguishable languages. Submitted revised version of [3].
5. H. Fernau. k-gram extensions of terminal distinguishable languages. In *Proc. 15th International Conference on Pattern Recognition*. 2nd Volume, pp. 125–128, IEEE Press, 2000.
6. H. Fernau. Identification of function distinguishable languages. In *Proc. 11th International Conference Algorithmic Learning Theory (ALT)*, volume 1968 of *LNCS/LNAI*, pages 116–130. Springer, 2000.
7. H. Fernau. Parallel communicating grammar systems with terminal transmission. *Acta Informatica*, 37:511–540, 2001.
8. H. Fernau. Learning XML Grammars. In *Proc. 2nd Machine Learning and Data Mining in Pattern Recognition MLDM'01*, volume 2123 of *LNCS/LNAI*, pages 73–87. Springer, 2001.
9. H. Fernau and J. M. Sempere. Permutations and control sets for learning non-regular language families. In *Proc. 5th International Colloquium on Grammatical Inference (ICGI): Algorithms and Applications*, volume 1891 of *LNCS/LNAI*, pages 75–88. Springer, 2000.
10. E. M. Gold. Language identification in the limit. *Information and Control (now Information and Computation)*, 10:447–474, 1967.
11. J. Gregor. Data-driven inductive inference of finite-state automata. *International Journal of Pattern Recognition and Artificial Intelligence*, 8(1):305–322, 1994.
12. S. Kobayashi and T. Yokomori. On approximately identifying concept classes in the limit. In *Proc. 6th International Conference Algorithmic Learning Theory (ALT)*, volume 997 of *LNCS/LNAI*, pages 298–312. Springer, 1995.
13. S. Kobayashi and T. Yokomori. Learning approximately regular languages with reversible languages. *Theoretical Computer Science*, 174:251–257, 1997.
14. J.E. Pin. On the languages accepted by finite reversible automata. In *14th ICALP'87*, volume 267 of *LNCS*, pages 237–249, 1987.
15. V. Radhakrishnan. *Grammatical Inference from Positive Data: An Effective Integrated Approach*. PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay (India), 1987.
16. V. Radhakrishnan and G. Nagaraja. Inference of regular grammars via skeletons. *IEEE Transactions on Systems, Man and Cybernetics*, 17(6):982–992, 1987.
17. Y. Takada. A hierarchy of language families learnable by regular language learning. *Information and Computation*, 123:138–145, 1995.

Quantum Finite State Transducers

Rūsiņš Freivalds[1] and Andreas Winter[2]

[1] Institute of Mathematics and Computer Science, University of Latvia,
Raiņa bulvāris 29, LV–1459, Riga, Latvia
Rusins.Freivalds@mii.lu.lv
[2] Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road, Bristol BS8 1UB, United Kingdom
winter@cs.bris.ac.uk

Abstract. We introduce *quantum finite state transducers (qfst)*, and study the class of relations which they compute. It turns out that they share many features with probabilistic finite state transducers, especially regarding undecidability of emptiness (at least for low probability of success). However, like their 'little brothers', the quantum finite automata, the power of qfst is incomparable to that of their probabilistic counterpart. This we show by discussing a number of characteristic examples.

1 Introduction and Definitions

The issue of this work is to introduce and to study the computational model of quantum finite state transducers. These can be understood as finite automata with the addition of an output tape which compute a relation between strings, instead of a decision (which we read as a binary valued function). After the necessary definitions, the relation to quantum finite automata is clarified (section 2), then decidability questions are addressed (section 3): it is shown that emptiness of the computed relation is undecidable both for quantum and probabilistic transducers. However, the membership problem for a specific output is decidable. Next, the relation between deterministic and probabilistic transducers is explored (section 4), and in section 5 quantum and probabilistic transducers are compared.

We feel our extension of quantum automata studies to this new model justified by the following quote from D. Scott [10]:

> *'The author (along with many other people) has come recently to the conclusion that the functions computed by the various machines are more important – or at least more basic – than the sets accepted by these devices. (...) In fact by putting the functions first, the relationship between various classes of sets becomes much clearer'.*

We start be reviewing the concept of probabilistic finite state transducer. For a finite set X we denote by X^* the set of all finite strings formed from X, the empty string is denoted ϵ.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 233–242, 2001.

Definition 1 *A* probabilistic finite state transducer (pfst) *is a tuple*

$$T = (Q, \Sigma_1, \Sigma_2, V, f, q_0, Q_{\mathrm{acc}}, Q_{\mathrm{rej}}),$$

where Q is a finite set of states, Σ_1, Σ_2 is the input/ output alphabet, $q_0 \in Q$ is the initial state, and $Q_{\mathrm{acc}}, Q_{\mathrm{rej}} \subset Q$ are (disjoint) sets of accepting and rejecting states, respectively. (The other states, forming set Q_{non}, are called non–halting). The transition function $V : \Sigma_1 \times Q \to Q$ is such that for all $a \in \Sigma_1$ the matrix $(V_a)_{qp}$ is stochastic, and $f_a : Q \to \Sigma_2^*$ is the output function. If all matrix entries are either 0 or 1 the machine is called a deterministic finite state transducer (dfst).

The meaning of this definition is that, being in state q, and reading input symbol a, the transducer prints $f_a(q)$ on the output tape, and changes to state p with probability $(V_a)_{qp}$, moving input and output head to the right. After each such step, if the machine is found in a halting state, the computation stops, accepting or rejecting the input, respectively.

To capture this formally, we introduce the *total state* of the machine, which is an element

$$(P_{\mathrm{NON}}, P_{\mathrm{ACC}}, p_{\mathrm{rej}}) \in \ell^1(Q \times \Sigma_2^*) \oplus \ell^1(\Sigma_2^*) \oplus \ell^1(\{\mathrm{REJ}\}),$$

with the natural norm

$$\|(P_{\mathrm{NON}}, P_{\mathrm{ACC}}, p_{\mathrm{rej}})\| = \|P_{\mathrm{NON}}\|_1 + \|P_{\mathrm{ACC}}\|_1 + |p_{\mathrm{rej}}|.$$

At the beginning, the total state is $((q_0, \epsilon), \mathbf{0}, 0)$ (where we identify an element of $Q \times \Sigma_2^*$ with its characteristic function). The computation is represented by the (linear extensions of the) transformations

$$T_a : ((q, w), P_{\mathrm{ACC}}, p_{\mathrm{rej}}) \mapsto \left(\left(\sum_{p \in Q_{\mathrm{non}}} (V_a)_{qp} p, w f_a(q) \right), P'_{\mathrm{ACC}}, p'_{\mathrm{rej}} \right),$$

of the total state, for $a \in \Sigma_1$, with

$$P'_{\mathrm{ACC}}(x) = \begin{cases} P_{\mathrm{ACC}}(x) + \sum_{p \in Q_{\mathrm{acc}}} (V_a)_{qp} & \text{if } x = w f_a(q), \\ P_{\mathrm{ACC}}(x) & \text{else,} \end{cases}$$

and $p'_{\mathrm{rej}} = p_{\mathrm{rej}} + \sum_{p \in Q_{\mathrm{rej}}} (V_a)_{qp}$.

For a string $x_1 \ldots x_n$ the map T_x is just the concatenation of the T_{x_i}. Observe that all the T_a conserve the probability.

Implicitely, we add initial and end marker symbols ($\ddagger, \$$) at the input, with additional stochastic matrices $V_\ddagger$ and $V_\$$, executed only at the very beginning, and at the very end. We assume that $V_\$$ puts no probability outside $Q_{\mathrm{acc}} \cup Q_{\mathrm{rej}}$.

By virtue of the computation, to each input string $v \in \Sigma_1^*$ there corresponds a probability distribution $T(\cdot|v)$ on the set $\Sigma_2^* \cup \{\mathrm{REJ}\}$:

$$T(\mathrm{REJ}|v) := T_{\ddagger v \$}((q_0, \epsilon), \mathbf{0}, 0)[\mathrm{REJ}]$$

is the probability to reject the input v, whereas

$$T(w|v) := T_{\ddagger v\$}((q_0, \epsilon), \mathbf{0}, 0)[w]$$

is the probability to accept, after having produced the output w.

Definition 2 *Let $\mathcal{R} \subset \Sigma_1^* \times \Sigma_2^*$.*

For $\alpha > 1/2$ we say that T computes the relation$\mathcal{R}$ with probabilityα *if for all v, whenever $(v, w) \in \mathcal{R}$, then $T(w|v) \geq \alpha$, and whenever $(v, w) \notin \mathcal{R}$, then $T(w|v) \leq 1 - \alpha$*

For $0 < \alpha < 1$ we say that T computes the relation$\mathcal{R}$ with isolated cutpointα *if there exists $\varepsilon > 0$ such that for all v, whenever $(v, w) \in \mathcal{R}$, then $T(w|v) \geq \alpha + \varepsilon$, but whenever $(v, w) \notin \mathcal{R}$, then $T(w|v) \leq \alpha - \varepsilon$.*

The following definition is modelled after the ones for pfst for quantum finite state automata [8]:

Definition 3 *A* quantum finite state transducer (qfst) *is a tuple $T = (Q, \Sigma_1, \Sigma_2, V, f, q_0, Q_{\text{acc}}, Q_{\text{rej}})$, where Q is a finite set of states, Σ_1, Σ_2 is the input/output alphabet, $q_0 \in Q$ is the initial state, and $Q_{\text{acc}}, Q_{\text{rej}} \subset Q$ are (disjoint) sets of accepting and rejecting states, respectively. The transition function $V : \Sigma_1 \times Q \to Q$ is such that for all $a \in \Sigma_1$ the matrix $(V_a)_{qp}$ is unitary, and $f_a : Q \to \Sigma_2^*$ is the output function.*

Like before, implicitely matrices $V_\ddagger$ and $V_\$$ are assumed, $V_\$$ carrying no amplitude from Q_{non} to outside $Q_{\text{acc}} \cup Q_{\text{rej}}$. The computation proceeds as follows: being in state q, and reading a, the machine prints $f_a(q)$ on the output tape, and moves to the superposition $V_a|q\rangle = \sum_p (V_a)_{qp}|p\rangle$ of internal states. Then a measurement of the orthogonal decomposition $E_{\text{non}} \oplus E_{\text{acc}} \oplus E_{\text{rej}}$ (with the subspaces $E_i = \text{span } Q_i \subset \ell^2(Q)$, which we identify with their respective projections) is performed, stopping the computation with accepting the input on the second outcome (while observing the output), with rejecting it on the third.

Here, too, we define total states: these are elements

$$(|\psi_{\text{NON}}\rangle, P_{\text{ACC}}, p_{\text{rej}}) \in \ell^2(Q \times \Sigma_2^*) \oplus \ell^1(\Sigma_2^*) \oplus \ell^1(\{\text{REJ}\}),$$

with norm

$$\|(|\psi_{\text{NON}}\rangle, P_{\text{ACC}}, p_{\text{rej}})\| = \||\psi_{\text{NON}}\rangle\|_2 + \|P_{\text{ACC}}\|_1 + |p_{\text{rej}}|.$$

At the beginning the total state is $(|q_0\rangle \otimes |\epsilon\rangle, \mathbf{0}, 0)$, the total state transformations, for

$$|\psi\rangle = \sum_{q \in Q} |q\rangle \otimes |\omega_q\rangle, \qquad \text{with } |\omega_q\rangle = \sum_{w \in \Sigma_2^*} \alpha_{qw}|w\rangle,$$

are (for $a \in \Sigma_1$)

$$T_a : (|\psi\rangle, P_{\text{ACC}}, p_{\text{rej}}) \mapsto \left(E_{\text{non}} \sum_q V_a|q\rangle \otimes |\omega_q f_a(q)\rangle, P'_{\text{ACC}}, p'_{\text{rej}} \right),$$

where $|\omega_q f_a(q)\rangle = \sum_w \alpha_{qw} |w f_a(q)\rangle$, and

$$P'_{\mathrm{ACC}}(x) = P_{\mathrm{ACC}}(x) + \left\| E_{\mathrm{acc}} \sum_{q,w \text{ s.t. } x=wf_a(q)} \alpha_{qw} V_a |q\rangle \right\|_2^2 ,$$

$$p'_{\mathrm{rej}} = p_{\mathrm{rej}} + \left\| E_{\mathrm{rej}} \sum_q V_a |q\rangle \otimes |\omega_q f_a(q)\rangle \right\|_2^2 .$$

Observe that the T_a do not exactly preserve the norm, but that there is a constant γ such that $\|T_a(X)\| \leq \gamma \|X\|$ for any total state X. Quite straightforwardly, the distributions $T(\cdot|v)$ are defined, and so are the concepts of computation with probability α or with isolated cutpoint α.

Observe also that we defined our model in closest possible analogy to quantum finite automata [8]. This is of course to be able to compare qfst to the latter. In principle however other definitions are conceivable, e.g. a mixed state computation where the T_a are any completely positive, trace preserving, linear maps (the same of course applies to quantum finite automata!). We defer the study of such a model to another occasion.

Notice the physical benefits of having the output tape: whereas for finite automata a superposition of states means that the amplitudes of the various transitions are to be added, this is no longer true for transducers if we face a superposition of states *with different output tape content*. I.e. the entanglement of the internal state with the output may prohibit certain interferences. This will be a crucial feature in some of our later constructions.

2 Quantum Finite Automata and Quantum Transducers

The definition of qfst is taylored in such a way that by excluding the output tape and the output function, we get a quantum finite automaton. One, however, with distinct acceptance and rejection properties, as compared to the qfst. Nevertheless, the decision capabilities of qfst equal those of quantum finite automata:

Theorem 4 *A language L is accepted by a 1–way quantum finite automaton with probability bounded away from 1/2 if and only if the relation $L \times \{0\} \cup \overline{L} \times \{1\}$ is computed with isolated cutpoint.*

Proof: First observe that for finite automata (probabilistic and quantum), recognizability with an isolated cutpoint is equivalent to recognizability with probability bounded away from 1/2 (by "shifting the cutpoint": just add in the ‡–step possibilities to accept or reject right away with certain probabilities). We have to exhibit two constructions:

Let there be given a quantum finite automaton. We may assume that it is such that $V_\$$ is a permutation on Q.

This can be forced by duplicating each $q \in Q_{\mathrm{acc}} \cup Q_{\mathrm{rej}}$ by a new state q', and modifying the transition function as follows: denote by σ the map interchanging

q with q' for $q \notin Q_{\text{non}}$, and being the identity on q, q' for $q \in Q_{\text{non}}$. Define a unitary U such that for $q \in Q_{\text{non}}$

$$U|q\rangle = \sum_p (V_\$)_{qp} |\sigma p\rangle,$$

and $U|q\rangle = |q\rangle$ for $q \in Q_{\text{acc}} \cup Q_{\text{rej}}$. Now let

$$V'_\ddagger := UV_\ddagger, \qquad V'_\$:= \sigma, \qquad V'_a := UV_aU^{-1}.$$

It is easily checked that this automaton behaves exactly like the initial one.

Construct a qfst as follows: its states are $Q \cup \widehat{Q}$, with $\widehat{Q} = \{\hat{q} : q \in Q_{\text{acc}} \cup Q_{\text{rej}}\}$ being the accepting states, and no rejecting states. Let the transition function be W with

$$\begin{aligned} W_a|q\rangle &= V_a|q\rangle \text{ for } q \in Q_{\text{non}}, \text{ but} \\ W_a|q\rangle &= |\hat{q}\rangle \text{ for } q \in Q_{\text{acc}} \cup Q_{\text{rej}}. \end{aligned}$$

Since $V_\$$ is the permutation σ on Q, we may define

$$W_\$|q\rangle = \begin{cases} |\widehat{\sigma q}\rangle & \text{for } \sigma q \in Q_{\text{acc}} \cup Q_{\text{rej}}, \\ |\sigma q\rangle & \text{for } \sigma q \in Q_{\text{non}}. \end{cases}$$

Finally, let the output function be (for $q \in Q$)

$$f_a(q) = \begin{cases} 0 & \text{for } q \in Q_{\text{acc}}, \\ 1 & \text{for } q \in Q_{\text{rej}}, \end{cases} \qquad f_\$(q) = \begin{cases} 0 & \text{for } \sigma q \in Q_{\text{acc}}, \\ 1 & \text{for } \sigma q \in Q_{\text{rej}}, \end{cases}$$

and ϵ in all other cases. It can be checked that it behaves in the desired way.

Given a qfst, construct a quantum finite automaton as follows: its states are $Q \times \Sigma_2^{\leq t}$, where the second component represents the tape content up to $t = 1 + \max_{a,q} |f_a(q)|$ many symbols. Initial state is (q_0, ϵ). Observe that by definition of the T_a amplitude that once is shifted onto output tapes of length larger than 1 is never recovered for smaller lengths. Hence we may as well cut such branches by immediate rejection: the states in $Q \times \Sigma_2^{\geq 2}$ are all rejecting, and so are $(Q_{\text{acc}} \cup Q_{\text{rej}}) \times \{1\}$. The accepting states are $Q_{\text{acc}} \times \{0\}$.

The transition function is partially defined by

$$W_a|q, x\rangle := \sum_{p \in Q} (V_a)_{qp} |p, x f_a(q)\rangle, \quad x \in \Sigma \cup \{\epsilon\},$$

(for $a = \$$ this is followed by mapping $|p, \epsilon\rangle$ to a rejecting state, while leaving the other halting states alone), i.e. the automaton performs like the qfst on the elements of Q, and uses the second component to simulate the output tape. We think of W_a being extended in an arbitary way to a unitary map. One can check that this construction behaves in the desired way. □

3 Decidability Questions

As is well known, the emptiness problem for the language accepted by a deterministic (or nondeterministic) finite automaton is decidable. Since the languages accepted by probabilistic and quantum finite automata with bounded error are regular [9,8], these problems are decidable, too.

For finite state transducers the situation is more complicated: In [6] it is shown that the emptiness problem for deterministic and nondeterministic fst is decidable. In contrast we have

Theorem 5 *The emptiness problem for pfst computing a relation with probability* $2/3$ *is undecidable.*

Likewise, the emptiness problem for qfst computing a relation with probability $2/3$ *is undecidable.*

Proof: By reduction to the Post Correspondence Problem [6] : let an instance $(v_1, \ldots, v_k)$, $(w_1, \ldots, w_k)$ of PCP be given (i.e. $v_i, w_i \in \Sigma^+$). It is to be decided whether there exists a sequence $i_1, \ldots, i_n$ $(n > 0)$ such that

$$v_{i_1} \cdots v_{i_n} = w_{i_1} \cdots w_{i_n}.$$

Construct the following qfst with input alphabet $\{1, \ldots, k\}$. It has the whoever states q_0, q_v, q_w, and q_{rej}. The initial transformation produces a superposition of $q_v, q_w, q_{\mathrm{rej}}$, each with amplitude $1/\sqrt{3}$. The unitaries U_i are all identity, but the output function is defined as $f_i(q_x) = x_i$, for $x \in \{v, w\}$. The endmarker maps q_v, q_w to accepting states. It is clear that $i_1, \ldots, i_n$ is a solution iff $(i_1 \ldots i_n, v_{i_1} \cdots v_{i_n})$ is in the relation computed with probability $2/3$ (the automaton is easily modified so that it rejects when the input was the empty word, in this way we force $n > 0$).

By replacing the unitaries by stochastic matrices (with entries the squared moduli of the corresponding amplitudes) the same applies to pfst.

Since it is well known that PCP is undecidable, it follows that there can be no decision procedure for emptiness of the relation computed by the constructed pfst, or qfst, respectively. □

Remark 6 *Undecidable questions for quantum finite automata were noted first for "*$1\frac{1}{2}$*–way" automata, i.e. ones which move only to the right on their input, but may also keep their position on the tape. In [1] it is shown that the equivalence problem for these is undecidable. The same was proved for 1–way–2–tape quantum finite automata in [3].*

Conjecture 7 *The emptiness problem for probabilistic and quantum fst computing a relation with probability* 0.99 *is decidable.*

The emptiness problem for probabilistic and quantum fst computing a relation with a single–letter input alphabet, with probability $1/2 + \varepsilon$ *is decidable.*

To prove this, we would like to apply a packing argument in the space of all total states, equipped with the above metric. However, this fails because of the infinite volume of this space (for finite automata it is finite, see [9] and [8]). In any case, a proof must involve the size of the gap between the upper and the lower probability point, as the above theorem shows that it cannot possibly work with gap 1/3.

Still, we can prove:

Theorem 8 *If the relation $\mathcal{R}$ is computed by a pfst or a qfst with an isolated cutpoint, then* $\mathrm{Range}(\mathcal{R}) = \{y : \exists x\ (x, y) \in \mathcal{R}\}$ *is a recursive set (so, for each specific output, it is decidable if it is ever produced above the threshold probability).*

We regret to omit the proof because of the page limit. The proof can be found in [5]. □

4 Deterministic vs. Probabilistic Transducers

Unlike the situation for finite automata, pfst are strictly more powerful than their deterministic counterparts:

Theorem 9 *For arbitrary $\varepsilon > 0$ the relation*

$$\mathcal{R}_1 = \{(0^m 1^m, 2^m) : m \geq 0\}$$

can be computed by a pfst with probability $1-\varepsilon$. It cannot be computed by a dfst.

Proof: The idea is essentially from [4]: for a natural number k choose initially an alternative $j \in \{0, \ldots, k-1\}$, uniformly. Then do the following: repeatedly read k 0's, and output j 2's, until the 1's start (remember the remainder modulo k), then repeatedly read k 1's, and output $k-j$ 2's. Compare the remainder modulo k with what you remembered: if the two are equal, output this number of 2's and accept, otherwise reject.

It is immediate that on input $0^m 1^m$ this machine outputs 2^m with certainty. However, on input $0^m 1^{m'}$ each 2^n receives probability at most $1/k$.

That this cannot be done deterministically is straightforward: assume that a dfst has produced $f(m)$ 2's after having read m 0's. Because of finiteness there are k, l such that after reading k 1's (while n_0 2's were output) the internal state is the same as after reading l further 1's (while n 2's are output). So, the output for input $0^m 1^{k+rl}$ is $2^{f(m)+n_0+rn}$, and these pairs are either all accepted or all rejected. Hence they are all rejected, contradicting acceptance for $m = k+rl$. □

By observing that the random choice at the beginning can be mimicked quantumly, and that all intermediate computations are in fact reversible, we immediately get

Theorem 10 *For arbitrary $\varepsilon > 0$ the relation $\mathcal{R}_1$ can be computed by a qfst with probability $1-\varepsilon$.* □

Note that this puts qfst in contrast to quantum finite automata: in [2] it was shown that if a language is recognized with probability strictly exceeding 7/9 then it is possible to accept it with probability 1, i.e. reversibly deterministically.

5 ... vs. Quantum Transducers

After seeing a few examples one might wonder if everything that can be done by a qfst can be done by a pfst. That this is not so is shown as follows:

Theorem 11 *The relation*

$$\mathcal{R}_3 = \{(0^m 1^n 2^k, 3^m) : n \neq k \wedge (m = k \vee m = n)\}$$

can be computed by a qfst with probability $4/7 - \varepsilon$*, for arbitrary* $\varepsilon > 0$.

Theorem 12 *The relation* $\mathcal{R}_3$ *cannot be computed by a pfst with probability bounded away from* $1/2$*. In fact, not even with an isolated cutpoint.*

Proof (of theorem 11): For a natural number l construct the following transducer: from q_0 go to one of the states q_1, $q_{j,b}$ ($j \in \{0, \ldots, l-1\}$, $b \in \{1, 2\}$), with amplitude $\sqrt{3/7}$ for q_1 and with amplitude $\sqrt{2/(7l)}$ each, for the others. Then proceed as follows (we assume the form of the input to be $0^m 1^n 2^k$, others are rejected): for q_1 output one 3 for each 0, and finally accept. For $q_{j,b}$ repeatedly read l 0's and output j 3's (remember the remainder $m \mod l$). Then repeatedly read l b's and output $l - j$ 3's (output nothing on the $(3 - b)$'s). Compare the remainder with the one remembered, and reject if they are unequal, otherwise output this number of 3's. Reading \$ perform the following unitary on the subspace spanned by the $q_{j,b}$ and duplicate states $q_{j',b}$:

$$(j \leftrightarrow j') \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Accepting are all $q_{j',2}$, rejecting are all $q_{j',1}$.

It can be checked that the automaton behaves in the desired way. □

Proof (of theorem 12): By contradiction. Suppose $\mathcal{R}_3$ is computed by a pfst T with isolated cutpoint α. It is easily seen that the cutpoint can be shifted $1/2$.

Hence, we may assume that T computes $\mathcal{R}$ with probability $\varphi > 1/2$, from this we shall derive a contradiction. The state set Q together with any of the stochastic matrices V_0, V_1, V_2 is a Markov chain. We shall use the classification of states for finite Markov chains (see [7]): for V_i Q is partitioned into the set R_i of *transient* states (i.e. the probability to find the process in R_i tends to 0) and a number of sets S_{ij} of *ergodic* states (i.e. once in S_{ij} the process does not leave this set, and all states inside can be reached from each other, though maybe only by a number of steps). Each S_{ij} is divided further into its *cyclic* classes $C_{ij\nu}$ ($\nu \in \mathbb{Z}_{d_{ij}}$), V_i mapping $C_{ij\nu}$ into $C_{ij\nu+1}$. By considering sufficiently high powers V_i^d (e.g. product of all the periods d_{ij}) as transition matrices, all these cyclic sets become ergodic, in fact, V_i^d restricted to each is *regular*.

Using only these powers amounts to concentrating on input of the form $\mathbf{0}^m \mathbf{1}^n \mathbf{2}^k$, with $\mathbf{i} = i^d$, which we will do from now on. Relabelling, the ergodic sets of $V_{\mathbf{i}} = V_i^d$ will be denoted S_{ij}. Each has its unique equilibrium distribution, to which every initial one converges: denote it by π_{ij}. Furthermore, there

are limit probabilities $a(j_0)$ to find the process $V_{\mathbf{0}}$ in S_{0j_0} after long time, starting from q_0. Likewise, there are limit probabilities $b(j_1|j_0)$ to find the process $V_{\mathbf{1}}$ in S_{1j_1} after long time, starting from π_{0j_0}, and similarly $c(j_2|j_1)$. So, by the law of large numbers, for large enough m, n, k the probability that $V_{\mathbf{0}}$ has passed into S_{0j_0} after $\sqrt{m}$ steps, after which $V_{\mathbf{1}}$ has passed into S_{1j_1} after $\sqrt{n}$ steps, after which $V_{\mathbf{2}}$ has passed into S_{2j_2} after $\sqrt{k}$ steps, is arbitrarily close to $P(j_0, j_1, j_2) = a(j_0)b(j_1|j_0)c(j_2|j_1)$. (Note that these probabilities sum to one).

Applying the ergodic theorem (or law of large numbers) and standard techniques (see [7]) one arrives at the conclusion that, if all inputs $\mathbf{0}^m\mathbf{1}^m\mathbf{2}^k$, with $k \neq m$, accept with output 3^{dm} above the cutpoint, so must $\mathbf{0}^m\mathbf{1}^m\mathbf{2}^m$, contradicting the fact that $(0^{dm}, 1^{dm}2^{dm}, 3^{dm})$ is not in the relation.

□

In general however, computing with isolated cutpoint is strictly weaker than with probability bounded away from 1/2 (observe that for finite automata, probabilistic and quantum, recognizability with an isolated cutpoint is equivalent to recognizability with probability bounded away from 1/2, see theorem 4):

Theorem 13 *The relation*

$$\mathcal{R}_4 = \{(0^m1^na, 4^l) : (a = 2 \to l = m) \wedge (a = 3 \to l = n)\}$$

can be computed by a pfst and by a qfst with an isolated cutpoint (in fact, one arbitrarily close to 1/2*), but not with a probability bounded away from* 1/2*.*

Proof: First the construction (again, only for qfst): initially branch into two possibilities c_0, c_1, each with amplitude $1/\sqrt{2}$. Assume that the input is of the correct form (otherwise reject), and in state c_i output one 4 for each i, ignoring the $(1 - i)$'s. Then, if $a = 2 + i$, accept, if $a = 3 - i$, reject. It is easily seen that 4^l is accepted with probability 1/2 if $(0^m1^na, 4^l) \in \mathcal{R}_4$, and with probability 0 otherwise.

That this cannot be done with probability above 1/2 is clear intuitively: the machine has to produce some output (because of memory limitations), but whether to output 4^m or 4^n it cannot decide until seeing the last symbol. Formally, assume that $|m - n| > 4t$, with $t = \max_{a,q} |f_a(q)|$. If

$$T_{\ddagger 0^m1^n2\$}((q_0, \epsilon), \mathbf{0}, 0)[4^m] = T(4^m|0^m1^n2) \geq 1/2 + \delta,$$

necessarily

$$T_{\ddagger 0^m1^n}((q_0, \epsilon), \mathbf{0}, 0)[4^m] + T_{\ddagger 0^m1^n}((q_0, \epsilon), \mathbf{0}, 0)[Q_{\text{non}} \times 4^{[m-2t,m+2t]}] \geq 1/2 + \delta.$$

But this implies

$$T_{\ddagger 0^m1^n}((q_0, \epsilon), \mathbf{0}, 0)[4^n] + T_{\ddagger 0^m1^n}((q_0, \epsilon), \mathbf{0}, 0)[Q_{\text{non}} \times 4^{[n-2t,n+2t]}] \leq 1/2 - \delta,$$

hence

$$T_{\ddagger 0^m1^n3\$}((q_0, \epsilon), \mathbf{0}, 0)[4^n] = T(4^n|0^m1^n3) \leq 1/2 - \delta,$$

contradicting $(0^m1^n3, 4^n) \in \mathcal{R}_4$. □

To conclude from these examples, however, that quantum is even better than probabilistic, would be premature:

Theorem 14 *The relation*

$$\mathcal{R}_5 = \{(wx, x) : w \in \{0,1\}^*, x \in \{0,1\}\}$$

cannot be computed by a qfst with an isolated cutpoint. (Obviously it is computed by a pfst with probability 1*, i.e. a dfst).*

Proof: The construction of a dfst computing the relation is straightforward. To show that no qfst doing this job exists, we recall from [8] that $\{0,1\}^*0$ is not recognized by a 1–way quantum finite automaton with probability bounded away from 1/2, and use theorem 4 for this language. □

Acknowledgements

Research of RF supported by contract IST–1999–11234 (QAIP) from the European Commission, and grant no. 96.0282 from the Latvian Council of Science. This work was carried out during RF's stay at Bielefeld University in October 2000. At this time AW was at Fakultät für Mathematik, Universität Bielefeld, Germany, and was supported by SFB 343 "Diskrete Strukturen in der Mathematik" of the Deutsche Forschungsgemeinschaft.

References

1. M. Amano, K. Iwama, "Undecidability on Quantum Finite Automata", in Proc. 31st STOC, 1999, pp. 368–375.
2. A. Ambainis, R. Freivalds, "1–way quantum finite automata: strengths, weaknesses, and generalizations", in Proc. 39th FOCS, 1998, pp. 332–341.
3. R. Bonner, R. Freivalds, R. Gailis, "Undecidability of 2–tape quantum finite automata", in Proceedings of *Quantum Computation and Learning. Sundbyholms Slott, Sweden, 27–29 May, 2000*, R. Bonner and R. Freivalds (eds.), Malardalen University, 2000, pp. 93–100.
4. R. Freivalds, "Language recognition using finite probabilistic multitape and multihead automata", Problems Inform. Transmission, vol. 15, no. 3, 1979, pp. 235–241.
5. R. Freivalds, A. Winter, "Quantum Finite State Transducers", http://xxx.lanl.gov/ps/quant-ph/0011052
6. E. Gurari, *Introduction to the Theory of Computation*, Computer Science Press, 1989.
7. J. G. Kemeny, J. L. Snell, *Finite Markov Chains*, Van Nostrand, Princeton, 1960.
8. A. Kondacs, J. Watrous, "On the power of quantum finite state automata", in Proc. 38th FOCS, 1997, pp. 66–75.
9. M. O. Rabin, "Probabilistic Automata", Information and Control, vol. 6, 1963, pp. 230–245.
10. D. Scott, "Some definitional suggestions for automata theory", J. of Comput. and Syst. Science, 1967, pp. 187–212.

Lemmatizer for Document Information Retrieval Systems in JAVA

Leo Galambos*

Department of Software Engineering, Faculty of Mathematics and Physics,
Charles University, Prague

Abstract. Stemming is a widely accepted practice in Document Information Retrieval Systems (DIRs), because it is more benefical than harmful [3] as well as having the virtue of improving retrieval efficiency by reducing the size of the term index. We will present a technique of semi-automatic stemming that is fine designed for JAVA environment. The method works without deep knowledge of grammar rules of a language in contradistinction to well-known Porter's algorithm [8]. From that point of view, we can call our method universal for more languages. We will also present tests to show quality of the method and its error-rate.

1 Introduction

Most document information retrieval systems operate with free text documents, where we cannot guarantee terms in their base forms. That situation can be difficult, because an user may put his query in any form and we would solve that query against huge set of documents that use any valid variant of the terms of that query. DIR system has to recognize these variants somehow. A solution, we can choose, is rather simple. The problem can be overcome with the substitution of the words by their respective base forms (so called "stems"). The process of transformation is called "stemming" and the modul of DIR that provides that functionality is called "lemmatizer". In this paper we will suppose, that the lemmatizer does not offer anything else than the stemming function. In common case it is not true, because the lemmatizer can provide a rich text analysis (i.e. recognition of nouns, verbs etc.) that can improve quality of DIR system.

Stemming is benefical, because it reduces the number of terms (and also size of data streams) in a system. Our goal is not to develop a perfect, error free stemming method. We need to provide a method that transforms a huge set of words to a small set of stems (corpus of a system) without a noise. The noise can be defined as a situation, when a word is transformed to a stem of another word. If we allow that noise, we lost meaning of a document.

* This work was supported by the GACR grant N° 201/00/1031.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 243–252, 2001.

There are four main types of stemming strategies: table lookup (equal to our basic reduction method), successor variety, n-grams and affix removal (as good as our best reduction methods). Table lookup consists simply of looking for the stem of a word in a table. Our basic reduction method does the same thing with a small modification - it looks for the edit command, that transforms a word to its stem. But both methods are dependent on table (pairs word-stem) for the whole language and it might not be available. Successor variety is more complex and is based on the determination of morpheme boundaries (see more in structural linquistics). N-grams strategy is based on identification of n-grams. Affix removal stemming is more simple, and can be implemented efficiently if we know the rules for affix removal in language. Our method tries to solve the problem, when we cannot supply the rules.

The tacit assumption in our paper is that we must get a sample set (so-called dictionary) of words and their stems. The dictionary cannot be generated by an automat, so we called our method "semi-automatic". We will show how we can define a set of transformation commands (P-commands) and how we can store the commands with original words. If the structure exists, we can create a stem of a word, because we have got a word and a transformation command that builds the stem. We will present the solution of two problems. How can we reduce the original huge data structure? And how can we transform words that we did not get with the original dictionary.

The method would be usable in JAVA environment. What does it mean for implementation? First, the data structure of the method would be small and static. We would like to achieve a small size of the structure - i.e. under 64kB. Second, the data structure must allow a nice support of threads for a better scalability. Third, processing of a document of length l would be done with time complexity $O(l)$. Fourth, we must achieve a very small overhead on processing time.

2 Stemming

We use the definition of stemming, that has been presented by Boot [1]: "Stemming is the transformation of all inflected word forms contained in a text to their dictionary look-up form". The dictionary look-up form is called a lemma. A stem is the portion of a word which is left after removal of its prefixes and suffixes. In our case, we do not need to recognize stem and lemma so well. If there are more dictionary look-up forms, we will choose just one of them and we will call it - a stem.

Let's define a function *stem* that transforms a word to its stem. The function (*stem*) mostly tranforms a word to its stem after removal of prefixes and suffixes. Sometimes the function must also insert dead characters (i.e. 'e' in English etc.). Our tests showed that we would also accept a 'replace' action in the *stem* function. That action concentrates removal and insertion in some cases. The

patch command (P-command) represents a functionality of the *stem* function. We must ensure that P-commands are universal, and P-commands can transform any word to its stem.

Our solution is based on Levenstein metric [6][1] that produces P-command as the minimum cost path in a directed graph (see below). The metric computes the distance between two strings (in our case between a word and its stem) as measured by the minimum cost sequence of edit operations. These basic operations will be called partial patch commands (PP-commands) in this paper. PP-command is responsible for an atomic transformation of a string. In previous text we explained that we will need three basic operations: removal, that deletes a sequence of characters; insertion, that inserts a character; substitution, that rewrites a character; and finaly we will also need a NOP (no-operation) PP-command that skips a sequence of characters.

One can imagine the P-command as an algorithm for an operator (editor) that rewrites a string to another string. We will slightly redefine the three basic operations, because we must also define the cursor movement with more precision: removal - deletes a sequence of characters starting at the current cursor position and moves the cursor to the next character (the length of that sequence is the parameter of this PP-command); insertion - inserts a character *ch*, cursor does not move (the character *ch* is a parameter); substitution - rewrites a character at the current cursor position to the character *ch* and moves the cursor to the next character (the character *ch* is a parameter).

Note 1. We need not store the last NOP command, because it does not change the string: "remove 3 characters, skip (NOP) 6 characters" is equal to "remove 3 characters". On the other hand, we cannot omit a significant NOP command: "remove three charaters, skip one character, insert 'abc' string, skip four characters". That P-command would be optimalized as "remove three charaters, skip one character, insert 'abc' string".

We will apply the P-commands backward (right to left), because there are more suffixes than prefixes. It can reduce the set of P-commands, because the last NOP is not stored. That NOP only moves the cursor to the end of a string without any changes. *stem*(eats) =eat, *stem*(provides) =provide; if we apply the P-command left to right, we would need two commands "skip three characters, remove one character" and "skip seven characters, remove one character"; if we apply the P-command right to left, we need only single command "remove one character".

PP-commands will be written as a pair of two characters. The first character will represent an operation (or instruction) and the second one will hold a parameter. We use these marks for our basic operations: 'D' - removal, 'I' - insertion, 'R' -

[1] Levenstein metric was chosen, because it will allow us the same modifications of a term as Porter's algorithm or any affix removal algorithm. The metric is also appropriate for English and Czech (after conversion to us-ascii) languages.

substitution, '-' - NOP. The second character is a parameter of PP-command. Sometimes we have to store a number[2] in the parameter (i.e. removal and NOP PP-commands). The encoding function we chose is defined as: 'a' = '1', 'b'='2', 'c'='3' etc. We have learned, that an original string o can be transformed to a string s with instructions of a P-command p. Let's define function *patch* that covers that process: $patch(o, p) = s$.

Example 1. "skip three characters, remove one character" will be represented by "-c" and "Ra" instructions. We will use shorter record "-cRa" for the complete P-command, i.e. patch('ABC','Rv-aRs')='sBv', patch('AB','IaIbDb')='ba'.

3 Building P-Commands

Our current implementation [5] uses a naive algorithm[3] that builds P-commands. A speedup can be achieved with more powerful algorithms, see [2], this is not aim of this paper.

Definition 1. *(Levenstein [6]) Let* $G = (N, E, m, w) = (N, E)$ *be a directed graph for two strings* $X = X_1X_2 \dots X_{|X|}$ *and* $Y = Y_1Y_2 \dots Y_{|Y|}$*, where* $N = \{[i, j]; (0 \leq i \leq |X|) \wedge (0 \leq j \leq |Y|)\}$*,* $E = \{\langle [i_1, j_1]; [i_2, j_2] \rangle ; (0 \leq i_2 - i_1 \leq 1) \wedge (0 \leq j_2 - j_1 \leq 1) \wedge ([i_1, j_1], [i_2, j_2] \in N)\}$*,* $Z = [0, 0]$ *and* $K = [|X|, |Y|]$*.*

Painting function m*:* $E \rightarrow PP - command$ *is given* $\forall e = \langle [i_1, j_1]; [i_2, j_2] \rangle \in E$ *by* $m(e) =$*Substitution* $\Leftrightarrow_{df} ((y_2 = y_1 + 1) \wedge (x_2 = x_1 + 1) \wedge (X_{x_2} \neq Y_{y_2}))$*;* $m(e) =$*NOP* $\Leftrightarrow_{df} ((y_2 = y_1 + 1) \wedge (x_2 = x_1 + 1) \wedge (X_{x_2} = Y_{y_2}))$*;* $m(e) =$ *Removal* $\Leftrightarrow_{df} ((x_1 = x_2) \wedge (y_2 = y_1 + 1))$*;* $m(e) =$ *Insertion* $\Leftrightarrow_{df} ((y_1 = y_2) \wedge (x_2 = x_1 + 1))$*.*

The penalty (weight) of path $P = \{p_1, p_2, \dots p_n\}$ *from* Z *to* K *is the cost of the path, described by* $w(P) = \sum_{(p_i)} w(m(p_i))$*.*

P-command is represented by a path through a directed graph $G = (N, E)$. For example, the graph of transformation 'ABCD' to 'BDD' is presented in Fig. 1. We search for the shortest weighted path from $Z \in N$ to $K \in N$. Any path from Z to K represents one P-command that transforms term 'ABCD' to 'BDD' (in case of Fig. 1). We would also describe edges of the figure: horizontal edges represent PP-command 'Removal', vertical edges - 'Insertion', solid diagonal edges - 'Substitution' and the last edges - 'NOP'. Each edge of the shortest weighted path through the directed graph can be individually encoded. The coding is rather simple and is based on function m and our coding of PP-commands (see above

[2] The number varies between 1 and 32, if we suppose words shorter than 32 characters. This is a realistic proposition.

[3] Our implementation works with complexity $O(l_1 l_2)$, where l_1 is the length of the original string and l_2 is the length of its final form (stem). There are many algorithms that have better complexity.

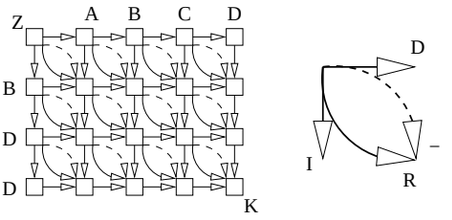


Fig. 1. Directed graph for building P-command that transforms 'ABCD' to 'BDD'.

- marks for our basic operations). We define the attribute of the PP-command for an edge type $m(\langle [a,b];[u,v] \rangle)$: 'Substitution' - attribute is character Y_v; 'Removal' - attribute is number 1; 'Insertion' - attribute is character Y_v,; 'NOP' - attribute is number 1. Sometimes we can optimalize a block (sequence) of PP-commands 'Removal' or 'NOP', i.e. '-a-a-a' will be rewritten to '-c' by our algorithm we use.

Our tests showed, that it is profitable, when we accept the path with more 'Removal' edges (iff $w('Removal')$ is cheap). For complete results, see Table 1. We would also choose the path that starts with a longer 'Removal' instruction.

Table 1. Number of P-commands for various weighting.

$w('Removal')$	$w('Insertion')$	$w('Substitution')$	$w('NOP')$	#P-commands
1	1	1	0	145
1	1	2	0	149
1	2	2	1	162
1	3	1	0	161
0	1	2	0	149
2	3	2	0	175

Example 2. A word 'momww' is transformed to its stem by P-commands 'Da' or '-aDa'. The first command is more universal and there is a chance we will use it in another transformation. If we vote for the first command, the set of P-commands can be smaller.

4 Data Structure

The dictionary (words and their P-commands[4]) will be stored in a trie. The key of that trie is an original word, see Fig. 2, and P-commands are stored in

[4] Now, the dictionary of words and their stems will be stored as the dictionary of words and relevant P-commands. This only changes representation of the dictionary and helps us with reduction of data structure.

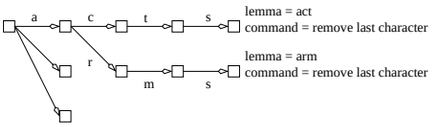


Fig. 2. Trie. Dictionary representation.

leaf nodes. The goal of this section is the reduction of that trie. We provided tests with an English dictionary of 25802 words (14760 stems). The dictionary is available on homepage of JAVA search engine[5]. Original pure data structure of the trie takes 85170 nodes. There are 140 P-commands. We mentioned 145 P-commands in the previous section, but they were calculated by a case sensitive algorithm that will not be used for practical reasons.

The trie can find a P-command for each word[5] of length l and then we can create a stem with (time) complexity $O(l)$. There are two main disadvantages. First, the data structure of the trie is huge. One node takes around 3 bytes in memory and it leads to final 250kB of the trie object. Second, we cannot run stemming against a term that was not in the original dictionary. The reductions (see below) would solve these two problems.

We will suppose that the trie is implemented by matrix $M_{m \times n}$, where m is the number of inner nodes and n is the number of characters (that appear in keys). Each inner node u is represented by the array M_u of edges that start in that inner node. The cell M_{ij} contains two values: pointer rf to the next inner node ($M_{ij}.rf$) and P-command *com* ($M_{ij}.com$). The cell $b = M_{uo}$ represents an edge $e = (u, v, o)$ of character o from inner node u to node $v = b.rf$; or an leaf node that holds value $b.com$ (the situation is clarified by Fig. 3).

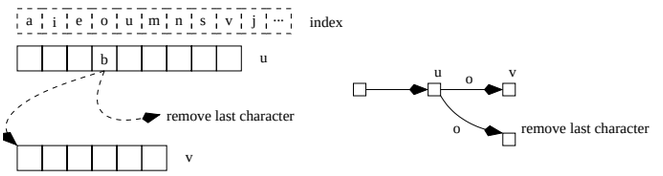


Fig. 3. Data structure of a trie.

Definition 2. *Let u and v be vectors. $R(u, v)$ represents a joint-collision of vectors u and v, $R(u, v) = \begin{cases} true \\ false \end{cases}$. When $R(u, v) = true$, then joint of u and v produces a collision.*

[5] Word of original dictionary.

Our reduction algorithms use simple merging strategy. They merge rows of a matrix, if it does not create a collision. The term 'collision' will be defined later, because it differs in the algorithms.

The algorithm will not produce the smallest trie, but our tests showed us that the final trie is good enough for our purposes. The main strategy is summarized in the next algorithm.

Algorithm (Reduction algorithm)
1. **read** matrix M.
2. **for** $j = m$ **to** 2 **do begin**
3. $r = M_j$
4. **for** $i = j - 1$ **to** 1 **do**
5. **if** we can merge (or rather join) M_i and r without a collision **then begin**
6. join M_i and r: $M_i = join(M_i, r)$, remove the row r
7. repair references to the row r (they must point to M_i)
8. **end**
9. **end**
10. reorder rows of matrix M, because the steps (6-7) produce gaps
11. **store** matrix M.

Definition 3. *Let x and y be vectors $(\langle com; rf\rangle)_i$. Then $join(x, y) = z$, where z is a vector $(\langle com; rf\rangle)_i$:*

$$z_i.field = \begin{cases} x_i.field \text{ iff } x_i.field \neq null \\ y_i.field \text{ iff } x_i.field = null \end{cases}, \forall field \in \{com; rf\}.$$

Basic Reduction. The intuition behind the method of joining two rows without any collision is naive. We will join two rows (arrays), if they are equal: $R(u, v) = \neg \bigwedge_i (u_i = v_i)$. That method mainly joins only leafs of an original trie, because the probability of a collision grows up when we move from leafs to rich inner nodes. It is obvious that the 'learning' factor will not be so high, but we would like to mention the method here, because it shows us the worse solution. The question is to what degree, if any, we can improve this method.

Tight Reduction. The intuition behind this method is also very simple. We required a perfect harmony of two rows in the previous method. Now, we will allow a joint, when there is no collision in not-null values of the rows. For example, a trie contains two rows (inner nodes) u and v. There are two edges: from u to g with a character c_u, and the second one from v to h with a character c_v. There is no collision if $c_v \neq c_u$. The basic reduction would require $(c_v = c_u \wedge g = h)$. But now, a collision becomes: $R(u, v) = \neg \bigwedge_i (u_i = v_i \bigvee \neg(u_i \bowtie_{com} v_i \vee u_i \bowtie_{rf} v_i))$, where $u_i \bowtie_{field} v_i \Leftrightarrow_{df} (u_i.field \neq null \wedge v_i.field \neq null \wedge u_i.field \neq v_i.field)$. It is obvious that we lost some information, i.e. we will not be able to recognize

words of the original dictionary. On the other hand, we can produce stems of words that we did not see. The walk through a trie must be modified in case of a trie after tight reduction. Because the reduction moves values from leafs to inner nodes we must handle that situation. We solved the problem with naive modification of a trie algorithm. By default, we go through trie and if we stop in a leaf, the leaf value is returned. If we cannot return that value (i.e. next step is not defined in a trie), last value that we met in an inner node of a trie is returned.

Multilevel Trie. It seems, that we can generate smaller tries, when we have a small set of P-commands. One would also recommend two level algorithm. The first level would recognize prefixes and the second one would recognize suffixes. Our P-commands can test these two approaches, because the commands can be easily disassembled. The first PP-commands of P-commands will form the first group (level 1), the second ones will form level 2, and so on. Our approach is stronger than the prefix/suffix recognition (see above), because we allow to disassemble the prefixes and suffixes to their atomic forms. In each level we can try and test the reduction methods (basic and tight).

5 Results

We use an English dictionary that contains 14760 stems of 25802 words (available on JAVA search engine homepage [5]). The dictionary was converted to lowercase. We also prefered 'Removal' over 'NOP' and we operated with 140 P-commands. The original trie (without reduction) contained 85170 nodes (60097 inner nodes). The dictionary generated 29 collisions, because 29 words were transformed to more than one stem.

We have also tried two tries. The first one reads the key left to right - it is a classic trie, we call it 'forward trie'. And the second one reads the key right to left (the key is read in reverse order), we call it 'back trie'. The second variant is suitable for a language where we suppose more suffixes (i.e. English). Results of each method (in this paper) are presented in Table 2. More detailed view of multilevel reduction is presented in Table 3.

Table 2. Results (number of nodes/number of inner nodes) of methods in this paper.

Method	Forward trie	Back trie
original trie	85170/60097	72551/50592
basic reduction	26716/11895	37725/17401
tight reduction	14431/1307	12714/977
multilevel, basic	194013/139310	159147/111158
multilevel, tight	28377/2310	28807/2053

Table 3. Number of nodes (total/inner nodes) is presented for each level (after BR - basic reduction, and TR - tight reduction) of multilevel trie.

Trie level	BR-Forward	BR-Back
1	85170/60097	72551/50592
2	85156/60085	72485/50537
3	22080/17839	12904/9089
4	1048/834	782/605
5	416/334	280/212
6	74/63	69/58
7	45/38	52/45
8	12/10	12/10
9	12/10	12/10
total	194013/139310	159147/111158

Trie level	TR-Forward	TR-Back
1	14216/1163	13783/930
2	12174/937	13070/906
3	1551/131	1615/153
4	293/47	178/32
5	72/14	91/16
6	33/10	29/8
7	22/4	25/4
8	8/2	8/2
9	8/2	8/2
total	28377/2310	28807/2053

The point is, that we can transform any word of the original dictionary to its stem without an error (there is a problem of noise, or rather collisions, but we omit that cases). How does the method work when we did not operate with a complete dictionary? That is the question we must answer.

Note 2. We do not use cross-validation with other methods, because we could not guarantee input of valid terms for testing. If we can do so, we can also append them to our dictionary. In this case our method would not return wrong results (errors), because any trie after reduction (see above) still resolves the words from the original dictionary. On the other hand we could test speed of the methods, but from theoretical point of view we know, that the complexity of our algorithms is linear (when the maximal length of stems is constant).

Error-Rate. The test was made with an English dictionary and our best methods (back trie). Some words of the dictionary were hidden, but the test was made against full dictionary, see Table 4. For example, when we built a trie, we did not accept 10% of words (random selection) that we could read in a dictionary. But we used them for testing of error-rate. The table presents approx. values of 100 runs. We would like to mention that our tests cannot cover all real situations in DIS. Another (and better) method of testing would be more complex and that is why it cannot be introduced in the size of this paper.

Table 4. Error-rate of back trie.

Method	tight reduction				basic reduction				multilevel tight			
unknown words in input (%)	10	15	20	30	10	15	20	30	10	15	20	30
error-rate (%)	3	5	7	11	10	14	20	30	5	7	10	15
#P-commands	134	127	128	119	133	129	127	123	133	129	128	125

Note 3. (Table 4) A dictionary has W words. We hide p (%) of words when the trie is built (unknown words in input). The set of $W \cdot (1 - p)$ words is transformed to a set of stems by a set of P-commands (#P-commands). That trie is optimalized by our methods. Then we test the trie (after reduction) with the full set of W words. It is obvious that we produce C stems wrong ($C \leq W \cdot p$). Then the error-rate becomes: $\frac{C}{W} \cdot 100\%$.

Conclusions

Our major intention of this research paper was to provide an universal stemming technique that can be easily implemented in JAVA. The data structures we used are static and small in size and they can be used in multithreaded environment. In combination, the methods we have described allow two important resources - processing time and memory space to be simultaneously reduced.

Another important feature is the semi-automatic processing of any language. We do not need to study the language, because the methods we presented here can learn common prefixes and suffixes without human being. On the other hand, we cannot say, that the process is automatic, because the methods are based on the dictionary of words and stems that must be prepared by a linguist. The good point is, that the dictionary need not be perfect and rich (see error-rate). Furthermore, we have also shown that the algorithm that removes suffixes and prefixes in two rounds does not save more memory space (see multilevel trie), and that the algorithm can have an opposite effect.

The concept of semi-automatic stemming is being integrated into the existing DIR system in JAVA (Perestroika engine [5]).The implementation of the engine shows us, that there are some open questions that we did not describe in that paper. Can we realize a reduction algorithm, that is optimal and fast? Can we achieve better results when we accept only frequent P-commands?

References

1. Boot, M.: Homography and Lemmatization in Dutch Texts. ALLC Bulletin 8 (1980), pp 175-189.
2. Berghel, H., Roach, D.: An Extension of Ukkonen's Enhanced Dynamic Programming ASM Algorithm. The ACM Trans. on Inf. Sys. 14:1, January 1996, pp 94-106.
3. Cleverdon, C.W., Mills, J., Keen, M.: Factors Determining the Performance of Indexing Systems. 2 vols. College of Aeronautics, Cranfield 1966.
4. Elmasre, R., Navathe, S.B.: Fundamentals of Database Systems. Benjamin Cummins, Menlo Park, California 1989.
5. Galambos, L.: Perestroika, Search Engine in JAVA. World Wide Web, homepage http://com-os2.ms.mff.cuni.cz/Perestroika/, Prague 1998-2001.
6. Levenstein, V.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Sov. Phys, Dokl., 10 (1966), pp 707-710.
8. Porter, M.F.: An Algorithm For Suffix Stripping. Program 14(3), July 1980, pp 130-137.

The Reconstruction of Polyominoes from Approximately Orthogonal Projections*

Maciej Gębala

[1] Institute of Mathematics, Opole University
[2] Institute of Computer Science, University of Wrocław
Przesmyckiego 20, 51–151 Wrocław, Poland
mgc@ii.uni.wroc.pl

Abstract. The reconstruction of discrete two-dimensional pictures from their projection is one of the central problems in the areas of medical diagnostics, computer-aided tomography, pattern recognition, image processing, and data compression. In this note, we determine the computational complexity of the problem of reconstruction of polyominoes from their approximately orthogonal projections. We will prove that it is NP-complete if we reconstruct polyominoes, horizontal convex polyominoes and vertical convex polyominoes. Moreover we will give the polynomial algorithm for the reconstruction of hv-convex polyominoes that has time complexity $O(m^3n^3)$.

1 Introduction

1.1 Definition of Problem

A finite binary picture is an $m \times n$ matrix of 0's and 1's, when the 1's correspond to black pixels and the 0's correspond to white pixels. The i-th *row projection* and the j-th *column projection* are the numbers of 1's in the i-th row and of 1's in the j-th column, respectively. In a reconstruction problem, we are given two vectors $H = (h_1, \ldots, h_m) \in \{1, \ldots, n\}^m$ and $V = (v_1, \ldots, v_n) \in \{1, \ldots, m\}^n$, and we want to decide whether there exists a picture which the i-th row projection equals h_i and which j-th column projection equals v_j.

Often, we consider several additional properties like symmetry, connectivity or convexity. In this paper, we consider three properties:

horizontal convex (*h-convex*) — in every row the 1's form an interval,

vertical convex (*v-convex*) — in every column the 1's form an interval, and

connected — the set of 1's is connected with respect to the adjacency relation, where every pixel is adjacent to its two vertical neighbours and to its two horizontal neighbours.

A connected pattern is called a *polyomino*. A pattern is *hv-convex* if it is both h-convex and v-convex.

* Supported by KBN grant 8 T11C 043 19

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 253–260, 2001.

In this paper we solve the problem (Woeginger [5]) whether there exists a polynomial time algorithm that takes as an input a horizontal projection vector $H \in \mathbb{R}_+^m$ and a vertical projection vector $V \in \mathbb{R}_+^n$, and which outputs a polyomino whose projections $H^* \in \{1,\ldots,n\}^m$ and $V^* \in \{1,\ldots,n\}^m$ approximate the vectors H and V, respectively. We consider two notions of "approximate"

(1) every component of (H,V) differs by at most one from the corresponding component of (H^*,V^*), i.e. we select only the nearest positive integers (we call this version *the approximation with the absolute error*), and

(2) for every h_i and v_j it is $|h_i - h_i^*| \leq \log(h_i+1)$ and $|v_j - v_j^*| \leq \log(v_j+1)$ for $i = 1,\ldots,m$ and $j = 1,\ldots,n$ (*the approximation with the logarithmic error*).

The algorithm outputs "NO" if there does not exist a polyomino with approximate projections V and H.

1.2 Known Results

First Ryser [6], and subsequently Chang [2] and Wang [7] studied the existence of a pattern satisfying orthogonal projections (H,V) in the class of sets without any conditions. They showed that the decision problem can be solved in time $O(mn)$. These authors also developed some algorithms that reconstruct the pattern from (H,V).

Woeginger [8] proved that the reconstruction problem in the class of polyominoes is an NP-complete problem. Barcucci, Del Lungo, Nivat, Pinzani [1] showed that the reconstruction problem is also NP-complete in the class of h-convex polyominoes and in the class v-convex polyominoes.

The first algorithm that establishes the existence of an hv-convex polyomino satisfying a pair of assigned vectors (H,V) in polynomial time was described by Barcucci et al. in [1]. Its time complexity is $O(m^4n^4)$ and it is rather slow. Gębala [4] showed the faster version of this algorithm with complexity $O(\min(m^2,n^2)\cdot mn\log mn)$. The latest algorithm described by Chrobak and Dürr in [3] reconstructs the hv-convex polyomino from orthogonal projection in time $O(\min(m^2,n^2)\cdot mn)$.

All above results concern to the reconstruction polyominoes from exact orthogonal projections (RPfOP).

1.3 Our Results

In this paper we study complexity of the problem of reconstruction polyominoes from approximately orthogonal projections (RPfAOP). In section 2 we prove that RPfAOP (for both kinds of errors) is NP-complete in the classes of (1) polyominoes, (2) horizontal convex polyominoes and (3) vertical convex polyominoes. In section 3 we show that RPfAOP, for an arbitrary chosen function of error, is in P for the class of hv-convex polyominoes. We describe an algorithm that solves this problem and has complexity $O(m^3n^3)$.

2 Hardness

In this section we show the reduction from the problem of reconstruction of polyominoes from exact orthogonal projections (RPfOP) to the problem of reconstruction of polyominoes from approximately orthogonal projections. Let

$$\tilde{H} = \{\tilde{h}_1, \ldots, \tilde{h}_m\} \in \{1, \ldots, n\}^m,$$

$$\tilde{V} = \{\tilde{v}_1, \ldots, \tilde{v}_n\} \in \{1, \ldots, m\}^n$$

be an instance of RPfOP problem. Moreover we assume that

$$\sum_{i=1}^{m} \tilde{h}_i = \sum_{j=1}^{n} \tilde{v}_j,$$

otherwise the polyomino with the projections $(\tilde{H}, \tilde{V})$ does not exist. By this instance, we will construct a row vector $H = \{h_1, \ldots, h_m\} \in \mathbb{R}_+^m$ and a column vector $V = \{v_1, \ldots, v_n\} \in \mathbb{R}_+^n$ adequate to the notion of error. For the approximation with the absolute error we fix

$$h_i = \tilde{h}_i - \frac{1}{2}, \qquad i = 1, \ldots, m,$$

$$v_j = \tilde{v}_j + \frac{1}{2}, \qquad j = 1, \ldots, n.$$

And for the logarithmic approximation we choose h_i such that

$$\tilde{h}_i \leq h_i + \log(h_i + 1) < \tilde{h}_i + 1,$$

and v_j such that

$$\tilde{v}_j - 1 < v_j - \log(v_j + 1) \leq \tilde{v}_j.$$

The choice is always possible because functions $x - \log(x+1)$ and $x + \log(x+1)$ are continuous and strictly increasing surjections on $\mathbb{R}_+$.

Now we can solve the RPfAOP problem for projections (H, V).

Lemma 1. *If there exists a polyomino P with row projections $H^* \in \{1, \ldots, n\}^m$ and column projections $V^* \in \{1, \ldots, m\}^n$, such that (H^*, V^*) is the approximation with the absolute (logarithmic) error of (H, V), then there exists a polyomino with projections $(\tilde{H}, \tilde{V})$.*

Proof. For the polyomino P the following properties hold

(i) $\sum_i h_i^* = \sum_j v_j^*$ (the sums are equal to number of 1's in polyomino P), and

(ii) $|h_i^* - h_i| \leq 1$ and $|v_j^* - v_j| \leq 1$ for the absolute error ($|h_i^* - h_i| \leq \log(h_i + 1)$ and $|v_j^* - v_j| \leq \log(v_j + 1)$ for the logarithmic error).

But from the definition of (H, V) the above properties occur if and only if for all i we have that h_i^* is equal to the maximal admissible value, i.e.

$$h_i^* = \lfloor h_i + 1 \rfloor = \tilde{h}_i \qquad (\ h_i^* = \lfloor h_i + \log(h_i + 1) \rfloor = \tilde{h}_i\ ),$$

and for all j we have that v_j^* is equal to the minimal admissible value, i.e.

$$v_j^* = \lceil v_j - 1 \rceil = \tilde{v}_j \qquad (\ v_j^* = \lceil v_j - \log(v_j + 1) \rceil = \tilde{v}_j\ ).$$

Therefore P also satisfies $(\tilde{H}, \tilde{V})$. □

Lemma 2. *If there exists a polyomino P with projections $(\tilde{H}, \tilde{V})$, then there exists a polyomino with row projections $H^* \in \{1, \ldots, n\}^m$ and column projections $V^* \in \{1, \ldots, m\}^n$, such that (H^*, V^*) is the approximation with the absolute (logarithmic) error of (H, V).*

Proof. From definition of (H, V) (for both kinds of approximation) we have that every component of (H, V) can be rounded to the corresponding component of $(\tilde{H}, \tilde{V})$. Hence P is also realisation of (H, V). □

Because we know that RPfOP for polyominoes, h-convex polyominoes and v-convex polyominoes is NP-complete (see [1] and [8]) we obtain from Lemma 1 and Lemma 2 the following result

Theorem 1. *The reconstruction of polyominoes, h-convex polyominoes and v-convex polyominoes from their approximately orthogonal projections is NP-complete.* □

3 hv-Convex Polyomino

In this section we use some ideas and notations from Chrobak and Dürr [3] while describing the algorithm for reconstruction hv-convex polyominoes from approximately orthogonal projection.

In the algorithm described below we generalise the error of approximation and assume that it is in the form of a function f. We assume that the function f is positive on $\mathbb{R}_+$. For example, the absolute error is a constant function equal 1 ($f(x) = 1$) and the logarithmic error is a logarithmic function ($f(x) = \log(x+1)$).

First we define some auxiliary expressions:

$$\check{v}_j = \max\{1, \lceil v_j - f(v_j) \rceil\} \quad \text{and} \quad \hat{v}_j = \min\{m, \lfloor v_j + f(v_j) \rfloor\}$$

for $j = 1, \ldots, n$, and

$$\check{h}_i = \max\{1, \lceil h_i - f(h_i) \rceil\} \quad \text{and} \quad \hat{h}_i = \min\{n, \lfloor h_i + f(h_i) \rfloor\}$$

for $i = 1, \ldots, m$.

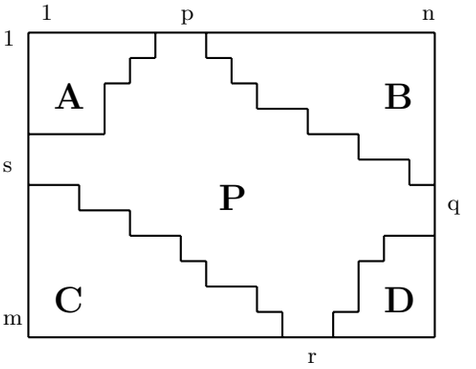


Fig. 1. The convex polyomino P that is anchored at (p, q, r, s), with corner regions A, B, C and D

These expressions have following properties: $\check{h}_i$ ($\check{v}_j$) is the minimal positive integer value and $\hat{h}_i$ ($\hat{v}_j$) is the maximal positive integer value for an horizontal (vertical) projection that differs by at most $f(h_i)$ ($f(v_j)$) from h_i (v_j).

Let (i, j) denote the cell of matrix that is in the i-th row and the j-th column. For an hv-convex polyomino an object A is called an *upper-left corner region* (see Fig.1) if $(i + 1, j) \in A$ or $(i, j + 1) \in A$ implies $(i, j) \in A$. In an analogous way we can define other corner regions. By $\overline{P}$ we denote the complement of P.

From the definition of hv-convex polyominoes we have the following lemma

Lemma 3 (Chrobak and Dürr [3]). *P is an hv-convex polyomino if and only if*

$$P = \overline{A \cup B \cup C \cup D},$$

where A, B, C, D are disjoint corner regions (upper-left, upper-right, lower-left and lower-right, respectively) such that

(i) $(i - 1, j - 1) \in A$ *implies* $(i, j) \notin D$, *and*

(ii) $(i - 1, j + 1) \in B$ *implies* $(i, j) \notin C$. □

We say that the hv-convex polyomino P is anchored at (p, q, r, s) if cells $(1, p), (q, n), (m, r), (s, 1) \in P$ (i.e. these cells do not belong to any corner region).

The main idea of our algorithm is, given (H,V) – vectors of approximately orthogonal projections, to construct a 2SAT expression $F_{p,q,r,s}(H, V)$ with the property that $F_{p,q,r,s}(H, V)$ is satisfiable if and only if there exists an hv-convex polyomino with projections (H^*, V^*) that is anchored at (p, q, r, s) and every component of (H, V) differs by at most the value of function f from the corresponding component of (H^*, V^*).

$F_{p,q,r,s}(H, V)$ consists of several sets of clauses, each set expressing a certain property: "Corners" (Cor), "Connectivity" (Con), "Anchors" (Anc), "Lower bound on column sums" (LBC), "Upper bound on column sums" (UBC), "Lower bound on row sums" (LBR) and "Upper bound on row sums" (UBR).

$$Cor \equiv \bigwedge_{i,j} \left\{ \begin{array}{ll} A_{i,j} \Rightarrow A_{i-1,j} & A_{i,j} \Rightarrow A_{i,j-1} \\ B_{i,j} \Rightarrow B_{i-1,j} & B_{i,j} \Rightarrow B_{i,j+1} \\ C_{i,j} \Rightarrow C_{i+1,j} & C_{i,j} \Rightarrow C_{i,j-1} \\ D_{i,j} \Rightarrow D_{i+1,j} & D_{i,j} \Rightarrow D_{i,j+1} \end{array} \right\}$$

$$Con \equiv \bigwedge_{i,j} \{ A_{i,j} \Rightarrow \overline{D}_{i+1,j+1} \quad B_{i,j} \Rightarrow \overline{C}_{i+1,j-1} \}$$

$$Anc_{p,q,r,s} \equiv \bigwedge \left\{ \begin{array}{llll} \overline{A}_{1,p} & \overline{B}_{1,p} & \overline{C}_{1,p} & \overline{D}_{1,p} \\ \overline{A}_{q,n} & \overline{B}_{q,n} & \overline{C}_{q,n} & \overline{D}_{q,n} \\ \overline{A}_{m,r} & \overline{B}_{m,r} & \overline{C}_{m,r} & \overline{D}_{m,r} \\ \overline{A}_{s,1} & \overline{B}_{s,1} & \overline{C}_{s,1} & \overline{D}_{s,1} \end{array} \right\}$$

$$LBC \equiv \bigwedge_{i,j} \left\{ \begin{array}{ll} A_{i,j} \Rightarrow \overline{C}_{i+\check{v}_j,j} & A_{i,j} \Rightarrow \overline{D}_{i+\check{v}_j,j} \\ B_{i,j} \Rightarrow \overline{C}_{i+\check{v}_j,j} & B_{i,j} \Rightarrow \overline{D}_{i+\check{v}_j,j} \end{array} \right\} \wedge \bigwedge_{j} \left\{ \begin{array}{l} \overline{C}_{\check{v}_j,j} \\ \overline{D}_{\check{v}_j,j} \end{array} \right\}$$

$$UBC_{p,r} \equiv \bigwedge_{i} \left\{ \begin{array}{ll} \bigwedge\limits_{j \leq \min\{p,r\}} & \overline{A}_{i,j} \Rightarrow C_{i+\hat{v}_j,j} \\ \bigwedge\limits_{p \leq j \leq r} & \overline{B}_{i,j} \Rightarrow C_{i+\hat{v}_j,j} \\ \bigwedge\limits_{r \leq j \leq p} & \overline{A}_{i,j} \Rightarrow D_{i+\hat{v}_j,j} \\ \bigwedge\limits_{\max\{p,r\} \leq j} & \overline{B}_{i,j} \Rightarrow D_{i+\hat{v}_j,j} \end{array} \right\}$$

$$LBR \equiv \bigwedge_{i,j} \left\{ \begin{array}{ll} A_{i,j} \Rightarrow \overline{B}_{i,j+\check{h}_i} & A_{i,j} \Rightarrow \overline{D}_{i,j+\check{h}_i} \\ C_{i,j} \Rightarrow \overline{B}_{i,j+\check{h}_i} & C_{i,j} \Rightarrow \overline{D}_{i,j+\check{h}_i} \end{array} \right\} \wedge \bigwedge_{i} \left\{ \begin{array}{l} \overline{B}_{i,\check{h}_i} \\ \overline{D}_{\check{h}_i} \end{array} \right\}$$

$$UBR_{s,q} \equiv \bigwedge_{j} \left\{ \begin{array}{ll} \bigwedge\limits_{i \leq \min\{s,q\}} & \overline{A}_{i,j} \Rightarrow B_{i,j+\hat{h}_i} \\ \bigwedge\limits_{s \leq i \leq q} & \overline{C}_{i,j} \Rightarrow B_{i,j+\hat{h}_i} \\ \bigwedge\limits_{q \leq j \leq s} & \overline{A}_{i,j} \Rightarrow D_{i,j+\hat{h}_i} \\ \bigwedge\limits_{\max\{s,q\} \leq j} & \overline{C}_{i,j} \Rightarrow D_{i,j+\hat{h}_i} \end{array} \right\}$$

(LBC) assigns the minimal distance between corner regions for columns (for j-th column it is equal to $\check{v}_j$). (UBC) assigns the maximal distance between corner regions for columns (for j-th column it is equal to $\hat{v}_j$). (LBR) and (UBR) are analogous for rows. Now we define a 2SAT formula

$$F_{p,q,r,s}(H,V) = Cor \wedge Con \wedge Anc_{p,q,r,s} \wedge LBC \wedge UBC_{p,r} \wedge LBR \wedge UBR_{q,s}.$$

All literals with indices outside the set $\{1,\ldots,m\} \times \{1,\ldots,n\}$ are assumed to have value 1.

Now we give the algorithm of reconstruction.

Algorithm

Input: $H \in \mathbb{R}_+^m$, $V \in \mathbb{R}_+^n$

FOR $p, r = 1,\ldots,n$ AND $q, s = 1,\ldots,m$ DO
  IF $F_{p,q,r,s}(H,V)$ is satisfiable
  THEN RETURN $P = \overline{A \cup B \cup C \cup D}$ AND HALT
RETURN "NO"

Theorem 2. *$F_{p,q,r,s}(H,V)$ is satisfiable if and only if P is an hv-convex polyomino with projections (H^*,V^*) that is anchored at (p,q,r,s) and every component of (H^*,V^*) differs from the correspondent component of (H,V) by at most the value of function f for this component.*

Proof. ($\Leftarrow$) If P is an hv-convex polyomino with properties like in the theorem, then let A, B, C, D be the corner regions from Lemma 3. By Lemma 3, A, B, C, D satisfy conditions (Cor) and (Con). Condition (Anc) is true because P is anchored at (p,q,r,s). Moreover for all $i = 1,\ldots,m$ we have $|h_i^* - h_i| \leq f(h_i)$ and $h_i^* \in \mathbb{N}$, hence $\check{h}_i \leq h_i^* \leq \hat{h}_i$ and conditions (LBR) and (UBR) hold. Analogous, conditions (LBC) and (UBC) hold for vertical projections.

($\Rightarrow$) If $F_{p,q,r,s}(H,V)$ is satisfiable, take $P = \overline{A \cup B \cup C \cup D}$. Conditions (Cor), (Con), (LBC) and (LBR) imply that the sets A, B, C, D satisfy Lemma 3 ((LBC) and (LBR) guarantee disjointness of A, B, C, D), and thus P is an hv-convex polyomino. Also, by (Anc), P is anchored at (p,q,r,s). Conditions (LBR) and (UBR) imply that $\check{h}_i \leq h_i^* \leq \hat{h}_i$ for each row i. Hence $|h_i^* - h_i| \leq f(h_i)$. Analogous, conditions (LBC) and (UBC) imply that $\check{v}_j \leq v_j^* \leq \hat{v}_j$ for each column j and therefore $|v_j^* - v_j| \leq f(v_j)$. Moreover because P is the finite set we have $\sum_i h_i^* = \sum_j v_j^*$. Hence P must be an hv-convex polyomino with approximately orthogonal projections (H,V) with respect to function f. □

Each formula $F_{p,q,r,s}(H,V)$ has size $O(mn)$ and can be computed in the linear time. Since a 2SAT formula can also be solved in the linear time, we obtain the following result

Theorem 3. *The problem of reconstruction of hv-convex polyominoes from approximately orthogonal projections can be solved in time $O(m^3n^3)$.* □

References

1. Barcucci, E., Del Lungo, A., Nivat, M., Pinzani, R.: Reconstructing Convex Polyominoes from Horizontal and Vertical Projections. Theor. Comp. Sci. 155 (1996) 321–347
2. Chang, S.K.: The Reconstruction of Binary Patterns from their Projections. Comm. ACM 14 (1971) 21–25
3. Chrobak, M., Dürr, Ch.: Reconstructing hv-convex Polyominoes from Orthogonal Projections. Information Processing Letters vol. 69(6) (1999) 283–289
4. Gębala, M.: The Reconstruction of Convex Polyominoes from Horizontal and Vertical Projections. LNCS vol. 1521, Springer-Verlag (1998) 350–359
5. Gritzmann, P., Nivat, M. (editors): Discrete Tomography: Algorithms and Complexity. Dagstuhl-Seminar-Report; 165 (1997)
6. Ryser, H.: Combinatorial Mathematics. The Carus Mathematical Monographs Vol. 14 (The Mathematical Association of America, Rahway, 1963)
7. Wang, X.G.: Characterisation of Binary Patterns and their Projections. IEEE Trans. Comput. C-24 (1975) 1032–1035
8. Woeginger, G.J.: The Reconstruction of Polyominoes from Their Orthogonal Projections. Technical report SFB-65, TU Graz, Austria, (1996)

Bounding Lamport's Bakery Algorithm

Prasad Jayanti, King Tan, Gregory Friedland, and Amir Katz

6211 Sudikoff Lab for Computer Science
Dartmouth College, Hanover, NH 03755
prasad@cs.dartmouth.edu
kytan@cs.dartmouth.edu

Abstract. Lamport's Bakery algorithm is among the best known mutual exclusion algorithms. A drawback of Lamport's algorithm is that it requires unbounded registers for communication among processes. By making a small modification to Lamport's algorithm, we remove the need for unbounded registers. The main appeal of our algorithm lies in the fact that it overcomes a drawback of a famous algorithm while preserving its elegance.

1 Introduction

Mutual Exclusion is a classic synchronization problem that has been extensively studied (see [5] for a survey of mutual exclusion algorithms). This problem is described as follows. There are n asynchronous processes with each process repeatedly performing four sections of code: remainder section, entry section, critical section and exit section. It is assumed that no process fails in the entry or exit sections and every process that enters the critical section eventually leaves it. The problem is to design a protocol for entry and exit sections that satisfies the following properties:

Mutual Exclusion: No two processes are in the critical section at the same time.
Starvation freedom: Each process in the entry section eventually enters the critical section.
Wait-free exit: Each process can complete the exit section in a bounded number of steps, regardless of the speeds of other processes.

The following fairness property is also desirable in any mutual exclusion protocol:

Doorway FIFO: The entry section begins with a straight line code called the *wait-free doorway*: a process can execute the doorway in a bounded number of steps, regardless of the speeds of other processes. If process P_i completes executing the doorway before process P_j begins executing the doorway, P_i enters the critical section before P_j.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 261–270, 2001.

Lamport's Bakery algorithm is among the best known mutual exclusion algorithms [2]. It is discussed in introductory Operating Systems textbooks [6] and in books on Concurrent Algorithms [1,4,5] and is well-known to the general Computer Science community. The algorithm satisfies all of the above properties, is elegant and has an intuitive appeal: processes assign themselves tokens in the doorway, and enter the critical section in the order of their token numbers. One drawback of Lamport's algorithm is that it requires unbounded registers for communication among processes. It is this drawback that our paper overcomes: with a simple modification, we show how Lamport's Bakery algorithm can be made to work with small bounded registers. Our algorithm adds just two lines and makes a small change to an existing line. Lamport's algorithm and our algorithm are presented in Figures 1 and 2, respectively. (Lines 3 and 8 in Figure 2 are new and line 4 is a slightly modified version of the corresponding line in Figure 1.)

Our algorithm compares with Lamport's algorithm as follows:

Properties: Our algorithm, like Lamport's algorithm, has all properties stated above, including the doorway FIFO property.
Size of registers: Lamport's algorithm requires unbounded registers while ours requires bounded registers, each of size either 1 bit or $\log 2n$ bits. This is the highlight of our algorithm.
Type of registers: Lamport's algorithm requires $2n$ single-writer multi-reader registers. Our algorithm requires the corresponding $2n$ single-writer multi-reader registers and an additional register that we call X. The register X is written by the process in the critical section. Therefore, there is at most one write operation on X at any time. Yet, strictly speaking, it is not a single-writer register because different processes write to it (although never concurrently).
Lamport's algorithm works even if registers are only safe, but our algorithm requires atomic registers.

Algorithms with better space complexity than ours are known: to our knowledge, the best algorithm, from the point of view of size of registers, is due to Lycklama and Hadzilacos [3]. Their algorithm has all properties stated above and requires only $5n$ safe boolean registers. Thus, the appeal of our algorithm is not due to its low space complexity, but because it shows that a small modification removes a well-known drawback of a famous algorithm while preserving its elegance.

1.1 Organization

Lamport's Bakery algorithm (Figure 1), which employs unbounded token values, is the starting point of our work and is described in Section 2. Our approach to bounding token values requires these values to be clustered. In Section 3, we show that Lamport's algorithm does not have the clustering property. In Section 4, we present a modification of Lamport's algorithm that continues to use unbounded tokens, but has the clustering property. Then, in Section 5, we present a bounded version of this algorithm.

2 Lamport's Algorithm

[initialize $\forall i : 1 \leq i \leq n : (gettoken_i :=$ `false`
$token_i := -1)$]

1. $gettoken_i :=$ `true`
2. $S := \{token_j | 1 \leq j \leq n\}$
3. $token_i := max(S) + 1$
4. $gettoken_i :=$ `false`
   for $j \in \{1 \cdots n\} - \{i\}$
5. **wait till** $gettoken_j =$ `false`
6. **wait till** $(token_j = -1) \vee ([token_i, i)] < [token_j, j])$
7. CS
8. $token_i := -1$

Fig. 1. Lamport's Bakery algorithm : P_i's protocol

We first describe Lamport's Bakery algorithm (Figure 1). The algorithm is based on the following idea. When a process gets into the entry section, it assigns itself a token that is bigger than the existing tokens. It then waits for its turn, i.e., until every process with a smaller token has exited the critical section (CS).

Each process P_i maintains two variables—$token_i$ and $gettoken_i$. The purpose of $token_i$ is for P_i to store its token number. A value of -1 in $token_i$ indicates that P_i is either in the remainder section or in the process of assigning itself a token. The variable $gettoken_i$ is a boolean flag. P_i sets it to *true* when it is in the process of assigning itself a token.

We now informally describe the actual lines of the algorithm. P_i sets $gettoken_i$ to *true* to signal that it is in the process of assigning itself a token (line 1). It then reads the tokens held by all other processes (line 2) and assigns itself a bigger token (line 3). Notice that if another process P_j is also computing a token concurrently with P_i, both P_i and P_j may end up with the same token number. Therefore, when comparing tokens (line 6), if the token values of two processes are equal, the process ID is used to decide which token should be considered smaller. Having set its token, P_i sets the flag $gettoken_i$ to *false* (line 4). Lines 1-4 constitute the doorway.

P_i then considers each process P_j in turn and waits until it has "higher priority" than P_j, i.e., until it is certain that P_j does not have (and will not later have) a smaller token than P_i's. This is implemented on lines 5 and 6, and the intuition is described in the rest of this paragraph. Essentially, P_i needs to learn P_j's token value to determine their relative priority. P_i can learn P_j's token value by reading $token_j$ except in one case: P_i cannot rely on $token_j$ when P_j is in the process of updating it (i.e., P_j is on lines 2 or 3). This is because, when P_j is on lines 2 or 3, even though the value of $token_j$ is -1, P_j may be about to write into $token_j$ a smaller value than P_i's token.

Specifically, for each P_j, P_i first waits until P_j's flag ($gettoken_j$) is false (line 5), i.e., until P_j is not in the process of assigning itself a token. After this wait, P_i can be certain of the following fact: if P_j assigns itself a new token before P_i has exited CS, P_j's token will be bigger than P_i's token since (on line 2) P_j will surely read $token_i$. P_i then waits until either P_j is in the remainder section or P_j's token is bigger than its token (line 6). At the end of this wait, P_i can be certain that it has a higher priority than P_j: if P_j is in the entry section or will enter the entry section, P_j's token will be bigger than P_i's and hence P_j will be forced to wait at line 6 until P_i has exited CS. Thus, when the for-loop terminates, P_i is certain that it has higher priority than all other processes. So it enters CS (line 7). When it exits CS, it sets $token_i$ to -1 to indicate that is in the remainder section.

3 Our Approach to Bounding Token Values

In Lamport's algorithm, let max_t and min_t denote the maximum and minimum non-negative token values at time t (they are undefined if all tokens are -1). Let $range_t$ be $max_t - min_t$. Lamport's algorithm is an unbounded algorithm because max_t can increase without bound. What is more pertinent for our purpose of bounding tokens, however, is the fact that $range_t$ can also increase without bound. In Section 3.1 we substantiate this claim that $range_t$ can grow unbounded. Our approach to deriving a bounded algorithm depends on limiting the growth of $range_t$. This approach is described in Section 3.2.

3.1 Unbounded Separation of Tokens in Lamport's Algorithm

Below we describe a scenario to show that the value of $range_t$ can increase without bound. Consider a system of two processes P_1, P_2 in the following state: Both P_1 and P_2 have just completed executing line 4. The values of $token_1$, $token_2$ are respectively either 0, v (for some $v \geq 0$) or v, 0. The range is therefore v.

- P_1 observes that $gettoken_2 = false$ (line 5). P_2 observes that $gettoken_1 = false$ (line 5).
- If $[token_1, 1] < [token_2, 2]$, let $P_{i_1} = P_1$,$P_{i_2} = P_2$. Otherwise, let $P_{i_1} = P_2$,$P_{i_2} = P_1$. Note that the value of $token_{i_2}$ is v. P_{i_1} executes line 6, enters and exits CS, and writes -1 in $token_{i_1}$. P_{i_1} then begins a new invocation of the protocol. P_{i_1} reads v in $token_{i_2}$, computes $v+1$ as the new value for $token_{i_1}$. P_{i_1} stops just before writing $v+1$ in $token_{i_1}$.
- P_{i_2} executes line 6, enters and exits CS, and writes -1 in $token_{i_2}$. P_{i_2} then begins a new invocation of the protocol. P_{i_2} reads -1 in $token_{i_1}$, and writes 0 in $token_{i_2}$. P_{i_2} then executes line 4.
- P_{i_1} writes $v+1$ in $token_{i_1}$ and executes line 4. The range is now $v+1$. The system state now is identical to the system state at the beginning of the scenario, except that the range has increased by 1.

The above scenario took us from a system state with range v to a system state with range $v + 1$. This scenario can be repeated arbitrarily many times, causing the range to increase without bound.

3.2 The Main Idea

Our approach to bounding the token values in Lamport's algorithm consists of two steps. In the first step, we introduce a mechanism that forces the token values to form a narrow cluster, thereby ensuring a small bounded value for $range_t$. The resulting algorithm, which we call UB-Bakery, still uses unbounded tokens, but the tokens are clustered. In the second step, we observe that the token clustering makes it possible to replace integer arithmetic with modulo arithmetic. The resulting algorithm, which we call B-Bakery, achieves our goal of using small bounded token values. The first step is described section 4 and the second step in Section 5.

4 Algorithm with Unbounded and Clustered Tokens

To implement the first step described above, we introduce a new shared variable X which stores the token value of the latest process to visit CS. The new algorithm, UB-Bakery, is in Figure 2. It makes two simple modifications to Lamport's algorithm. First, a process writes its token value in X (line 8) immediately before it enters CS. The second modification is on lines 3 and 4. In addition to reading the tokens of all the processes, P_i also reads X (line 3). P_i then computes its new token value as 1+ (maximum of the values of all processes' tokens and X). No other changes to Lamport's algorithm are needed.

To see the usefulness of X, let us revisit the scenario described in the previous section. There, P_{i_1} read the value v in $token_{i_2}$, but P_{i_2} read -1 in $token_{i_1}$. Consequently, while P_{i_1} set its token to $v+1$, P_{i_2} set its token to 0. This possibility, of one process adopting a large token value and the other a small token value, is what causes the range to increase without bound. In the new algorithm, this is prevented because, even though P_{i_2} might read -1 in $token_{i_1}$, it would find v in X. As a result, P_{i_2}'s token value would be $v + 1$ also. More importantly, the token values of P_{i_1} and P_{i_2} would be close to each other (in this scenario, they would be the same).

Intuitively, the new algorithm ensures that X grows monotonically and all non-negative token values cluster around X. The exact nature of this clustering is stated and proved in the next subsection.

4.1 Properties of UB-Bakery

We now state the desirable clustering properties of UB-Bakery (Theorems 2 and 3), and give proof outlines of how they are achieved. Rigorous proofs of these properties are provided in the full version of this paper.

Observation 1 *The value of X is non-decreasing.*

```
[initialize X := 0;
   ∀i : 1 ≤ i ≤ n : (gettoken_i := false
      token_i := −1) ]

1.    gettoken_i := true
2.    S := {token_j | 1 ≤ j ≤ n}
3.    x := X
4.    token_i := max(S ∪ {x}) + 1
5.    gettoken_i := false
      for j ∈ {1 ··· n} − {i}
6.       wait till gettoken_j = false
7.       wait till (token_j = −1) ∨ ([token_i, i)] < [token_j, j])
8.    X := token_i
9.    CS
10.   token_i := −1
```

Fig. 2. Algorithms UB-Bakery and B-Bakery : P_i's protocol

Proof Sketch: Suppose the value of X decreases. Specifically, let $x_i, x_j, (x_i > x_j)$ be two consecutive values of X, with x_j immediately following x_i. Let P_i, P_j be the processes that write x_i, x_j in X respectively. Thus, P_j immediately follows P_i in the order of their entering CS.

If P_j reads $token_i$ (line 2) after P_i has written x_i in $token_i$ (line 4), then P_j either reads x_i in $token_i$ (line 2) or reads x_i in X (line 3). Thus, x_j, the value of $token_j$ computed by P_j (line 4), must be greater than x_i. This contradicts our assumption that $x_i > x_j$. Therefore, P_j reads $token_i$ (line 2) before P_i writes x_i in $token_i$ (line 4). P_i must subsequently wait until $gettoken_j = false$ (line 6). Since P_j must first write x_j in $token_j$ before setting $gettoken_j$ to *false*, P_i will read x_j in $token_j$ when it executes line 7. Since $x_i > x_j$, P_i will not exit its waiting loop w.r.t. P_j (line 7) until P_j has exited CS and set $token_j$ to -1. This contradicts our assumption that P_i enters CS before P_j. This completes the proof of Observation 1.

□

Theorem 1 (Bounded token range). *At any time t and for any j, let v be the value of $token_j$ and x be the value of X. If $v \neq -1$, then $x \leq v \leq x + n$.*

Proof Sketch: This Theorem asserts that at any time t, the non-negative token values lie in the range $[x, x + n]$, where x is the value of X at t. We now give an informal proof of this Theorem. Suppose Theorem 1 is false. If $v < x$, then P_j has not entered CS at time t. Suppose P_j subsequently writes v in X at time t' (prior to entering CS). The value of X decreases at some time in $[t, t']$ (because $v < x$). This violates Observation 1. If $v > x + n$, then there is some integer a, where $x + 1 \leq a \leq x + n$, such that a is not the token value of any process at t. (This is true for the following reason: Excluding v, there are at

most $n-1$ distinct token values at t, whereas there are n integers in the interval $[x+1, x+n]$.) Since v is the value of $token_j$ at t, and $v > a$, then a must have been the token value of some process P_k at some time before t, This implies that P_k has written a in X at some time t' before t. Since $a > x$, the value of X decreases at some time in the interval $[t', t]$. This again violates Observation 1. This completes the proof.

□

Theorem 2 (Token cluster around X). *Let v be the value of $token_j$ read by P_i executing line 2. Let x be the value of X read by P_i executing line 3. Then, $(v \neq -1) \Rightarrow x-(n-1) \leq v \leq x+(n-1)$.*

Proof Sketch: This Theorem states that any token value v read by P_i on line 2 lies within the range $[x-(n-1), x+(n-1)]$, where x is the value of X read by P_i on line 3. We say that the token values v read on line 2 *cluster* around the *pivot* x. To establish Theorem 2, we first prove two Observations:

Observation 2 *Let t_1 be the time P_i executes line 1 and t_2 be the time P_i executes line 3. Then, any process P_j executes line 8 (writing the value of $token_j$ in X) at most once in (t_1, t_2).*

Proof Sketch: This observation is true for the following reason: Suppose P_j executes line 8 for the first time in (t_1, t_2), and then begins a new invocation of UB-Bakery. Within the interval (t_1, t_2), P_j cannot proceed beyond line 6 of the invocation, where P_j waits for $gettoken_i$ to become false (because $gettoken_i = true$ throughout (t_1, t_2)).

□

Observation 3 *Let t_1 be the time P_i executes line 1 and t_2 be the time P_i executes line 3. Let the value of $token_j$ be v_1 and v_2 at t_1 and t_2, respectively. Let v be the value that P_i reads in $token_j$ on line 2. If $v \neq -1$, then either $v = v_1$ or $v = v_2$.*

Proof Sketch: Suppose $v \neq v_1$. Then P_j must have written v in $token_j$ at some time t, where $t_1 < t < t_2$. Consider the invocation by P_j during which P_j writes v in $token_j$. P_j cannot exit its waiting loop on line 6 w.r.t. P_i at any time before t_2 (because $gettoken_i = true$ throughout (t_1, t_2)). Therefore the value of $token_j$ is v at t_2, i.e. $v = v_2$.

□

Now we continue with the proof sketch of Theorem 2. Theorem 1 establishes that the token range is bounded at any time. Further, the token values are bounded below by the value of X. Observation 2 implies that the value of X increases by no more than $(n-1)$ in (t_1, t_2). Observation 3 says that any non-negative value of $token_j$ read by P_i on line 2 is the value of $token_j$ at either t_1 or t_2. Manipulating the inequalities that result from these assertions gives Theorem 2.

□

Theorem 3 (Token cluster around $token_i$). *Let v_i be the value of $token_i$ (written by P_i on line 4) when P_i is executing line 7. Let v_j be the value of $token_j$ read by P_i when executing line 7. If $v_j \neq -1$, then $v_i-(n-1) \leq v_j \leq v_i+(n-1)$.*

Proof Sketch: This Theorem states that all non-negative token values read by P_i on line 7 cluster around the pivot $token_i$ (which is unchanged throughout the interval during which P_i executes line 7). The proof proceeds as follows: Suppose P_i reads $token_j$ (line 7) at time t. By contradiction, suppose $v_j < v_i - (n-1)$ (resp. $v_i < v_j - (n-1)$) at time t. Then, there is some integer a, $v_j < a < v_i$ (resp. $v_i < a < v_j$), such that a is not the token value of any process at t. (This is true for the following reason: There are at most n distinct token values, whereas there are more than n integers in the interval $[v_j, v_i]$ (resp. $[v_i, v_j]$).) Since v_i (resp. v_j) is the value of $token_i$ (resp. $token_j$) at t, and $v_i > a$ (resp. $v_j > a$), then a must have been the token value of some process P_k at some time before t. Therefore the value of X was a at some time t' before t. Let x be the value of X at t. By Theorem 1, $x \leq v_j$ (resp. $x \leq v_i$). This implies that $x < a$. This in turn implies that the value of X decreases at some time in $[t', t]$, which contradicts Observation 1. This completes the proof.

□

5 Algorithm with Bounded Tokens

In the algorithm UB-Bakery, the values in X and $token_i$ increase without bound. However, by exploiting token clustering (Theorems 2 and 3), it is possible to bound these values. This is the topic of this section.

The main idea is that it is sufficient to maintain the value of X and the non-negative values of $token_i$ modulo $2n-1$. Specifically, let B-Bakery denote the Bounded Bakery algorithm whose text is identical to UB-Bakery (Figure 2), except that the addition operation on line 4 is replaced with *addition modulo* $2n-1$, denoted by $\oplus$, and the operators *max* and $<$ (on lines 4 and 7, respectively) are replaced with max and $\prec$ respectively (these new operators will be defined shortly). Thus, in B-Bakery, we have $X \in \{0, 1, \ldots, 2n-2\}$ and $token_i \in \{-1, 0, 1, \ldots, 2n-2\}$.

Define a function f that maps values that arise in UB-Bakery algorithm to values that arise in B-Bakery algorithm as follows: $f(-1) = -1$ and for all $v \geq 0$, $f(v) = v \, mod \, (2n-1)$. For a set S, define $f(S)$ as $\{f(v) \mid v \in S\}$.

To define the operators max and $\prec$ for B-Bakery, we consider two runs: a run R in which processes execute UB-Bakery algorithm and a run R' in which processes execute B-Bakery algorithm. It is assumed that the order in which processes take steps is the same in R and in R'. Our goal is to define max and $\prec$ for B-Bakery so that processes behave "analogously" in R and R'. Informally this means that the state of each P_i at any point in run R is the same as P_i's state at the corresponding point in R'; and the value of each shared variable at any point in R is congruent (modulo $2n-1$) to its value at the corresponding point in R'. We realize this goal as follows.

– Definition of max: Consider a process P_i that executes lines 2 and 3 in the run R of UB-Bakery algorithm. Let S_{ub} denote the set of token values that P_i reads on line 2 and x_{ub} denote the value of X that P_i reads on line 3. Let $max_{ub} = max\,(S_{ub} \cup \{x_{ub}\})$.
Now consider P_i performing the corresponding steps in run R' of B-Bakery. Let S_b denote the set of token values that P_i reads on line 2 and x_b denote the value of X that P_i reads on line 3.
If processes behaved analogously in runs R and R' so far, we would have $x_b = f(x_{ub})$ and $S_b = f(S_{ub})$. Further, since non-negative values in S_{ub} are in the interval $[x_{ub} - (n-1), x_{ub} + (n-1)]$ (by Theorem 2), if a and b are distinct values in S_{ub}, then $f(a)$ and $f(b)$ would be distinct values in S_b. We want to define the operator max so that $\mathsf{max}(S_b \cup \{x_b\})$ that P_i computes on line 4 is $f(max_{ub})$. This requirement is met by the definition of max described in the next paragraph. (Let $\oplus$ and $\ominus$ be defined as follows: $a \oplus b = (a+b)\,mod\,(2n-1)$ and $a \ominus b = (a-b)\,mod\,(2n-1)$.)
Non-negative values in S_{ub} lie in the interval $[x_{ub}-(n-1), x_{ub}+(n-1)]$. The elements in this interval are ordered naturally as $x_{ub}-(n-1) < x_{ub}-(n-2) < \ldots < x_{ub} < \ldots < x_{ub}+(n-2) < x_{ub}+(n-1)$. Correspondingly, non-negative values in S_b should be ordered according to $x_b \ominus (n-1) < x_b \ominus (n-2) < \ldots < x_b < \ldots < x_b \oplus (n-2) < x_b \oplus (n-1)$. Therefore, if we shift all non-negative values in $S_b \cup \{x_b\}$ by adding to them $n-1-x_b$ (modulo $2n-1$), then the smallest possible value $x_b \ominus (n-1)$ shifts to 0 and the largest possible value $x_b \oplus (n-1)$ shifts to $2n-2$. Then, we can take the ordinary maximum over the shifted values of $S_b \cup \{x_b\}$ and then shift that maximum back to its original value. More precisely, let max be the ordinary maximum over a set of integers. Let $T = S_b \cup \{x_b\} - \{-1\}$. Then, we define $\mathsf{max}(S_b \cup \{x_b\}) = max\,(\{v \oplus (n-1-x_b) \mid v \in T\}) \ominus (n-1-x_b)$. The following Lemma states the desired relationship between max and max that we have established.

Lemma 1. *Consider a run in which processes, including P_i, execute* UB-Bakery. *Let S_{ub} denote the set of token values that P_i reads on line 2 and x_{ub} denote the value of X that P_i reads on line 3. Let $S_b = f(S_{ub})$ and $x_b = f(x_{ub})$. Then $\mathsf{max}(S_b \cup \{x_b\}) = f(max\,(S_{ub} \cup \{x_{ub}\}))$.*

– Definition of $\prec$: Consider a process P_i executing line 7 in run R of UB-Bakery algorithm. Let v_j be the value that P_i reads in $token_j$ and v_i be the value of $token_i$.
Now consider the corresponding step of P_i in the run R' of B-Bakery. Let v'_j be the value that P_i reads in $token_j$ and v'_i be the value of $token_i$.
If processes behaved analogously in runs R and R' so far, we would have $v'_i = f(v_i)$ and $v'_j = f(v_j)$. By Theorem 3, if $v_j \neq -1$ then v_j is in the interval $[v_i-(n-1), v_i+(n-1)]$. We want to define $\prec$ so that $[v'_i, i] \prec [v'_j, j]$ holds if and only if $[v_i, i] < [v_j, j]$ holds. To achieve this, we proceed as before by shifting both v'_i and v'_j by adding $(n-1-v'_i)$ (modulo $2n-1$) and then comparing the shifted values using the ordinary "less than" relation for

integers. More precisely, $\prec$ is defined as follows: $[v'_i, i] \prec [v'_j, j]$ if and only if $[v'_i \oplus (n-1-v'_i), i] < [v'_j \oplus (n-1-v'_i), j]$. The following Lemma states the desired relationship between $\prec$ and $<$ that we have established.

Lemma 2. *Consider a run in which processes, including P_i, execute* UB-Bakery. *Let v_j be the value that P_i reads (on line 7) in $token_j$ and v_i be the value of $token_i$ when P_i is executing line 7. Let $v'_i = f(v_i)$ and $v'_j = f(v_j)$. Then $[v'_i, i] \prec [v'_j, j]$ holds if and only if $[v_i, i] < [v_j, j]$ holds.*

To summarize, B-Bakery, our final algorithm that uses bounded tokens, is identical to UB-Bakery (Figure 2) with the operators $+$, *max* and $<$ replaced with $\oplus$, max and $\prec$ as defined above. The following theorem states our result. Formal proof of this theorem is presented in the full version of this paper.

Theorem 4. *The algorithm* B-Bakery *satisfies the following properties:*

- **Mutual Exclusion:** *No two processes can be simultaneously in CS.*
- **Starvation Freedom:** *Each process that invokes* B-Bakery *eventually enters CS.*
- **Doorway FIFO :** *Let t be the time when P_i writes in $token_i$ when executing line 4. If P_j initiates an invocation after t, then P_i enters CS before P_j.*
- **Bounded Registers:** *Every shared register is either a boolean or has* $\log 2n$ *bits.*

References

1. Attiya, H., and Welch, J.: *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* McGraw-Hill Publishing Company, May 1998.
2. Lamport, L.: A new solution of Dijkstra's concurrent programming problem. Communications of the ACM **17,8** (August 1974) 453-455.
3. Lycklama, E., and Hadzilacos, V.: A first-come-first-served mutual-exclusion algorithm with small communication variables. ACM Transactions on Programming Languages and Systems **13,4** (October 1991) 558-576.
4. Lynch, N.: *Distributed Algorithms.* Morgan Kaufmann Publishers 1996.
5. Raynal, M.: *Algorithms for Mutual Exclusion.* The MIT Press 1986.
6. Silberschatz, A., Peterson, J., and Galvin, P.: *Operating System Concepts.* Addison-Wesley Publishing Company 1991.

Fast Independent Component Analysis in Kernel Feature Spaces

András Kocsor and János Csirik

Research Group on Artificial Intelligence
of the Hungarian Academy of Sciences and the University of Szeged
H-6720 Szeged, Aradi vértanúk tere 1., Hungary
{kocsor, csirik}@inf.u-szeged.hu
http://www.inf.u-szeged.hu/speech

Abstract. It is common practice to apply linear or nonlinear feature extraction methods before classification. Usually linear methods are faster and simpler than nonlinear ones but an idea successfully employed in the nonlinearization of Support Vector Machines permits a simple and effective extension of several statistical methods to their nonlinear counterparts. In this paper we follow this general nonlinearization approach in the context of Independent Component Analysis, which is a general purpose statistical method for blind source separation and feature extraction. In addition, nonlinearized formulae are furnished along with an illustration of the usefulness of the proposed method as an unsupervised feature extractor for the classification of Hungarian phonemes.

KeyWords. feature extraction, kernel methods, Independent Component Analysis, FastICA, phoneme classification

1 Introduction

Feature extraction methods, whether in linear or a nonlinear form, produce preprocessing transformations of high dimensional input data, which may increase the overall performance of classifiers in many real world applications. These algorithms also permit the restriction of the entire input space to a subspace of lower dimensionality. In general, experience has shown that dimensionality reduction has a favorable effect on the classification performance, i.e. reducing superfluous features which can disturb the goal of separation.

In this study Independent Component Analysis will be derived in a nonlinearized form, where the method of nonlinearization was performed by employing the so-called "kernel-idea" [11]. This notion can be traced back to the potential function method [1], and its renewed use in the ubiquitous Support Vector Machine [4], [21].

Without loss of generality we shall assume that as a realization of multivariate random variables, there are m-dimensional real attribute vectors in a compact set $\mathcal{X}$ over $\mathbb{R}^m$ describing objects in a certain domain, and that we have a finite $m \times n$ sample matrix $X = [\mathbf{x_1}, \ldots, \mathbf{x_n}]$ containing n random observations.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 271–281, 2001.

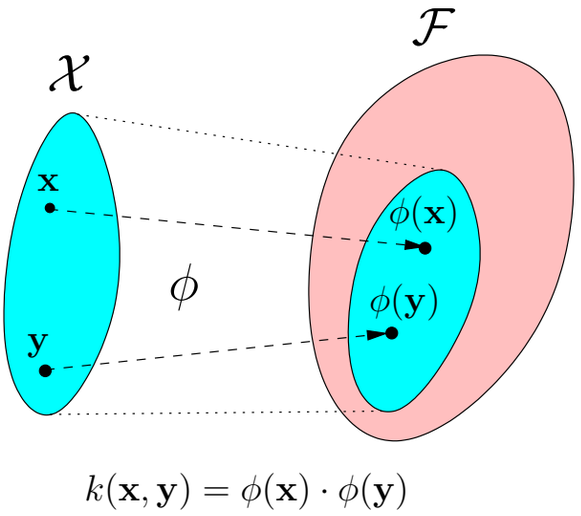


Fig. 1. The "kernel-idea". $\mathcal{F}$ is the closure of the linear span of the mapped data. The dot product in the kernel feature space $\mathcal{F}$ is defined implicitly. The dot product of $\sum_{i=1}^{n} \alpha_i \phi(\mathbf{x_i})$ and $\sum_{i=1}^{n} \beta_i \phi(\mathbf{x_i})$ is $\sum_{i,j} \alpha_i \beta_j k(\mathbf{x_i}, \mathbf{x_j})$.

The aim of Independent Component Analysis (ICA) is to linearly transform the sample matrix X into components that are as independent as possible. The definition of independence of the components can be viewed in different ways. In [8] and [9] Hyvärinen proposed a new concept and a new method (i.e. FastICA) that extends Comon's information theoretic ICA approach [6] with a new family of contrast functions. FastICA is a fast approximate Newton iteration procedure (the convergence is at least quadratic) for the optimization of the negentropy approximant (see definition later), which serves as a measure for selecting new independent components. Fortunately this method can be reexpressed as its input is $K = X^\top X$ instead of X, where the $n \times n$ symmetric matrix K is the pairwise combination of dot products of the sample ($K = [\mathbf{x_i} \cdot \mathbf{x_j}]_{ij}$). Now let the dot product be implicitly defined (Fig. 1) by the kernel function k in some finite or infinite dimensional feature space $\mathcal{F}$ with associated transformation ϕ:

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}). \tag{1}$$

Going the other way, constructing an appropriate kernel function (i.e. where such a function ϕ exists) is a non-trivial problem, but there are many good suggestions about the sorts of kernel functions which might be adopted along with some background theory [21], [5]. However, the two most popular kernels are the following:

$$\text{Polynomial kernel:} \quad k_1(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d, \qquad d \in \mathbb{N}, \tag{2}$$

$$\text{Gaussian RBF kernel:} \quad k_2(\mathbf{x}, \mathbf{y}) = \exp\left(-||\mathbf{x} - \mathbf{y}||^2 / r\right), \qquad r \in \mathbb{R}_+. \tag{3}$$

For a given kernel function the pair $(\phi, \dim \mathcal{F})$ is not always unique and for the kernels k_1 and k_2 the following statements hold:

i) If the dot product is computed as a polynomial kernel, then the dimension of the feature space is at least $\binom{m+d-1}{d}$.

ii) The dot product using the Gaussian RBF kernel induces infinite dimension feature spaces.

As the input of FastICA is represented only by dot products, matrix K is easily redefinable by

$$K = [k(\mathbf{x_i}, \mathbf{x_j})]_{ij}. \tag{4}$$

With this substitution FastICA produces a linear transformation matrix in the kernel feature space $\mathcal{F}$, but now this is no longer a linear transformation of the input data owing to the nonlinearity of ϕ. Still, dot products in $\mathcal{F}$ computed with kernels offer a fast implicit access to this space that in turn leads to a low complexity nonlinear extractor. If we have a low-complexity (perhaps linear) kernel function the dot product $\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_j})$ can also be computed with fewer operations (e.g. $O(m)$) whether or not $\phi(\mathbf{x})$ is infinite in dimension.

Using this general schema various feature extraction methods such as Principal Component Analysis (the first generalization right after SVM, proposed by Schölkopf et al.) [19], [15], [13], Linear Discriminant Analysis [16], [18], [20] and Independent Component Analysis have already been nonlinearized. Hopefully other statistical methods will uncover their nonlinear counterparts in the near future.

In the subsequent section we will review the standard Independent Component Analysis and FastICA algorithms. Afterwards we will reformulate ICA in such a way that its input is the dot product of the input data, and then this basic operation will be replaced by kernel functions. The final part of the paper will discuss the results of experiments on the phoneme classification followed by some concluding remarks.

2 Independent Component Analysis

Independent Component Analysis [6], [8], [9], [7] is a general purpose statistical method that originally arose from the study of blind source separation (BSS). A typical BSS problem is the cocktail-party problem where several people are speaking simultaneously in the same room and several microphones record a mixture of speech signals. The task is to separate the voices of different speakers using the recorded samples. Another application of ICA is feature extraction, where the aim is to linearly transform the input data into uncorrelated components, along which the distribution of the sample set is the least Gaussian. The reason for this is that along these directions the data is supposedly easier to classify. For optimal selection of the independent directions several contrast function were defined using approximately equivalent approaches. Here we follow the way proposed by Hyvärinen [8], [9], [7]. Generally speaking, we expect

these functions to be non-negative and have a zero value for a Gaussian distribution. Negentropy is a useful measure having just this property, used for assessing non-Gaussianity (i.e. the least Gaussianity). Since obtaining this quantity via its definition is computationally rather difficult, a simple easily-computable approximation is normally employed. The negentropy of a variable η with zero mean and unit variance is estimated by the formula

$$J_G(\eta) \approx \left(E\{G(\eta)\} - E\{G(\nu)\}\right)^2 \tag{5}$$

where $G() : \mathbb{R} \to \mathbb{R}$ is an appropriate nonquadratic function, E denotes the expected value and ν is a standardized Gaussian variable. The following three choices of $G(\eta)$ are conventionally used: η^4, $\log(\cosh(\eta))$ or $-\exp(-\eta^2/2)$. It should be noticed that in (5) the expected value of $G(\nu)$ is a constant, its value only depending on the selected contrast function (e.g. $E(G_1(\nu)) = 3$). Hyvärinen recently proposed a fast iterative algorithm called FastICA for the selection of the new base vectors of the linearly transformed space. The goodness of a new direction $\mathbf{a}$ is measured by the following function, where η is replaced with $\boldsymbol{a} \cdot \mathbf{x}$ in the negentropy approximant (5):

$$\tau_G(\boldsymbol{a}) = \left(E\{G(\boldsymbol{a} \cdot \mathbf{x})) - E\{G(\nu)\}\right)^2. \tag{6}$$

As a matter of fact, FastICA is an approximate Newton iteration procedure for the local optimization of the function $\tau_G(\boldsymbol{a})$. Before running FastICA, however, the raw input data X must first be preprocessed – by centering and whitening it. Between centering and whitening we may, perhaps, also apply a deviance normalization because the standardized data used as an input for the whitening sometimes improves the efficiency of the FastICA algorithm. However, we should mention here there are many other iterative methods for performing Independent Component Analysis. Some of these (similar to FastICA) do require centering and whitening, while others do not. In general, experience has taught us that all these algorithms should converge faster on centered and whitened data, even those which do not really require it.

Centering. An essential step is to shift the original sample set $\mathbf{x_1}, \ldots, \mathbf{x_n}$ with its mean $\boldsymbol{\mu} = E\{\mathbf{x}\}$, to obtain data $\mathbf{x'_1} = \mathbf{x_1} - \boldsymbol{\mu}, \ldots, \mathbf{x'_n} = \mathbf{x_n} - \boldsymbol{\mu}$, with a mean of zero.

Whitening. The goal of this step is to transform the centered samples $\mathbf{x'_1}, \ldots, \mathbf{x'_n}$ via an orthogonal transformation Q into a space where the covariance matrix $\hat{C} = E\{\hat{\mathbf{x}}\hat{\mathbf{x}}^\top\}$ of the points $\hat{\mathbf{x}}_\mathbf{1} = Q^\top \mathbf{x'_1}, \ldots, \hat{\mathbf{x}}_\mathbf{n} = Q^\top \mathbf{x'_n}$ is the unit matrix. Since the standard principal component analysis [10] transforms the covariance matrix into a diagonal form, the elements in the diagonal being the eigenvalues of the original covariance matrix $C' = E\{\mathbf{x}'\mathbf{x}'^\top\}$, it only remains to transform each diagonal element to one. It is readily seen that the sample covariance matrix C' is symmetric positive semidefinite, so the eigenvectors are orthogonal and the corresponding real eigenvalues are nonnegative. If we then further assume that the eigenpairs of C' are $(\mathbf{c_1}, \lambda_1), \ldots, (\mathbf{c_m}, \lambda_m)$ and $\lambda_1 \geq \ldots \geq \lambda_m$, the transformation matrix Q will take the form $[\mathbf{c_1}\lambda_1^{-1/2}, \ldots, \mathbf{c_k}\lambda_k^{-1/2}]$. If k is less than m a dimensionality reduction is employed.

Properties of the preprocessing stage. Firstly, after centering and whitening for every normalized $\boldsymbol{a}$ the mean of $\boldsymbol{a} \cdot \hat{\mathbf{x}}_\mathbf{1}, \ldots, \boldsymbol{a} \cdot \hat{\mathbf{x}}_\mathbf{n}$ is zero, and its variance is one. Actually we need this since (5) requires that η has a zero mean and variance of one hence, with the substitution $\eta = \boldsymbol{a} \cdot \hat{\mathbf{x}}$, the projected data $\boldsymbol{a} \cdot \hat{\mathbf{x}}$ must also have this property. Secondly, for any matrix W the covariance matrix C_W of the transformed points $W\hat{\mathbf{x}}_\mathbf{1}, \ldots, W\hat{\mathbf{x}}_\mathbf{n}$ will remain a unit matrix if and only if W is orthogonal, since

$$C_W = E\{W\hat{\mathbf{x}}(W\hat{\mathbf{x}})^\top\} = WE\{\hat{\mathbf{x}}\hat{\mathbf{x}}^\top\}W^\top = WIW^\top = WW^\top \tag{7}$$

FastICA. After preprocessing, this method finds a new orthogonal base W for the preprocessed data, where the values of the non-Gaussianity measure τ_G for the base vectors are large[1]. The following pseudo-code give further details[2]:

```
% The input for this algorithm is the sample matrix X and the
% nonlinear function G, while the output is the transformation
% matrix W. The first and second order derivatives of G are
% denoted by G' and G''. (W_i W_i^T)^{-1/2} W_i is a symmetric
% decorrelation, where (W_i W_i^T)^{-1/2} can be obtained from its
% eigenvalue decomposition. If W_i W_i^T = E D E^T, then
% (W_i W_i^T)^{-1/2} is equal to E D^{-1/2} E^T.
procedure FastICA(X, G);
      μ = E{x};  x' = x − μ;  x̂ = Q^T x';   % centering & whitening
      Let W_0 be a random m × m orthogonal matrix;
      W_0 = (W_0 W_0^T)^{-1/2} W_0;
      i = 0;
      While W has not converged;
            for j = 1 to m
                  let s_j be the jth raw vector of W_i;
                  w_j = E{x̂ G'(s_j · x̂)} − E{G''(s_j · x̂)} s_j;
            end;
            i = i + 1;
            W_i = [w_1, ..., w_p]^T;
            W_i = (W_i W_i^T)^{-1/2} W_i;
      do
End procedure
```

Transformation of test vectors. For an arbitrary test vector $\mathbf{y} \in \mathcal{X}$ the transformation can be done using $\mathbf{y}^* = WQ^\top(\mathbf{y} - \boldsymbol{\mu})$. Here W denotes the orthogonal transformation matrix we obtained as the output from FastICA, while Q is the matrix obtained from whitening.

[1] Note that since the data remains whitened after an orthogonal transformation, ICA can be considered an extension of PCA.

[2] `MatLab` code available in [7].

3 Independent Component Analysis with Kernels

In this section we derive the kernel counterpart of *FastICA*. To this end, let the inner product be implicitly defined by the kernel function k in $\mathcal{F}$ with associated transformation ϕ. Now we need only extend nonlinearly the centering and whitening of the data, since after nonlinearizing $Q^\top(\mathbf{y} - \boldsymbol{\mu})$ we get data in $\mathcal{F}$ thus the nonlinearization of the iterative section becomes superfluous.
Centering in $\mathcal{F}$. We shift the data $\phi(\mathbf{x_1}), \ldots, \Phi(\mathbf{x_n})$ with its mean $\boldsymbol{\mu}^\phi$, to obtain data $\phi'(\mathbf{x_1}), \ldots, \phi'(\mathbf{x_n})$ with a mean of zero:

$$\phi'(\mathbf{x_1}) = \phi(\mathbf{x_1}) - \boldsymbol{\mu}^{\Phi}, \ldots, \phi'(\mathbf{x_n}) = \phi(\mathbf{x_n}) - \boldsymbol{\mu}^{\phi}, \qquad \boldsymbol{\mu}^{\phi} = \frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x_i}) \tag{8}$$

Whitening in $\mathcal{F}$. Much like that in linear ICA, the goal of this step is to find a transformation $Q_{\hat{\phi}}$ such that the covariance matrix

$$C^{\hat{\phi}} = \frac{1}{n}\sum_{i=1}^{n}\hat{\phi}(\mathbf{x_i})\hat{\phi}(\mathbf{x_i})^\top \tag{9}$$

of the sample $\hat{\phi}(\mathbf{x_1}) = Q_{\hat{\phi}}{}^\top\phi'(\mathbf{x_1})$, …, $\hat{\phi}(\mathbf{x_n}) = Q_{\hat{\phi}}{}^\top\phi'(\mathbf{x_n})$ is a unit matrix. As we saw earlier the column vectors of $Q_{\hat{\phi}}$ are the weighted eigenvectors of the positive semidefinite matrix $C^{\hat{\phi}}$. Because this eigen-problem is equivalent to determining the stationary points of the Rayleigh Quotient

$$\frac{\mathbf{a}^\top C^{\hat{\phi}}\mathbf{a}}{\mathbf{a}^\top\mathbf{a}}, \quad \mathbf{0} \neq \mathbf{a} \in \mathcal{F}, \tag{10}$$

this formula will be rearranged as an expression of dot products of the input data. Owing to the special form of $\mathbf{C}^{\hat{\phi}}$ we suppose that when we search for stationary points, $\mathbf{a}$ has the form

$$\mathbf{a} = \sum_{i=1}^{n}\alpha_i\hat{\phi}(\mathbf{x_i}). \tag{11}$$

We may arrive at this assumption in various ways, e.g. by decomposing an arbitrary vector $\mathbf{a}$ into $\mathbf{a_1} + \mathbf{a_2}$, where $\mathbf{a_1}$ is that component of $\mathbf{a}$ which falls in $SPAN(\hat{\phi}(\mathbf{x_1}), \ldots, \hat{\phi}(\mathbf{x_n}))$, while $\mathbf{a_2}$ is the component perpendicular to it. Then from the derivation of (10) we see that $\mathbf{a_2} \cdot \mathbf{a_2} = 0$ for the stationary points.
The following formulas give the Rayleigh Quotient as a function of $\boldsymbol{\alpha}$ and $k(\mathbf{x_i}, \mathbf{x_j})$:

$$\frac{\mathbf{a}^\top C^{\hat{\phi}}\mathbf{a}}{\mathbf{a}^\top\mathbf{a}} = \frac{\left(\sum_{t=1}^{n}\alpha_t\hat{\phi}(\mathbf{x_t})^\top\right)C^{\hat{\phi}}\left(\sum_{k=1}^{n}\alpha_k\hat{\phi}(\mathbf{x_k})\right)}{\left(\sum_{t=1}^{n}\alpha_t\hat{\phi}(\mathbf{x_t})^\top\right)\left(\sum_{k=1}^{n}\alpha_k\hat{\phi}(\mathbf{x_k})\right)} = \frac{\boldsymbol{\alpha}^\top\frac{1}{n}\hat{K}\hat{K}\boldsymbol{\alpha}}{\boldsymbol{\alpha}^\top\hat{K}\boldsymbol{\alpha}}, \tag{12}$$

where

$$\hat{K}_{tk} = \left(\phi(\mathbf{x_t})^\top - \left(\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x_i})^\top\right)\right)\left(\phi(\mathbf{x_k}) - \left(\frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x_i})\right)\right) = k(\mathbf{x_t},\mathbf{x_k}) - \left(\frac{1}{n}\sum_{i=1}^{n}\left(k(\mathbf{x_i},\mathbf{x_k}) + k(\mathbf{x_t},\mathbf{x_i})\right)\right) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}k(\mathbf{x_i},\mathbf{x_j}) \tag{13}$$

From differentiating (12) with respect to $\boldsymbol{\alpha}$ we see that the stationary points are the solution vectors of the general eigenvalue problem $\frac{1}{n}\hat{K}\hat{K}\boldsymbol{\alpha} = \lambda\hat{K}\boldsymbol{\alpha}$, which in this case is obviously equivalent to the problem $\frac{1}{n}\hat{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}$. Moreover, since $k(\mathbf{x_t},\mathbf{x_k}) = k(\mathbf{x_k},\mathbf{x_t})$ and[3] $\boldsymbol{\alpha}^\top\frac{1}{n}\hat{K}\boldsymbol{\alpha} = \frac{1}{n}\mathbf{a}^\top\mathbf{a} \geq 0$, the matrix $\frac{1}{n}\hat{K}$ is symmetric positive semidefinite and hence its eigenvectors are orthogonal and the corresponding real eigenvalues are non-negative. Let the k positive dominant eigenvalues of $\frac{1}{n}\hat{K}$ be denoted by $\lambda_1 \geq \ldots \geq \lambda_k > 0$ and the corresponding normalized eigenvectors be $\boldsymbol{\alpha^1},\ldots,\boldsymbol{\alpha^m}$. Then the orthogonal matrix of the transformation can be calculated via:

$$Q_{\hat{\phi}} := n^{-1/2}\left[\lambda_1^{-1}\sum_{i=1}^{n}\boldsymbol{\alpha^1}_i\hat{\phi}(\mathbf{x_i}),\ldots,\lambda_k^{-1}\sum_{i=1}^{n}\boldsymbol{\alpha^k}_i\hat{\phi}(\mathbf{x_i})\right], \tag{14}$$

where the factors $n^{-1/2}$ and λ^{-1} are needed to keep the column vectors of $Q_{\hat{\phi}}$ normalized [4].

Transformation of Test Vectors. Let $\mathbf{y} \in \mathcal{X}$ be an arbitrary test vector. New features can be expressed by $\phi(\mathbf{y})^* = WQ_{\hat{\phi}}^\top(\phi(\mathbf{y}) - \boldsymbol{\mu}^\phi)$, where $Q_{\hat{\phi}}$ denotes the matrix we obtained from whitening, while W denotes the orthogonal transformation matrix we got as the output of Kernel-FastICA. Practically speaking, Kernel-FastICA = Kernel-Centering + Kernel-Whitening + iterative section of the original FastICA. Of course, the computation of $\phi(\mathbf{y})^*$ involves only dot products:

$$\phi(\mathbf{y})^* = Wn^{-1/2}\left[{\lambda_1}^{-1}\sum_{i=1}^{n}\boldsymbol{\alpha^1}_i c_i,\ldots,{\lambda_k}^{-1}\sum_{i=1}^{n}\boldsymbol{\alpha^k}_i c_i\right]^\top, \tag{15}$$

$$c_i = \hat{\phi}(\mathbf{x_i})\cdot\hat{\phi}(\mathbf{y}) = k(\mathbf{x_i},\mathbf{y}) - \left(\frac{1}{n}\sum_{j=1}^{n}\left(k(\mathbf{x_i},\mathbf{x_j}) + k(\mathbf{x_j},\mathbf{y})\right)\right) + \frac{1}{n^2}\sum_{t=1}^{n}\sum_{j=1}^{n}k(\mathbf{x_t},\mathbf{x_j}). \tag{16}$$

4 Experimental Results

In these trials we wanted to see how well independent component analysis and its nonlinear counterpart could reduce the number of features and increase classification performance. Since automatic phoneme classification is of great importance

[3] Here we temporarily disregard the constraint $\mathbf{a} \neq 0$.

[4] If we use the factors $\lambda^{-1/2}$ instead of λ^{-1} in (14), then we obtain the Kernel Principal Component Analysis.

in the computer-assisted training of the speech & hearing handicapped, we chose phoneme classification as an area of investigation.

We developed a program to help with speech training of the hearing impaired, where the intention was to support or replace their diminished auditory feedback with a visual one. In our initial experiments we focussed on the classification of vowels, as the learning of the vowels is the most challenging for the hearing-impaired. The software we designed assumes that the vowels are pronounced in isolation or in the form of two-syllable words, which is a conventional training strategy. Visual feedback is provided on a frame-by-frame basis in the form of flickering letters, their brightness being proportional to the vowels recognizer's output (see Fig. 2.).

Corpus. For training and testing purposes we recorded samples from 25 speakers. The speech signals were recorded and stored at a sampling rate of 22050 Hz in 16-bit quality. Each speakers uttered 59 two-syllable Hungarian words of the CVCVC form, where the consonants (C) are mostly unvoiced plosives so as to ease the detection of the vowels (V). The distribution of the 9 vowels (long and short versions were not distinguished) is approximately uniform in the database. In the trials 20 speakers were used for training and 5 for testing.

Feature Sets. The signals were processed in 10 ms frames, from which the log-energies of 24 critical-bands were extracted using FFT and triangular weighting [17]. In our early tests we only utilized the filter-bank log-energies from the most centered frame of the steady-state part of each vowel ("FBLE" set). Then we added the derivatives of these features to model the signal dynamics ("FBLE+Deriv" set). In another experiment we smoothed the feature trajectories so as to remove the effects of short noises and disturbances ("FBLE Smooth" set). In yet another set of features we extended the log-energies with the gravity centers of four frequency bands which approximately corresponds to the possible values of the formants. These gravity centers provide a crude approximation of the formants ("FBLE+Grav" set) [2].

Classifiers. In all the trials with *Artificial Neural Nets* (ANN) [3] the well-known three-layer feed-forward MLP networks were employed with the back-propagation learning rule. The number of hidden neurons was equal to the number of features. In the *Support Vector Machine* (SVM) [21] experiments we always applied the Gaussian RBF kernel function ($k_2,\ r = 10$).

Transformations. In our tests with ICA and Kernel-ICA the eigenvectors belonging to the 16 dominant eigenvalues were selected as basis vectors for the transformed space and the nonlinear function $G(\eta)$ was η^4. In Kernel-ICA the kernel function was as before. Naturally when we applied a certain transformation on the training set before learning, we used the same transformation on the test data during testing.

5 Results and Discussion

Table 1 shows the recognition errors. Here the rows represent the four feature sets, while the columns correspond to the applied transformation and classifier combinations.

On examining the results the first striking point is that although the transformations retained only 16 features, the classifiers could achieve the same or better scores. The reason for this is that ICA determines directions with high non-Gaussianity, which is proven to be a beneficial feature extraction strategy before the classification. As regards the various feature sets, we realized that the gravity center features and smoothing the trajectories both lead to a remarkable improvement in the results, while adding the derivatives in no way increased performance. Most likely, a clever combination of smoothing and taking derivatives (or RASTA filtering) could yield still better results. Another notable observation is that SVM consistently outperformed ANN by several percent. This can mostly be attributed to the fact that the SVM algorithm can deal with overfitting. The latter is a common problem in ANN training.

Finally, with Kernel-ICA, we have to conclude that it is worthwhile continuing doing experiments with this type of nonlinearity. However, the problem of finding the best kernel function for the dot product extension or of choosing the best nonlinearity for the contrast function remains an open one at present.

Table 1. Recognition errors for the vowel classification task. The numbers in parenthesis correspond to the number of features.

	none ANN	**none SVM**	**ICA ANN** (16)	**ICA SVM** (16)	**K−ICA ANN** (16)	**K−ICA SVM** (16)
FBLE (24)	26.71%	22.70%	25.65%	23.84%	23.19%	22.20%
FBLE+Deriv (48)	25.82%	24.01%	28.62%	26.81%	24.67%	23.35%
FBLE+Grav (32)	24.01%	22.03%	23.68%	23.35%	20.88%	20.06%
FBLE Smooth (24)	23.68%	21.05%	23.84%	23.84%	22.03%	20.39%

6 Conclusion

In this paper we presented a new nonlinearized version of Independent Component Analysis using a kernel approach. Encouraged by [19] and [8], we could perform further extensions on Kernel Principal Component Analysis (KPCA), since ICA can be viewed as a modified PCA (centering and whitening) and an additional iterative process. But regardless of this we have demonstrated the superiority of Kernel-ICA over its linear counterpart on the phoneme classification task. Unfortunately, feature extraction in kernel feature spaces is currently much slower, than the traditional linear version. Hence in the near future we will focus our effors on working with a sparse data representation scheme that is hoped will speed-up the computations somewhat. This seems to be a good direction to go in.

References

1. Aizerman, M. A., Braverman, E. M. and Rozonoer L. I., Theoretical foundation of the potential function method in pattern recognition learning, *Automat. Remote Cont.*, vol. 25, pp. 821-837, 1964.
2. Albesano, D., De Mori, R., Gemello, R., and Mana, F., A study on the effect of adding new dimensions to trajectories in the acoustic space, *Proc. of EuroSpeech'99,* pp. 1503-1506, 1999.
3. Bishop, C. M., *Neural Networks for Pattern Recognition,* Oxford Univ. Press, 1995.
4. Boser, B. E., Guyon, I. M. and Vapnik V. N, A training algorithm for optimal margin classifier, in *Proc. 5th Annu. ACM Workshop Computat. Learning Theory*, D. Haussler, ed., Pittsburgh, PA, pp. 144-152, 1992.
5. Cristianini, N. and Shawe-Taylor, J. *An Introduction to Support Vector Machines and other kernel-based learning methods* Cambridge University Press, 2000.
6. Comon, P. Independent component analysis, A new concept? *Signal Processing,* 36:287-314, 1994.
7. FastICA Web Page, http:// www.cis.hut.fi/projects/ica/fastica/index.shtml, 2001.
8. Hyvärinen, A. A family of fixed-point algorithms for independent component analysis In *Proceedings of ICASSP*, Munich, Germany, 1997.
9. Hyvärinen, A. New Approximations of Differential Entropy for Independent Component Analysis and Projection Pursuit. In *Advances in Neural Information Processing Systems*, 10:273-279, MIT Press, 1998.
10. Jolliffe, I. J. *Principal Component Analysis,* Springer-Verlag, New York, 1986.
11. Kernel Machines Web Page, http:// www.kernel-machines.org, 2001.
12. Kocsor, A., Tóth, L., Kuba, A. Jr., Kovács, K., Jelasity, M., Gyimóthy, T. and Csirik, J., A Comparative Study of Several Feature Transformation and Learning Methods for Phoneme Classification, *Int. Journal of Speech Technology*, Vol. 3., No. 3/4, pp. 263-276, 2000.
13. Kocsor, A., Kuba, A. Jr. and Tóth, L. Phoneme Classification Using Kernel Principal Component Analysis, *Priodica Polytechnica*, in print, 2001.
14. Kocsor, A., Tóth, L. and Paczolay, D., A Nonlinearized Discriminant Analysis and its Application to Speech Impediment Therapy, *in V. Matousek et al. (eds.): Text, Speech and Dialogue, Proc. of TSD 2001,* Springer Verlag LNAI, in print, 2001.
15. Müller, K.-R., Mika, S., Rätsch, G., Tsuda, K. and Schölkopf, B., An Introduction to Kernel-Based Learning Algorithms, *IEEE Transactions on Neural Networks*, Vol. 12, No. 2, 2001.
16. Mika, S., Rätsch, G., Weston, J., Schölkopf, B. and Müller, K.-R., Fisher Discriminant Analysis with kernels, in *Neural Networks for Signal Processing IX*, Hu, J. Larsen, E. Wilson and S. Douglas, Eds. Piscataway, NJ:IEEE, pp. 41-48, 1999.
17. Rabiner, L. and Juang, B.-H. *Fundamentals of Speech Recognition,* Prentice Hall, 1993.
18. Roth, V. and Steinhage, V., Nonlinear discriminant analysis using kernel functions, In *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen and K.-R. Müeller, Eds. Cambridge, MA:MIT Press, pp. 526-532, 2000.
19. Schölkopf, B., Smola, A. J. and Müller, K.-R., *Nonlinear component analysis as a kernel eigenvalue problem, Neural Comput.,* vol. 10, pp. 1299-1319, 1998.

20. Toth, L., Kocsor, A., and Kovács, K., A Discriminative Segmental Speech Model and Its Application to Hungarian Number Recognition, *in Sojka, P. et al.(eds.):Text, Speech and Dialogue, Proceedings of TSD 2000,* Springer Verlag LNAI series, vol. 1902, pp. 307-313, 2000.
21. Vapnik, V. N., *Statistical Learning Theory,* John Wiley & Sons Inc., 1998.

On Majority Voting Games in Trees*

Rastislav Královič

Department of Computer Science
Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava
Slovakia
kralovic@dcs.fmph.uniba.sk

Abstract. We consider the following synchronous colouring game played on a simple connected graph with vertices coloured black or white. During one step of the game, each vertex is recoloured according to the majority of its neighbours. The variants of the model differ by the choice of a particular tie-breaking rule and possible rule for enforcing monotonicity. Two tie-breaking rules we consider are *simple majority* and *strong majority*, the first in case of a tie recolours the vertex black and the latter does not change the colour. The monotonicity-enforcing rule allows the voting only in white vertices, thus leaving all black vertices intact. This model is called *irreversible*.
These synchronous dynamic systems have been extensively studied and have many applications in molecular biology, distributed systems modelling, etc.
In this paper we give two results describing the behaviour of these systems on trees. First we count the number of fixpoints of strong majority rule on complete binary trees to be asymptotically $4N \cdot (2\alpha)^N$ where N is the number of vertices and $0.7685 \leq \alpha \leq 0.7686$.
The second result is an algorithm for testing whether a given configuration on an arbitrary tree evolves into an all-black state under irreversible simple majority rule. The algorithm works in time $O(t \log t)$ where t is the number of black vertices and uses labels of length $O(\log N)$.

1 Introduction

Let us consider the following colouring game played on a simple connected graph. Initially, each vertex is assigned a colour from the set $\{black, white\}$. The game proceeds in synchronous rounds. In each round, every vertex checks the colours of all its neighbours and adopts the colour of the majority. In order to have a rigorous definition we must, however, supply a tie-breaking rule. In case of a tie there are generally four possibilities: the vertex may recolour itself white, black, may keep its colour or flip its colour. The choice of different tie-breaking rule leads to different behaviour of the system [17]. We consider the two most studied rules, so called strong and simple majority. When using *simple majority* rule, in

* The research has been supported by grants Comenius University 102/2001/UK and VEGA 1/7155/20.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 282–291, 2001.

case of a tie the vertex recolours itself *black*. In *strong majority* rule, in case of a tie the vertex does not change its colour.

This kind of systems appears in many different areas of research (e.g. gene expression networks [9], spin glass model [6], fault-local distributed mending [10], etc.) and they have been extensively studied. The most often addressed issues were the periodicity behaviour [7], [19], [20], [6], [13], [14], the number of fixpoints [1], and various questions concerning dynamos [11], [17], [18], [2], [4].

In this paper we address two important questions about this game. The first one is the number of different fixpoints (i.e. colourings which are stable under the majority rule). It is known (e.g. [6]) that the limit structures of this game are either fixpoints or have period 2. In [1] the number of fixpoints on rings was given. The study of this question was motivated by experiments in molecular biology which showed that even very large gene expression networks have only a small number of stable structures. In [1] it was shown that in a very simplified model of such networks with only the strong majority function on a ring topology, the number of fixpoints is an exponentially small fraction of all configurations. We give a similar result for the case of complete binary trees.

The second question concerns a particular fixpoint, namely the monochromatic (all-black) one. The configurations that form the basin of attraction of this fixpoint (i.e. the configurations for which iterative application of the majority rule leads to all-black configuration) are called *dynamic monopolies (dynamos)*. The importance of this notion follows from the fact that if the game is viewed as a model of a behaviour of a faulty point-to-point system based on majority voting, dynamos correspond to the sets of initial faults that cause the entire system to fail. Dynamos have been introduced in [11] and since that time they have been intensively studied in the literature. The main line of the previous research has been oriented on the size of dynamos for various topologies (see e.g. survey paper [16]).

In this paper we address the issue of testing whether a given configuration on an arbitrary tree is an irreversible simple majority dynamo when only the identities of the black nodes are given. We use the notion of a labelling scheme ([15]) where each vertex is given a (short) label in a preprocessing phase. The input to the algorithm are the labels of black vertices. We give an algorithm which decides whether the given set of black vertices is a dynamo in time $O(t \log t)$ with $O(\log N)$-bit labels where t is the number of black vertices and N is the number of all vertices.

The paper is organised as follows. In Section 2 we give the asymptotic ratio of fixpoints of the strong majority rule in complete binary trees. In Section 3 we present a testing algorithm for simple irreversible dynamos in arbitrary trees.

2 The Number of Fixpoints on Complete Binary Trees

Given a dynamic system, the number of fixpoints (i.e. configurations that are stable under the system's action) is one of the first and most important questions

to be asked. In this section we consider the strong majority rule acting on a complete binary tree.

Agur et al [1] studied the number of fixpoints on rings under the strong majority rule. The asymptotic ratio of fixpoints (i.e. the ratio of fixpoints to all configurations) is c^N, where N is the number of vertices and $c = (1+\sqrt{5})/4$.

This result shows that only a small fraction of all configurations are fixpoints. In [1], this fact is compared with experimental results about gene expression networks. The genetic code of a cell can be imagined as a set of genes which are either active (expressed) or passive. Each gene has associated a boolean function which decides if this particular gene should be activated based on the status of a number of other genes. Experiments and computer simulations showed that although the number of possible configurations of a network with size comparable to the genetic code of a cell is very large, there is only a small number of fixpoints with large attractors. In this section we ask the question whether this property is preserved also when there is no feedback, i.e. the graph is acyclic.

We give the asymptotic ratio of fixpoints of strong majority rule on complete binary trees of height n.

Theorem 1. *Let f_n be the number of different fixpoints of strong majority rule on a complete binary tree of height n. Then the asymptotic ratio of fixpoints is $\lim_{n\mapsto\infty} f_n/2^N = 4N\cdot\alpha^N$, where $N = 2^n$ is the number of vertices and $0.7685 \le \alpha \le 0.7686$.*

Proof. Let $t(i,n)$ be the number of different fixpoints on a tree of height n such that the root has value i. As we are using the strong majority rule, it holds $t(0,n) = t(1,n)$. Let us denote $t(n) = t(0,n)$. Then it holds $f_n = 2t(n)$.

Consider a fixpoint configuration on a tree of height n. W.l.o.g. let the root v be black. There are two possible cases: either both its children u, w are black or one of them is black and the other is white (note that in the case of tie the colour remains unchanged). In the former case, each of u and w is black and has one black neighbour v. It follows that it is necessary and sufficient that the configuration restricted to either of the subtrees rooted at u and w is a fixpoint. In the later case, let u be black and w white. Now u is in the same situation as before, however w needs both children to be white. Hence each of w's children is white and has one white neighbour; again a situation similar to the above. This analysis leads us to the recurrence:

$$\begin{aligned} &t(0) = 1, t(1) = 1\\ &t(n) = t(n-1)^2 + 2t(n-1)t(n-2)^2 \end{aligned}$$

which can be transformed by substitution $q(n) = t(n)/t(n-1)^2$ to the form

$$\begin{aligned} &q(1) = 1\\ &q(n) = 1 + \tfrac{2}{q(n-1)} \end{aligned}$$

Looking for a solution of the form $q(n) = s(n)/s(n-1)$ we get $s(n) = 2^{n+1} + (-1)^n$. It follows that

$$t(n) = \prod_{i=1}^{n}\left(\frac{s(i)}{s(i-1)}\right)^{2^{n-i}} = \frac{2^{n+1}+(-1)^n}{3^{2^{n-1}}}\prod_{i=1}^{n-1} s(i)^{2^{n-1-i}}$$

Now let $r(n) = \sum_{i=1}^{n-1} 2^{-i} \ln(2^{i+1} + (-1)^i)$. Clearly

$$t(n) = \frac{2^{n+1} + (-1)^n}{3^{2^{n-1}}} e^{2^{n-1} r(n)}$$

As can be easily seen, the sequence $\{r(n)\}_1^\infty$ converges and we are interested in $\lim_{n \mapsto \infty} r(n)$. Splitting $r(n)$ to odd and even i's we get

$$r(n) = \sum_{i=1}^{\lceil \frac{n}{2} \rceil - 1} \frac{\ln(2 \cdot 4^i + 1)}{4^i} + 2 \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{\ln(4^i - 1)}{4^i}$$

Both of the sums converge, so we can take the limit $\lim_{n \mapsto \infty} r(n) = r = \sum_{i=1}^{\infty} \frac{\ln(2 \cdot 4^i + 1)}{4^i} + 2 \sum_{i=1}^{\infty} \frac{\ln(4^i - 1)}{4^i} = r^{(1)} + 2r^{(2)}$. Let us consider the two sums separately. Each of them is of the form $r^{(j)} = \sum_{i=1}^{\infty} \frac{\ln(a \cdot 4^i + b)}{4^i}$, $a \geq 1$, $b \geq -1$. As the inner function is continuous and decreasing in $\langle 2, \infty)$, $r^{(j)}$ can be approximated by

$$\sum_{i=1}^{p} \frac{\ln(a \cdot 4^i + b)}{4^i} \leq r^{(j)} \leq \sum_{i=1}^{p} \frac{\ln(a \cdot 4^i + b)}{4^i} + \int_p^\infty \frac{\ln(a \cdot 4^x + b)}{4^x} dx, \quad p \geq 2$$

It holds $I_x := \int \frac{\ln(a \cdot 4^x + b)}{4^x} dx = \frac{a \ln(a)}{b \ln(4)} + \frac{ax}{b} - \frac{\ln(a \cdot 4^x + b)}{\ln(4)} \left(\frac{1}{4^x} + \frac{a}{b} \right)$ and $\lim_{x \mapsto \infty} I_x = 0$. An easy computation for $p = 10$ reveals, that

$$\begin{array}{r} 0.87867 \leq r^{(1)} \leq 0.87869 \\ 0.53990 \leq r^{(2)} \leq 0.53992 \\ \hline 1.95847 \leq r \quad \leq 1.95853 \end{array}$$

The asymptotic ratio of fixpoints is

$$\lim_{n \mapsto \infty} \frac{2t(n)}{2^{2^n}} = \lim_{n \mapsto \infty} (4 \cdot 2^n + 2(-1)^n) \left(\frac{e^{\frac{r}{2}}}{2\sqrt{3}} \right)^{2^n} = 4N \cdot \alpha^N$$

where $0.7685 \leq \alpha \leq 0.7686$. □

3 Dynamo Testing on Trees

Let us now focus our attention to another question concerning the colouring game: what is the complexity of deciding, given a configuration C, whether it belongs to the attractor of a particular, namely all-black, fixpoint (i.e. whether C is a dynamo). The motivation behind this problem comes from the modelling of faulty networks. The configuration may be viewed as the subset of vertices that are coloured black, which represent faulty nodes. A node may become faulty if the majority of its neighbours is faulty. This steams from the fact that the state of a node depends on the messages it has received from its neighbours. The (Byzantine) faulty majority of its neighbours can hence control its behaviour.

In order to take into account the fact that the faulty nodes remain faulty, a modification of the majority rule called *irreversible majority* has been widely

studied [3,4,5,12]. Using this rule, the (simple or strong) majority voting is applied only in white vertices. The black ones remain black throughout the whole execution.

The configurations in the attractor of the all-black fixpoint are called dynamos:

Definition 1. *A set of black vertices M is called a (irreversible) dynamo if and only if the (irreversible) majority computation starting from M colours black the whole graph.*

In our view of the black vertices as faulty processors, a dynamo represents a state of the system which eventually leads to a complete crash. Therefore it is important to be able to test whether the given configuration forms a dynamo.

The problem we study is the following:

Definition 2. TESTING *is a problem to decide, given a graph G and a set M of black vertices, whether M is an irreversible dynamo.*

It is easy to see that since the black vertices are never recoloured white, the sequential simulation of the whole computation can be done in $O(m)$ time.

Observation 1. *There is an $O(m)$ algorithm for* TESTING *in arbitrary graphs, where m is the number of edges.*

Proof. Consider the edge boundary $\partial(M)$ of M. During the computation of the system each edge is added and withdrawn from $\partial(M)$ at most once. □

Corollary 1. *The complexity of* TESTING *in arbitrary graphs is $\Theta(m)$.*

It is, however, often desirable to measure the complexity of TESTING not in terms of the size of the entire graph but in terms of the number of the black nodes. This corresponds to a situation when the only information about the state of the system consists of some identifiers of the faulty nodes (the actual topology is unknown except for the fact that it is acyclic). We want an algorithm which, given the identifiers of black vertices, decides whether the current state is a dynamo. As we consider the topology to be fixed, we allow some preprocessing to choose the vertex labels. In order to achieve this goal we use the notion of labelling schemes.

Informally, a labelling scheme allows a preprocessing phase in which each vertex of a graph receives a (short) label. The aim is, given labels of some vertices, to compute the value of a function defined over subsets of vertices of the graph. The previous research involves mainly distance labelling schemes [8] where the function to be computed is the distance of two vertices (and some default value for all other subsets of vertices), and labelling schemes for least common ancestor and Steiner tree on trees [15].

Another examples of labelling schemes have been used in various areas of applications, e.g. compact routing using ILS [22,23,21].

Definition 3. *An $f(N)$-labelling scheme is a polynomially computable function that assigns to each vertex of an N-node graph G a binary string of length at most $f(N)$.*

The local version of the TESTING problem, in which only the labels of faulty nodes are known can be described as follows:

Definition 4. *Let $\mathcal{L}$ be a labelling scheme. $\mathcal{L}$-TESTING is a problem to tell, given a set of t labels of some nodes of G, whether these nodes form an irreversible dynamo in G.*

$f(N)$-TESTING is a problem to devise an $f(N)$-labelling scheme and an algorithm for $\mathcal{L}$-TESTING.

Next, we give an efficient $O(\log N)$-TESTING algorithm on trees. In the phase of assigning labels, we shall use the following lemma:

Lemma 1. *Consider a string α, $|\alpha| = m$. Then for any $n > 2$ and for any $k < n$ there are strings $\beta_1, ..., \beta_n$ such that for any choice of k strings $\beta_{i_1}, ..., \beta_{i_k}$ it is possible to reconstruct α. Moreover, for each β_i it holds $|\beta_i| \leq \lceil m\left(1 - \frac{k-1}{n}\right)\rceil$.*

Proof. Consider an $n \times m$ matrix M of zeroes and ones. Let the i-th column contain all ones except for a consecutive block of $k-1$ zeroes starting from row $1+(i-1)(k-1)$. Let β_i be a sequence of those bits a_j from α for which $M_{i,j} = 1$. Because in each column of M there are $k-1$ zeroes, in every k strings there is for each bit at least one β_i containing that bit. Moreover, as the number of ones in every two rows differs at most by 1, the maximal number of ones in one row is $\lceil m\left(1 - \frac{k-1}{n}\right)\rceil$. □

The algorithm in the following theorem uses $O(\log N)$ bit labels in order to test the given configuration in time $O(t \log t)$ where t is the number of black vertices. The running time of this algorithm on configurations of size $\omega(N/\log N)$ is slower than the trivial testing algorithm equipped with full topology information. This is due to the fact that the input may contain the labels of black vertices in any order which requires a sorting phase. However on configuration with $O(N/\log N)$ black vertices, the running time of this algorithm is linear. Please note that as there are trees for which some configurations of size $O(N/\log N)$ are dynamos and some are not, it is not possible to solve the problem based on the number of black vertices only.

Theorem 2. *Let the unit-cost operations be operations on $\log N$-bit words. Then there exists an $O(t \log t)$-time algorithm for irreversible $O(\log N)$-TESTING in arbitrary trees using simple (strong) majority.*

Proof. Given an N-vertex tree T, we show how to construct $O(\log N)$ bits long labels L_v for each vertex v and how to test for a dynamo in time $O(t \log t)$ given t labels.

First consider the case when every vertex v in T has $deg_v \neq 2$. Suppose there is an unique identifier ID_v for each vertex v, such that $|ID_v| = O(\log N)$. Choose an arbitrary vertex to be the root. In each vertex choose a fixed ordering of its children. By "level i" we mean the set of vertices having distance i from the root. To each vertex v assign a local label L'_v such that $L'_v = \langle ID_v, deg_v, ID_{parent_v}, deg_{parent_v}, level_v, s_v, l_v, c_v\rangle$, where s_v denotes the v's number in the children's ordering of its parent, l_v denotes the number of

vertices with degree more than one in the same level as v, and c_v is *true* if all children of v are leaves. For the case of root, the values $ID_{parent_{root}}$ and ID_{root} are equal. Clearly, L'_v can be constructed in $O(N \log N)$ time. The labels L_v are constructed recursively as follows. For the root it is $L_{root} = L'_{root}$. Now consider a vertex v with label L_v such that $deg_v = d + 1$. If $d = 2$ let β_1 be the first half of L_v and β_2 the second half. If $d > 2$ choose $\beta_1, ..., \beta_d$ according to Lemma 1 in such a way that it is possible to reconstruct L_v from arbitrary $\lceil \frac{d}{2} \rceil$ β's. Let w be the i-th son of v in v's ordering. Then $L_w = \langle L'_w, \beta_i \rangle$. The length of each β_i is at most $\lceil |L_v| \frac{d+2}{2d} \rceil$, i.e. $|\beta_i| \leq \lceil |L_v| \frac{5}{6} \rceil$. Hence the length of a label is bounded from above by $6L + 6$, where L is the maximal length of a local label, i.e. $L = O(\log N)$.

The testing algorithm proceeds as follows. Given t labels, sort them according to the $level_v$. Start with the bottom-most level. For each initially black vertex v, check if its local c_v is true; if not, the configuration is not a dynamo. Count the number of initially black vertices with degree more than one. Using the value l_v, if there are some vertices with degree more than one in the bottom-most level which are not initially black, the configuration is not a dynamo. Now suppose that level l has been already checked. The level $l - 1$ is checked as follows. Construct a list $\mathcal{L}$ of IDs of all parents of all black vertices from level l. For each of them, the information about their ID, *degree* and the number of black children is available from the labels of vertices in level l. For each initially black vertex in level $l-1$ not in $\mathcal{L}$ check its c_v value. Assign colour to each vertex $v \in \mathcal{L}$ as follows: if v has at least $\lfloor (deg(v) + 1)/2 \rfloor$ (for strong majority) or $\lceil deg(v)/2 \rceil$ (for simple majority) black children (or is initially black) colour v black. If v has exactly $\lfloor deg(v)/2 \rfloor$ (for strong majority) or $\lceil deg(v)/2 \rceil - 1$ (for simple majority) black children colour v gray. Otherwise v is white and the configuration is not a dynamo. For all $v \in \mathcal{L}$ that have been coloured black, construct their L_v from labels of their children. At this stage, there must be at least one black vertex in level $l - 1$. Extract the l_v from its label. Using this value and the number of black and gray vertices check the existence of a white vertex at level $l - 1$. If the root is coloured black, the configuration is a dynamo.

First note that in the algorithm a vertex is coloured black only if it becomes black sometimes during the computation. Similarly if a vertex is coloured white by the algorithm it will never be coloured black during the computation. The gray vertices may become black in the computation only if there is a gray ancestor-path ending in a black vertex. Then, the whole gray subtree is coloured black.

In each level all vertices are considered: either they are in $\mathcal{L}$, or they are white non-leaf and are checked from the l_v value, or they are white leaves and are checked in the next level from their parent. Hence the algorithm is correct.

Since the explicitly constructed black vertices form a tree with degree $\neq 2$ and the number of gray vertices is at most linear in the number of black ones, the dominating term in the time complexity comes from the initial sorting.

Now consider the case when some vertices may have degree 2. In the phase of assigning labels, the following changes will be made. First, before a root is chosen,

every path consisting of vertices of degree 2 will be replaced by a "supernode". This "supernode" will be treated differently in simple and strong models.

In simple majority, each "supernode" act as one node, i.e. all nodes from a supernode will have the same label. This is because in the computation the whole chain recolours black exactly if one of the member vertices is black. So now there are only two cases of vertices with degree 2: the root or a vertex with one parent and one child, both of degree $\neq 2$. For the case of root the algorithm works properly. Consider a vertex u with the parent v and a child w. As vertex u can be coloured by either w or v, it is sufficient that the *parent* information and the additional bits used to reconstruct the parent's label are set in both u and w to "point to" v. Moreover, L'_w may contain additional information about both w's and u's levels.

In the strong majority model, nodes of each supernode have additional identifiers and each supernode is checked if it can be coloured, i.e. all nodes in the supernode except for first and last must be coloured black. The situation is then similar to the case above with the difference that the first and last nodes of a supernode are treated separately. □

4 Conclusion

We have addressed two questions concerning the iterated majority voting system. The first one concerns the number of fixpoints of the iterated majority voting on a complete binary tree. The result compares to similar results on rings from [1]. An interesting question for the future research may be to show how the number of fixpoints varies over different topologies with different models (simple/strong reversible/irreversible majority).

The other question we asked is the complexity of the dynamo testing in trees. We showed that it is possible to test whether a configuration is a dynamo in time $O(t \log t)$ where t is the size of the configuration (i.e. the number of its black vertices) using $O(\log N)$-bit labels. It might be interesting to have other examples of topologies for which an $O(\log N)$-TESTING algorithm with time complexity $f(N)$ exists such that $f(d) = o(m)$ where d is the size of minimal dynamo and m is the number of edges.

Acknowledgements

The author would like to thank Peter Ružička for many illuminating discussions about the paper.

References

1. Z. Agur, A.S. Frankel, S.T. Klein: The Number of Fixed Points of the Majority Rule. *Discrete Mathematics 70:295–302, 1988*
2. J.C. Bermond, J. Bond, D. Peleg, S. Pérennes: Tight Bounds on the Size of 2-monopolies. *Proc. 3rd Colloq. on Structural Information & Communication Complexity (SIROCCO),* Sienna, Italy, June 1996
3. P. Flocchini, E. Lodi, F. Luccio, L. Pagli, N. Santoro: Irreversible Dynamos in Tori. *Proc. Euro-Par,* Southampton, England, 1998, 554–562
4. P. Flocchini, R. Královič, A. Roncato, P. Ružička, N. Santoro: On Time versus Size for Monotone Dynamic Monopolies in Regular Topologies. *Proc. 7th Colloq. on Structural Information & Communication Complexity (SIROCCO),* L'Aquilla, Italy, June 1999, 111–127
5. P. Flocchini, F. Geurts, N.Santoro: Irreversible Dynamos in Chordal Rings. *Proc. 25th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG),* Ascona, Switzerland, June 1999, 202–214
6. E. Goles-Chacc, F. Fogelman-Soulie, D. Pellegrin: Decreasing Energy Functions as a Tool for Studying Threshold Networks. *Discrete Applied Mathematics 12:261–277, 1985*
7. E. Goles, J. Olivos: Periodic behaviour of generalized threshold functions. *Discrete Mathematics 30:187–189, 1980*
8. C. Gavoille, D. Peleg, S. Pérennes, R. Raz: Distance Labeling in Graphs. *Research Report RR-1182-97, LaBRI, Univ. of Bordeaux, 1999*
9. S. Huang: Gene Expression Profiling, Genetic Networks, and Cellular States: an Integrating Concept for Tumorigenesis and Drug Discovery. *To appear in Journal of Molecular Medicine, August 2000, Springer Verlag*
10. S. Kutten, D. Peleg: Fault-Local Distributed Mending. *Proc. Principles of Distributed Computing (PODC),* Ottawa, 1995, ACM, 20–27
11. N. Linial, D. Peleg, Y. Rabinovich, M. Saks: Sphere packing and local majorities in graphs. *2nd ISTCS,* June 1993, IEEE Computer Soc. Press, 141–149
12. F. Luccio, L. Pagli, H. Sanossian: Irreversible Dynamos in Butterflies. *Proc. 6th Colloq. on Structural Information & Communication Complexity (SIROCCO),* Bordeaux, 1999, Carleton University Press
13. G. Moran: On r-majority vote action on 0–1 sequences. *Discrete Mathematics 132:145–174, 1994*
14. G. Moran: On the period-two-property of the majority operator in infinite graphs. *Transactions of the American Mathematical Society 347(5):1649–1667, 1995*
15. D. Peleg: Informative Labeling Schemes for Graphs. *Proc. Mathematical Foundations of Computer Science (MFCS 2000),* 2000, LNCS 1893, Springer Verlag, 579–588 (see also Technical Report MCS00-05, The Weizmann Institute of Science, 2000)
16. D. Peleg: Local Majorities, Coalitions and Monopolies in Graphs: A Review. *Theoretical Computer Science, to appear*

17. D. Peleg: Local Majority Voting, Small Coalitions and Controlling Monopolies in Graphs: A Review. *Proc. 3rd Colloq. on Structural Information & Communication Complexity (SIROCCO),* 1996, 152–169 (see also Technical Report CS96-12, the Weizmann Institute of Science, 1996)
18. D. Peleg: Graph Immunity Against Local Influence. *Technical Report CS96-11, The Weizmann Institute of Science, 1996*
19. S. Poljak, M. Sura: On periodical behaviour in societies with symmetric influences. *Combinatorica 3:119–121, 1983*
20. S. Poljak, D. Turzik: On an application of convexity to discrete systems. *Discrete Applied Mathematics 13:27–32, 1986*
21. P. Ružička: On the Efficiency of Interval Routing Algorithms. *Proc. MFCS'88* LNCS 324, 492–500, 1988 (see also *The Computer Journal 34:475–486, 1991*)
22. N. Santoro, R. Khatib: Labelling and implicit routing in networks. *The Computer Journal 28:5–8, 1985*
23. J. van Leeuwen, R. B. Tan: Interval Routing. *The Computer Journal 30:298–307, 1987*

Time and Space Complexity of Reversible Pebbling*

Richard Královič

Department of Computer Science
Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava
Slovakia

Abstract. In the context of quantum computing, reversible computations play an important role. In this paper the model of the reversible pebble game introduced by Bennett is considered. Reversible pebble game is an abstraction of a reversible computation, that allows to examine the space and time complexity for various classes of problems. We present techniques for proving lower and upper bounds on time and space complexity. Using these techniques we show a partial lower bound on time for optimal space (time for optimal space is not $o(n \lg n)$) and a time-space tradeoff (space $O(\sqrt[k]{n})$ for time $2^k n$) for a chain of length n. Further, we show a tight optimal space bound $(h + \Theta(\lg^* h))$ for a binary tree of height h and we discuss space complexity for a butterfly. By these results we give an evidence, that for reversible computations more resources are needed with respect to standard irreversible computations.

1 Introduction

Standard pebble game was introduced as a graph-theoretic model, that enables to analyse time-space complexity of deterministic computations. In this model, values to be computed are represented by vertices of a directed acyclic graph. An edge from a vertex a to a vertex b represents the fact, that for computing the value a, the value b has to be already known. Computation is modelled by laying and removing pebbles on/from the vertices. Pebbles represent the memory locations. A pebble laying on a certain vertex represents the fact that the value of this vertex is already computed and stored in the memory.

The importance of the pebble game is in the following two step paradigm:

1. the inherent structure of studied problem forms the class of acyclic graphs; investigate time-space complexity of pebbling this class of graphs;
2. apply the obtained time-space results to create a time efficient space restricted computation of the original studied problem.

Various modifications of this game were studied in connection with different models of computations (e.g. pebble game with black and white pebbles for nondeterministic computations, two person pebble game for alternation computations, pebble game with red and blue pebbles for input-output complexity

* Supported in part by grant from VEGA 1/7155/20.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 292–303, 2001.

analysis, pebble game with labels for database serializability testing, etc., see [7]).

In connection with quantum computing, the model of reversible computations is very interesting. As the basic laws of quantum physics are reversible, also the quantum computation has to be reversible. That means, that each state of the computation has to uniquely define both the *following* and the *preceding* state of the computation.

Another motivation to examine the model of reversible computation follows from the fact, that reversible operations are not known to require any heat dissipation. With continuing miniaturisation of computing devices, reduction of the energy dissipation becomes very important. Both these reasons for studying reversible computations are mentioned in [9], [2], [5] and [4].

A modification of the standard pebble game for modelling reversible computations is the reversible pebble game. Reversible pebble game enables to analyse time and space complexity and time-space trade-offs of reversible computations.

In this paper, three basic classes of dags are considered: the chain topology, the complete binary tree topology and the butterfly topology. These topologies represent the structure of the most common problems.

It is evident, that minimal space complexity for standard pebble game on chain topology is $O(1)$, minimal time complexity is $O(n)$ and minimal space and time complexities can be achieved simultaneously. For reversible pebble game, in [4] was proved minimal space complexity on the chain topology in the form $O(\lg n)$ and upper bound on time complexity for optimal space complexity in the form $O(n^{\lg 3})$. In [1] it was introduced a pebbling strategy, that yields an upper bound of time-space tradeoff for reversible pebble game on chain in the form: space $O(\frac{k-1}{\lg k} \lg n)$ versus time $\Omega(n^{\frac{\lg(2k-1)}{\lg k}})$.

We show that optimal time for optimal space complexity cannot be $o(n \lg n)$. Further, we show the upper bound on the time-space tradeoff for reversible pebble game on chain in the form: space $O(\sqrt[k]{n})$ versus time $2^k n$.

Minimal space complexity $h + 1$ for standard pebble game on a complete binary tree of height h was proved in [6]. We show a tight space bound for reversible pebble game on a complete binary tree in the form $h + \Theta(\lg^* h)$. These results give an evidence, that more resources are needed for reversible computation in comparison with irreversible computation.

2 Preliminaries

Reversible Pebble Game is played on directed acyclic graphs. Let G be a dag. A *configuration on* G is a set of its vertices covered by pebbles. Let C be a configuration, the formula $C(v) = 1$ denotes the fact that the vertex v is covered by a pebble. Analogically, $C(v) = 0$ denotes that the vertex v is uncovered. We denote the number of pebbles used in a configuration C as $\#(C)$. An *empty configuration* on G is denoted as $\mathrm{E}(C)$. Empty configuration is a configuration without pebbles. The rules of *Reversible pebble game* are the following:

R1 A pebble can be laid on a vertex v if and only if all direct predecessors of the vertex v are covered by pebbles.

R2 A pebble can be removed from a vertex v if and only if all direct predecessors of the vertex v are covered by pebbles.

Reversible pebble game differs from standard pebble game in rule R2 – in standard pebble game, pebbles can be removed from any vertex at any time.

An ordered pair of configurations on dag G, such that the second one follows from the first one according to these rules, is called a *transition*.

For our purposes, a transition can be also a pair of two identical configurations. A *nontrivial transition* is a transition not formed by identical configurations.

Important property of a transition in a reversible pebble game is *symmetry*. From the rules of the game follows, that if (C_1, C_2) forms a transition, then also (C_2, C_1) forms a transition.

A *computation on graph* G is a sequence of configurations on G such that each successive pair forms a transition. Let $\mathcal{C}$ be a computation, $\mathcal{C}(i)$ denotes the i-th configuration in the computation $\mathcal{C}$. A computation $\mathcal{C}$ is a *complete computation*, if and only if the first and the last configurations of $\mathcal{C}$ are empty (e.g. $\#(\mathcal{C}(1)) = \#(\mathcal{C}(n)) = 0$, where n is the length of the computation $\mathcal{C}$) and for each vertex v there exists a configuration C in $\mathcal{C}$, such that v is covered in C.

We shall be interested in *space* and *time* complexities of a computation $\mathcal{C}$. *Space* of a computation $\mathcal{C}$ (denoted as $\mathrm{S}(\mathcal{C})$) is the number of pebbles needed to perform the computation – that is the maximum number of pebbles used over all configurations of $\mathcal{C}$. *Time* of a computation $\mathcal{C}$ (denoted as $\mathrm{T}(\mathcal{C})$) is the number of nontrivial transitions in $\mathcal{C}$.

The *minimal space* of the reversible pebble game on the dag G (denoted as $\mathrm{S}_{\min}(G)$) is the minimum of $\mathrm{S}(\mathcal{C})$ over all complete computations $\mathcal{C}$ on G. The *time* $\mathrm{T}(G, s)$ of the reversible pebble game on the dag G with *at most s pebbles* is the minimum of $\mathrm{T}(\mathcal{C})$ over all complete computations $\mathcal{C}$ on G such that $\mathrm{S}(\mathcal{C}) \leq s$.

Let $\mathcal{G}$ be a class of dags. Then the *minimal space function* $\mathrm{S}_{\min}(n)$ of a class $\mathcal{G}$ is the maximum of $\mathrm{S}_{\min}(G)$ over all dags in the subclass $\mathcal{G}_n$. The *time function* $\mathrm{T}(n, s)$ is the maximum of $\mathrm{T}(G, s)$ over all dags G in the subclass $\mathcal{G}_n$.

2.1 Operations on Computations

For proving upper and lower bounds on time and space complexities of the reversible pebble game, it is useful to manipulate formally with reversible computations. We will use an algebraic way to describe computations. An advantage of this approach is in high precision of the description. In this section we introduce some operations for constructing and modifying computations.

For changing state of a particular vertex in a configuration, we use the operation *Put*.

Definition 1. *Let $G = (V, E)$ be a dag, C be a configuration on G. Let $v \in V$ and $h \in \{0, 1\}$. Then* $\mathrm{Put}(C, v, h)$ *is a configuration on G defined as follows:*

- $\mathrm{Put}(C, v, h)(u) = C(u)$, *if* $u \in V$ *and* $u \neq v$;
- $\mathrm{Put}(C, v, h)(u) = h$, *if* $u \in V$ *and* $u = v$.

An important property of reversible computations is the following one: Let G be a dag, G' be a subgraph of G and $\mathcal{C}$ be a computation on G. If we remove all vertices not in G' from all configurations of $\mathcal{C}$, we obtain a reversible computation on G'. The correctness of such construction is clear – we cannot violate any rule of reversible pebble game by removing a vertex from all configurations of a computation. Another important fact is, that removing some configurations from the beginning and the end of a reversible computation does not violate a property of a reversible computation, too.

Also, we can define an operator for a "restriction" of a computation:

Definition 2. *Let* $G = (V, E)$ *be a dag,* $V' \subseteq V$. *Let* $\mathcal{C}$ *be a computation of the length* n *on* G. *A* Restriction $\mathcal{C}' = \mathrm{Rst}(\mathcal{C}, i, j, V')$ *of the computation* $\mathcal{C}$ *to an interval* $\{i \dots j\}$ $(1 \leq i \leq j \leq n)$ *and to a subgraph* $G' = (V', E \cap (V' \times V'))$ *is a computation* $\mathcal{C}'$ *of the length* $j - i + 1$ *on* G' *defined as follows:*

$$(\forall k \in \{1 \dots j - i + 1\})(\forall v \in V')\mathcal{C}'(k)(v) = \mathcal{C}(i + k - 1)(v)$$

We use a notation $\mathrm{Rst}(\mathcal{C}, i, j)$ *when no vertices should be removed (e.g.* $\mathrm{Rst}(\mathcal{C}, i, j) = \mathrm{Rst}(\mathcal{C}, i, j, V)$ *for the graph* $G = (V, E)$*).*

From the symmetry of the rules of the reversible pebble game follows, that reversing a reversible computation does not violate the reversible computation property. We can therefore define an operator *Rev*.

Definition 3. *Let* $\mathcal{C}$ *be a computation on* G *of the length* n. *Then the* reverse *of the computation* $\mathcal{C}$ *(denoted as* $\mathrm{Rev}(\mathcal{C})$*) is a computation on* G *defined as follows:*

$$(\forall i \in \{1 \dots n\})\,\mathrm{Rev}(\mathcal{C})(i) = \mathcal{C}(n + 1 - i)$$

Now we introduce operations, that are inverse to restriction in some sense.

Definition 4. *Let* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *are computations on a dag* G, *let* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *have length* n_1 *and* n_2 *respectively. Let* $\mathcal{C}_1(n_1)$ *and* $\mathcal{C}_2(1)$ *form a transition. Then the* join *of computations* C_1 *and* C_2 *(denoted as* $\mathcal{C}_1 + \mathcal{C}_2$*) is a computation on* G *of length* $n_1 + n_2$ *defined as follows:*

- $(\mathcal{C}_1 + \mathcal{C}_2)(i) = \mathcal{C}_1(i)$, *if* $i \leq n_1$
- $(\mathcal{C}_1 + \mathcal{C}_2)(i) = \mathcal{C}_2(i - n_1)$, *if* $i > n_1$

It is clear, that this definition is correct. Configurations $(\mathcal{C}_1 + \mathcal{C}_2)(n_1)$ and $(\mathcal{C}_1 + \mathcal{C}_2)(n_1 + 1)$ form a transition by assumption. All other successive pairs of configurations form transitions, because $\mathcal{C}_1$ and $\mathcal{C}_2$ are computations.

Let C be a configuration on a dag G. Then we can look at a configuration C also as at a computation of length 1, so that $\mathcal{C}(1) = C$. Therefore we can also join a computation with a configuration and vice versa.

The join of two computations is an inverse operation to restriction by removing configurations. Now we define an inverse operation to the restriction performed by removing vertices.

Definition 5. *Let $G = (V, E)$ be a dag, $V_1 \subseteq V$, $V_2 \subseteq V$, $V_1 \cap V_2 = \emptyset$. Let C be a configuration on the graph $(V_2, E \cap (V_2 \times V_2))$. Let $\{(w, v) | v \in V_1 \wedge w \in V_2 \wedge C(w) = 0 \wedge (w, v) \in E\} = \emptyset$. Let $\mathcal{C}$ be a computation of length n on the graph $(V_1, E \cap (V_1 \times V_1))$. The* computation $\mathcal{C}$ merged with the configuration C *(denoted as $\mathcal{C} \cdot C$) is a computation on the graph $(V_1 \cup V_2, E \cap ((V_1 \cup V_2) \times (V_1 \cup V_2)))$ of length n defined as follows:*

- $(\mathcal{C} \cdot C)(i)(v) = \mathcal{C}(i)(v)$, *if* $v \in V_1$
- $(\mathcal{C} \cdot C)(i)(v) = C(v)$, *if* $v \in V_2$

This definition is clearly correct. By adding the same configuration to all configurations of some computation $\mathcal{C}$, it is only one way to violate the rules of the reversible pebble game: if some of the added direct predecessors of a vertex, which the pebble is laid on or removed from, are not pebbled. But this is prohibited by the assumption of definition.

Any computation on a graph G can be applied on any graph G' that is isomorphic with G. The *application* of a computation can be defined as follows:

Definition 6. *Let $\mathcal{C}$ be a computation of length n on a dag G and G' be a dag isomorphic with G. Let φ is the isomorphism between G' and G. Then a computation $\mathcal{C}$* applied to *the graph G' (denoted as $\mathcal{C}|G'$) is a computation on G' of length n such that $(\mathcal{C}|G')(i)(v) = \mathcal{C}(i)(\varphi(v))$ for all $1 \leq i \leq n$ and for all vertices v of G'.*

3 Chain Topology

The simplest topology for a pebble game is a *chain*. Chain with n vertices (denoted as $Ch(n)$) is a dag $Ch(n) = (V, E)$, where $V = \{1 \ldots n\}$ and $E = \{(i-1, i) | i \in \{2 \ldots n\}\}$. This topology is an abstraction of a simple straightforward computation, where the result of step $n + 1$ can be computed only from the result of step n.

In this section we discuss optimal space complexity for a reversible pebble game on the chain topology – the minimal space function $\mathrm{S}_{\min}(n)$ for Ch, where the subclass Ch_n contains only a chain $Ch(n)$. We will discuss also partial lower and upper bounds for optimal time and space complexities – the time function $\mathrm{T}(n, \mathrm{S}_{\min}(n))$ and the upper bound of the time-space tradeoff for the chain topology.

3.1 Optimal Space for the Chain Topology

For determining space complexity of the reversible pebble game on the chain topology we will examine the maximum length of the chain, that can be pebbled by p pebbles. We denote this length as $\mathrm{S}^{-1}(p)$. It holds, that $\mathrm{S}^{-1}(p) = \max\{m | (\exists \mathcal{C} \in \mathbb{C}_{Ch(m)})\ \mathrm{S}(\mathcal{C}) \leq p\}$, where $\mathbb{C}_{Ch(m)}$ is the set of all computations on the graph $Ch(m)$.

Reversible pebble game on the chain topology was studied in connection with reversible simulation of irreversible computation. C. H. Bennett suggested in [1] a pebbling strategy, whose special case has space complexity $\Theta(\lg n)$. Space optimality of this algorithm was proved in [5] and [4]. This result is formulated in following theorem.

Theorem 1. *It holds that* $\mathrm{S}^{-1}(p) = 2^p - 1$. *Therefore for minimal space function of chain topology* $\mathrm{S}_{\min}(n)$ *it holds*

$$\mathrm{S}_{\min}(n) = \Theta(\lg n)$$

3.2 Optimal Time and Space for the Chain Topology

In this section we present upper and partial lower bounds on time for space optimal reversible pebble game played on a chain topology.

We will use two auxiliary lemmas. Their proofs are not difficult and are left out due to space reasons.

Lemma 1. *Let* $\mathcal{C}$ *be a complete computation of length* l *on* $Ch(n)$, $\mathrm{S}(\mathcal{C}) = \mathrm{S}_{\min}(n)$, $\mathrm{T}(\mathcal{C}) = \mathrm{T}(n, \mathrm{S}_{\min}(n))$. *Let* $i = \min\{i | i \in \{1 \dots l\} \wedge \mathcal{C}(i)(n) = 1\}$. *Then it holds that*

$$\mathrm{T}(\mathrm{Rst}(\mathcal{C}, 1, i)) = \mathrm{T}(\mathrm{Rst}(\mathcal{C}, i, l)) = \frac{\mathrm{T}(\mathcal{C})}{2}$$

Lemma 2. *Let* $\mathcal{C}$ *be a complete computation on* $Ch(\mathrm{S}^{-1}(p+1))$ *such that* $\mathrm{S}(\mathcal{C}) = p + 1$. *It holds that*

$$\max\{\min\{j | j \in \{1 \dots n\} \wedge \mathcal{C}(i)(j) = 1\} | i \in \{1 \dots l\}\} = \mathrm{S}^{-1}(p) + 1$$

Now we prove the upper and partial lower bound on time for space optimal pebble game:

Theorem 2. $\mathrm{T}(\mathrm{S}^{-1}(p+1), p+1) \geq 2\mathrm{S}^{-1}(p) + 2 + 2\mathrm{T}(\mathrm{S}^{-1}(p), p)$

Proof. Let $\mathcal{C}$ be a time optimal complete computation on $Ch(\mathrm{S}^{-1}(p+1))$, such that $\mathrm{S}(\mathcal{C}) = p+1$. Let l be the length of $\mathcal{C}$. Clearly $\mathrm{T}(\mathcal{C}) = \mathrm{T}(\mathrm{S}^{-1}(p+1), p+1)$. We prove, that $\mathrm{T}(\mathcal{C}) \geq 2\mathrm{S}^{-1}(p) + 2 + 2\mathrm{T}(\mathrm{S}^{-1}(p), p)$ holds.

Let $n = \mathrm{S}^{-1}(p)$, $G_1 = (\{n+1\}, \emptyset)$ and G_2 be a graph obtained from $Ch(n)$ by renaming vertices to $n+2 \dots 2n+1 = \mathrm{S}^{-1}(p+1)$. Let $i = \min\{i | i \in \{1 \dots l\} \wedge \mathcal{C}(i)(2n+1) = 1\}$. By Lemma 1 it holds $\mathrm{T}(\mathrm{Rst}(\mathcal{C}, 1, i)) = \frac{1}{2}\mathrm{T}(\mathcal{C})$. From Lemma 2 follows, that $\mathrm{Rst}(\mathcal{C}, k, k, \{1 \dots n+1\}) \neq \mathrm{E}(Ch(n)) \cdot \mathrm{E}(G_1))$ for all k and that there exists j such that $\mathrm{Rst}(\mathcal{C}, j, j, \{1 \dots n+1\}) = \mathrm{E}(Ch(n)) \cdot \mathrm{Put}(\mathrm{E}(G_1), n+1, 1)$. W.l.o.g. we can assume $j \leq i$ (otherwise we can replace $\mathcal{C}$ by $\mathrm{Rev}(\mathcal{C})$). Let k be a configuration such that $\mathcal{C}(k-1)(n+1) = 0$ and $(\forall q)(k \leq q \leq j)\mathcal{C}(q)(n+1) = 1)$. Clearly $\mathcal{C}(k-1)(n) = \mathcal{C}(k)(n) = 1$.

Now consider the computation $\mathcal{C}_2 = \mathrm{Rst}(\mathcal{C}, 1, k-1, \{1 \dots n\}) \cdot \mathrm{E}(G_1) \cdot \mathrm{E}(G_2) + \mathrm{Rst}(\mathcal{C}, k, j, \{1 \dots n\}) \cdot \mathrm{Put}(\mathrm{E}(G_1), n+1, 1) \cdot \mathrm{E}(G_2) + \mathrm{Rst}(\mathcal{C}, 1, i, \{n+2 \dots 2n+1\}) \cdot \mathrm{E}(Ch(n)) \cdot \mathrm{Put}(\mathrm{E}(G_1), n+1, 1)$. Clearly $\mathrm{S}(\mathcal{C}_2) \leq \mathrm{S}(\mathcal{C})$. Also, $\mathcal{C}_2 + \mathrm{Rev}(\mathcal{C}_2)$ is complete on $Ch(2n+1)$ and $\mathrm{T}(\mathcal{C}_2) \leq \mathrm{T}(\mathrm{Rst}(\mathcal{C}, 1, i))$.

It is clear, that $\mathrm{T}(\mathrm{Rst}(\mathcal{C}, 1, k-1, \{1 \dots n\})) \geq n$ – we cannot pebble n vertices with time less than n. $\mathrm{Rev}(\mathrm{Rst}(\mathcal{C}, k, j, \{1 \dots n\}))+\mathrm{Rst}(\mathcal{C}, k, j, \{1 \dots n\})$ is a space optimal complete computation on $Ch(n)$, therefore $\mathrm{T}(\mathrm{Rst}(\mathcal{C}, k, j, \{1 \dots n\})) \geq \frac{1}{2}\mathrm{T}(n, \mathrm{S}_{\min}(n)) = \frac{1}{2}\mathrm{T}(\mathrm{S}^{-1}(p), p)$. Analogically, $\mathrm{T}(\mathrm{Rst}(\mathcal{C}, 1, i, \{n+2 \dots 2n+1\})) \geq \frac{1}{2}\mathrm{T}(n, \mathrm{S}_{\min}(n)) = \frac{1}{2}\mathrm{T}(\mathrm{S}^{-1}(p), p)$.

From these inequalities follows, that $\mathrm{T}(\mathrm{Rst}(\mathcal{C}, 1, i)) \geq n+1+2\frac{1}{2}\mathrm{T}(n, \mathrm{S}_{\min}(n))$. Therefore $\mathrm{T}(\mathcal{C}) \geq 2n + 2 + 2\mathrm{T}(n, \mathrm{S}_{\min}(n)) = 2\mathrm{S}^{-1}(p) + 2 + 2\mathrm{T}(\mathrm{S}^{-1}(p), p)$. □

Corollary 1. $\mathrm{T}(n, \mathrm{S}_{\min}(n)) = O(n^{\log_2 3})$, $\mathrm{T}(n, \mathrm{S}_{\min}(n)) \neq o(n \lg n)$

Proof. The upper bound was presented in [4]. By solving recurrent inequality proved in preceding theorem, we obtain that $T(n, \mathrm{S}_{\min}(n)) = \Omega(n \lg n)$ for $n = 2^p - 1$. Since this function is a restriction of $T(n, \mathrm{S}_{\min}(n))$ for integer n, function $T(n, \mathrm{S}_{\min}(n))$ cannot be $o(n \lg n)$. □

3.3 Upper Bound on Time-Space Tradeoff for Chain Topology

In the previous section it was analysed time complexity of reversible pebbling for space optimal computations. Now we discuss the time complexity for computations, that are not space optimal.

It is obvious, that for any complete computation $\mathcal{C}$ on $Ch(n)$ it holds $\mathrm{T}(\mathcal{C}) \geq 2n$, because each vertex has to be at least one time pebbled and at least one time unpebbled. It is also easy to see, that space of such computation is exactly n.

Now we will analyse space complexity of complete computations on $Ch(n)$ that are running in time at most $c \cdot n$. Let $\mathrm{S}^{-1}(c, k) = \max\{n | \exists \mathcal{C} \in \mathbb{C}_{Ch(n)}\ \mathrm{S}(\mathcal{C}) \leq k \wedge \mathrm{T}(\mathcal{C}) \leq cn\}$, where $\mathbb{C}_{Ch(n)}$ is the set of all complete computations on $Ch(n)$.

Theorem 3. *For a fixed k, it holds* $\mathrm{S}^{-1}(2^k, p) = \Omega(p^k)$.

Proof. We prove a statement $\mathrm{S}^{-1}(2^k, p) \geq c(k)p^k$ by induction on k and p. Let $c(1) = 1$. The base case $\mathrm{S}^{-1}(2^1, p) \geq p$ holds trivially. (It is easy to make a complete computation $\mathcal{C}$ on $Ch(p)$ satisfying $\mathrm{S}(\mathcal{C}) = p$ and $\mathrm{T}(\mathcal{C}) = 2p$.)

Assume by the induction hypothesis that it holds $(\forall k' < k)(\forall p')\mathrm{S}^{-1}(2^{k'}, p') \geq c(k')p'^{k'}$ and $(\forall p' < p)\mathrm{S}^{-1}(2^k, p') \geq c(k)p'^k$. We prove, that $\mathrm{S}^{-1}(2^k, p) \geq c(k)p^k$ holds.

Let $\mathcal{C}_1$ be a complete computation on $Ch(\mathrm{S}^{-1}(2^{k-1}, p-1))$, $\mathrm{S}(\mathcal{C}_1) \leq p-1$, $\mathrm{T}(\mathcal{C}_1) \leq 2^{k-1}\mathrm{S}^{-1}(2^{k-1}, p-1)$. Denote the length of $\mathcal{C}_1$ by l_1. Clearly there exists m such that $\mathcal{C}_1(m)(\mathrm{S}^{-1}(2^{k-1}, p-1)) = 1$.

Let $\mathcal{C}_2$ be a complete computation on $Ch(\mathrm{S}^{-1}(2^k, p-1))$, $\mathrm{S}(\mathcal{C}_2) \leq p-1$, $\mathrm{T}(\mathcal{C}_2) \leq 2^k\mathrm{S}^{-1}(2^k, p-1)$.

Let $G_1 = (\{\mathrm{S}^{-1}(2^{k-1}, p-1) + 1\}, \emptyset)$. Let G_2 is a graph obtained from $Ch(\mathrm{S}^{-1}(2^k, p-1))$ by renaming its vertices to $\mathrm{S}^{-1}(2^{k-1}, p-1)+2, \dots, \mathrm{S}^{-1}(2^{k-1}, p-1)+1+\mathrm{S}^{-1}(2^k, p-1)$. Now assume the following computation $\mathcal{C}_3 = \mathrm{Rst}(\mathcal{C}_1, 1, m) \cdot \mathrm{E}(G_1) \cdot \mathrm{E}(G_2) + \mathrm{Rst}(\mathcal{C}_1, m, l_1) \cdot \mathrm{Put}(\mathrm{E}(G_1), \mathrm{S}^{-1}(2^{k-1}, p-1) + 1, 1) \cdot \mathrm{E}(G_2) +$

$(\mathcal{C}_2|G_2)\cdot \mathrm{E}(Ch(\mathrm{S}^{-1}(2^{k-1},p-1)))\cdot \mathrm{Put}(\mathrm{E}(G_1),\mathrm{S}^{-1}(2^{k-1},p-1)+1,1)+\mathrm{Rev}(\mathrm{Rst}(\mathcal{C}_1,m,l_1))\cdot \mathrm{Put}(\mathrm{E}(G_1),\mathrm{S}^{-1}(2^{k-1},p-1)+1,1)\cdot \mathrm{E}(G_2)+\mathrm{Rev}(\mathrm{Rst}(\mathcal{C}_1,1,m))\cdot \mathrm{E}(G_1)\cdot \mathrm{E}(G_2)$.

Clearly $\mathcal{C}_3$ is a complete computation on $Ch(\mathrm{S}^{-1}(2^{k-1},p-1)+1+\mathrm{S}^{-1}(2^k,p-1))$ satisfying $\mathrm{S}(\mathcal{C}_3)\le p$ and $\mathrm{T}(\mathcal{C}_3)\le 2\mathrm{T}(\mathcal{C}_1)+2+\mathrm{T}(\mathcal{C}_2)\le 2^k\mathrm{S}^{-1}(2^{k-1},p-1)+2+2^k\mathrm{S}^{-1}(2^k,p-1)\le 2^k(\mathrm{S}^{-1}(2^{k-1},p-1)+1+\mathrm{S}^{-1}(2^k,p-1))$. Therefore $\mathrm{S}^{-1}(2^k,p)\ge \mathrm{S}^{-1}(2^{k-1},p-1)+1+\mathrm{S}^{-1}(2^k,p-1)$.

By induction hypothesis we have $\mathrm{S}^{-1}(2^k,p)\ge c(k-1)(p-1)^{k-1}+c(k)(p-1)^k$. For a suitable value of $c(k)$ (we can choose $c(k)=\frac{c(k-1)}{2^k}$) it holds that $c(k-1)(p-1)^{k-1}+c(k)(p-1)^k\ge c(k)p^k$. Also there exists $c(k)$ such that $\mathrm{S}^{-1}(2^k,p)\ge c(k)p^k$. □

Corollary 2. *Let k be fixed. Then $O(\sqrt[k]{n})$ pebbles are sufficient for a complete computation on $Ch(n)$ with time $O(2^k n)$.*

Another upper bound of the time-space tradeoff for the reversible pebbling on chain topology can be obtained by using Bennett's pebbling strategy introduced in [1]. Since this strategy pebbles chain of length k^n with $n(k-1)+1$ pebbles in time $(2k-1)^n$, it yields time-space tradeoff in the form: space $O(\frac{k-1}{\lg k}\lg n)$ versus time $\Omega(n^{\frac{\lg(2k-1)}{\lg k}})$.

4 Binary Tree Topology

In this section we will discuss space complexity of reversible pebble game on complete binary trees. A *complete binary tree* of height 1 (denoted as $\mathrm{Bt}(1)$) is a graph containing one vertex and no edges. A complete binary tree of height $h>1$ (denoted as $\mathrm{Bt}(h)$) consists of a root vertex and two subtrees, that are complete binary trees of height $h-1$.

This topology represents a class of problems, where the result can be computed from two different subproblems.

We denote the root vertex of $Bt(h)$ as $\mathrm{R}(Bt(h))$, the left subtree of $Bt(h)$ as $\mathrm{Lt}(Bt(h))$ and the right subtree of $Bt(h)$ as $\mathrm{Rt}(Bt(h))$.

As mentioned in section 2, we denote the minimal number of pebbles needed to perform a complete computation on $\mathrm{Bt}(h)$ as $\mathrm{S}_{\min}(h)$. In the sequel we also consider the minimal number of pebbles needed to perform a computation from the empty configuration to a configuration, where only the root is pebbled.

Definition 7. *Let $\mathcal{C}$ be a computation of length l on* $\mathrm{Bt}(h)$. *Let* $\mathcal{C}(1)=\mathrm{E}(\mathrm{Bt}(h))$ *and* $\mathcal{C}(l)=\mathrm{Put}(\mathrm{E}(\mathrm{Bt}(h)),\mathrm{R}(\mathrm{Bt}(h)),1)$. *Then $\mathcal{C}$ is called a* semicomplete computation.

The minimal number of pebbles needed to perform a semicomplete computation on $\mathrm{Bt}(h)$ (e.g. $\min\{\mathrm{S}(\mathcal{C})\}$, where $\mathcal{C}$ is a semicomplete computation) will be denoted as $\mathrm{S}'_{\min}(h)$.

We will use the following inequalities between $\mathrm{S}_{\min}(h)$ and $\mathrm{S}'_{\min}(h)$. Their proofs are not difficult and are left out due to space reasons.

Lemma 3. $\mathrm{S}_{\min}(h) + 1 \geq \mathrm{S}'_{\min}(h) \geq \mathrm{S}_{\min}(h)$

Lemma 4. $\mathrm{S}'_{\min}(h+1) = \mathrm{S}_{\min}(h) + 2$

4.1 Tight Space Bound for Binary Tree Topology

From the previous lemmas follows, that $\mathrm{S}'_{\min}(h)$ equals to h plus the number of such $i < h$, that $\mathrm{S}'_{\min}(i) = \mathrm{S}_{\min}(i)$. In the following considerations we use a function $\mathrm{S}'^{-1}_{\min}$. The value $h = \mathrm{S}'^{-1}_{\min}(p)$ denotes the maximal height of binary tree that can be pebbled by a semicomplete computation, that uses at most $h + p$ pebbles. Formally, $\mathrm{S}'^{-1}_{\min}(p) = \max\{h | \exists C \in \ \mathrm{Sc}_h \wedge \mathrm{S}(\mathcal{C}) = h+p\}$, where Sc_h is the set of all semicomplete computations on $\mathrm{Bt}(h)$. From the definition of $\mathrm{S}'^{-1}_{\min}(p)$ follows, that $\mathrm{S}'_{\min}(h) = h + (\mathrm{S}'^{-1}_{\min})^{-1}(h)$.

Now we prove the upper (lower) bound of $\mathrm{S}'^{-1}_{\min}(p)$. From that follows lower (upper) bound of $\mathrm{S}'_{\min}(h)$ and therefore also lower (upper) bound of $\mathrm{S}_{\min}(h)$ respectively.

Lemma 5. *Let* $h' = \mathrm{S}'^{-1}_{\min}(p)$, $h = \mathrm{S}'^{-1}_{\min}(p+1)$. *Then the following inequality holds:*

$$h - h' - 1 \leq 2^{h'+p+1} - 1$$

Proof. A configuration on a binary tree is called *opened*, if there exists a path from the root to some leaf of the tree, such that no pebble is laid on this path. Otherwise, the configuration is called *closed*.

From the assumption $h = \mathrm{S}'^{-1}_{\min}(p+1)$ follows, that there exists some semicomplete computation $\mathcal{C}$ of length l on $\mathrm{Bt}(h)$, such that $\mathrm{S}(\mathcal{C}) = h + p + 1$. Let i be the first configuration of $\mathcal{C}$, such that $\mathcal{C}(j)(\mathrm{R}(\mathrm{Bt}(h))) = 1$ for any $j \geq i$ (e.g. $i = \min\{i | (\forall j \geq i)\ \mathcal{C}(j)(\mathrm{R}(\mathrm{Bt}(h))) = 1\}$).

Because $\mathcal{C}$ is a reversible computation, $\mathcal{C}(i)(\mathrm{R}(\mathrm{Lt}(\mathrm{Bt}(h)))) = \mathcal{C}(i)(\mathrm{R}(\mathrm{Rt}(\mathrm{Bt}(h)))) = 1$. Therefore $\mathrm{Put}(\mathcal{C}(i), \mathrm{R}(\mathrm{Bt}(h)), 0)$ is a closed configuration. Because $\mathrm{Put}(\mathcal{C}(l), \mathrm{R}(\mathrm{Bt}(h)), 0) = \mathrm{E}(\mathrm{Bt}(h))$, this configuration is opened. Let j be the minimal number such that $j \geq i$ and $\mathrm{Put}(\mathcal{C}(j), \mathrm{R}(\mathrm{Bt}(h)), 0)$ is opened.

Because $\mathrm{Put}(\mathcal{C}(j), \mathrm{R}(\mathrm{Bt}(h)), 0)$ is opened and $\mathrm{Put}(\mathcal{C}(j-1), \mathrm{R}(\mathrm{Bt}(h)), 0)$ is closed and $\mathcal{C}$ is a reversible computation, there exists exactly one path in $\mathcal{C}(j)$ from the root to a leaf, such that no pebble is laid on it. Without loss of generality we can assume, that this path is $\mathrm{R}(\mathrm{Bt}(h)), \mathrm{R}(\mathrm{Rt}(\mathrm{Bt}(h))), \mathrm{R}(\mathrm{Rt}^2(\mathrm{Bt}(h))), \ldots, \mathrm{R}(\mathrm{Rt}^{h-1}(\mathrm{Bt}(h)))$.

Now we prove, that for each k, $h \geq k \geq h' + 2$, and for each p, $i \leq p < j$, it holds that $\#(\mathcal{C}(p)(\mathrm{Lt}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h))))) > 0$.

Assume, that this conjecture does not hold. Let k be the maximal number such that violates this conjecture. Let p be the maximal number such that $i \leq p < j$ and $\#(\mathcal{C}(p)(\mathrm{Lt}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h))))) = 0 \vee \#(\mathcal{C}(p)(\mathrm{Rt}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h))))) = 0$. Without loss of generalit y, let $\#(\mathcal{C}(p)(\mathrm{Lt}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h))))) = 0$. Because $\mathrm{Put}(\mathcal{C}(p), \mathrm{R}(\mathrm{Bt}(h)), 0)$ is closed, in a configuration $\mathcal{C}(p)$ is pebbled at least one vertex from $\mathrm{R}(\mathrm{Rt}(\mathrm{Bt}(h)))$, $\mathrm{R}(\mathrm{Rt}^2(\mathrm{Bt}(h)))$, $\ldots$, $\mathrm{R}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h)))$. In a configuration $\mathcal{C}(j)$ are all these vertices unpebbled. Let q be the minimal number such

that $q > p$ and all these vertices are unpebbled in $\mathcal{C}(q)$. Because $\mathcal{C}$ is a reversible computation, $\mathcal{C}(q-1)(\mathrm{R}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h)))) = 1$, $\mathcal{C}(q)(\mathrm{R}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h)))) = 0$ and $\mathcal{C}(q)(\mathrm{R}(\mathrm{Lt}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h))))) = 1$. Now consider the computation $\mathcal{C}' = \mathrm{Rst}(\mathcal{C}, p, q, \mathrm{Lt}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h))))$. Computation $\mathcal{C}' + \mathrm{Rev}(\mathcal{C}')$ is a complete computation on $\mathrm{Lt}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h)))$ (this graph is isomorphic to $\mathrm{Bt}(k-1)$). Space of this computation is at most $\mathrm{S}(\mathcal{C}' + \mathrm{Rev}(\mathcal{C}')) \leq \mathrm{S}(\mathcal{C}) - (3 + h - k) = k + p - 2$. From our assumption follows, that space for any semicomplete computation on $\mathrm{Bt}(k-1)$ is at least $k+p$. From Lemma 3 follows, that the space for any complete computation on $\mathrm{Bt}(k-1)$ is at least $k + p - 1$, what is a contradiction.

Now consider $\mathcal{C}_2 = \mathrm{Rst}(\mathcal{C}, i, j, \mathrm{R}(\mathrm{Rt}(\mathrm{Bt}(h))) \cup \mathrm{R}(\mathrm{Rt}^2(\mathrm{Bt}(h))) \cup \ldots \cup \mathrm{R}(\mathrm{Rt}^{h-h'-1}(\mathrm{Bt}(h))))$. It is a computation on a graph isomorphic to $Ch(h-h'-1)$. In the first configuration of $\mathcal{C}_2$, vertex $\mathrm{R}(\mathrm{Rt}(\mathrm{Bt}(h)))$ is pebbled. In the last configuration of $\mathcal{C}_2$, no vertex is pebbled. Therefore $\mathrm{Rev}(\mathcal{C}_2) + \mathcal{C}_2$ is a complete computation on a graph isomorphic to $Ch(h - h' - 1)$.

Because for each k, $h \geq k \geq h' + 2$, and for each p, $i \leq p \leq j$, it holds that $\#(\mathcal{C}(p)(\mathrm{Lt}(\mathrm{Rt}^{h-k}(\mathrm{Bt}(h))))) > 0$ and $\mathcal{C}(p)(\mathrm{R}(\mathrm{Bt}(h))) = 1$, we can estimate upper bound for space of $\mathcal{C}_2$: $\mathrm{S}(\mathcal{C}_2) \leq (h+p+1)-(1+h-h'-1) = h'+p+1$. Using space upper bound for chain topology (Theorem 1) we have $h - h' - 1 \leq 2^{h'+p+1} - 1$. □

Lemma 6. *Let* $h' = \mathrm{S}'^{-1}_{\min}(p)$, $h = \mathrm{S}'^{-1}_{\min}(p+1)$. *Then the following inequality holds:*

$$h - h' - 1 \geq 2^{h'+p-2} - 1$$

Proof. We prove by induction, that for each $k \in \{h' + 1 \ldots h' + 2^{h'+p-2}\}$ there exists a semicomplete computation $\mathcal{C}$ on $Bt(k)$ such that $\mathrm{S}(\mathcal{C}) \leq k + p + 1$. This implies, that $h \geq h' + 2^{h'+p-2}$.

The base case is $k = h' + 1$. By assumption there exists a semicomplete computation $\mathcal{C}$ on $\mathrm{Bt}(h')$ such that $\mathrm{S}(\mathcal{C}) = h' + p$. After applying $\mathcal{C}$ to $\mathrm{Lt}(\mathrm{Bt}(k))$ and $\mathrm{Rt}(\mathrm{Bt}(k))$, pebbling $\mathrm{R}(\mathrm{Bt}(k))$ and applying reversed $\mathcal{C}$ to $\mathrm{Lt}(\mathrm{Bt}(k))$ and $\mathrm{Rt}(\mathrm{Bt}(k))$ we obtain a semicomplete computation on $\mathrm{Bt}(k)$ that uses at most $h' + p + 2 = k + p + 1$ pebbles.

Now assume that the induction hypothesis holds for each $i \in \{h'+1 \ldots k-1\}$. We construct a computation $\mathcal{C}$ on $\mathrm{Bt}(k)$ as follows: At first we apply semicomplete computations on $\mathrm{Lt}(\mathrm{Bt}(k))$, $\mathrm{Lt}(\mathrm{Rt}(\mathrm{Bt}(k)))$, …, $\mathrm{Lt}(\mathrm{Rt}^{k-h'-2}(\mathrm{Bt}(k)))$, $\mathrm{Lt}(\mathrm{Rt}^{k-h'-1}(\mathrm{Bt}(k)))$, $\mathrm{Rt}^{k-h'}(\mathrm{Bt}(k))$ sequentially. By induction hypothesis, the space of a semicomplete computation on $\mathrm{Lt}(\mathrm{Rt}^{i}(\mathrm{Bt}(k)))$ is less than or equal to $(k-i-1)+p+1$ for $i \leq k-h'-2$. By assumption, the space of a semicomplete computation on $\mathrm{Lt}(\mathrm{Rt}^{k-h'-1}(\mathrm{Bt}(k)))$ and $\mathrm{Rt}(\mathrm{Rt}^{k-h'-1}(\mathrm{Bt}(k)))$ is less than or equal to $h' + p$. Therefore space of this part of $\mathcal{C}$ is less than or equal to $k + p$.

In the second part of $\mathcal{C}$, we perform a space optimal complete computation on a chain consisting of vertices $\mathrm{R}(\mathrm{Bt}(k))$, $\mathrm{R}(\mathrm{Rt}(\mathrm{Bt}(k)))$, …, $\mathrm{R}(\mathrm{Rt}^{k-h'-1}(\mathrm{Bt}(k)))$. Due to the Theorem 1, space of this part is less than or equal to $\lceil \log_2(k-h'+1) \rceil + k - h' + 1$. Because $k \leq h' + 2^{h'+p-2}$, it holds $\lceil \log_2(k-h'+1) \rceil + k - h' + 1 \leq k + p$.

The third part of the computation $\mathcal{C}$ is the reversed first part.

Also, $\mathcal{C}$ is a complete computation on $\mathrm{Bt}(k)$ and $\mathrm{S}(\mathcal{C}) \leq k+p$. Hence, $\mathrm{S}_{\min}(k) \leq k+p$. Using Lemma 3, $\mathrm{S}'_{\min}(k) \leq k+p+1$. Therefore there exists a semicomplete computation on $\mathrm{Bt}(k)$ with space less than $k+p+1$. □

Lemma 7. *For $p \geq 2$ it holds that $2^{\mathrm{S}'^{-1}_{\min}(p)} \leq \mathrm{S}'^{-1}_{\min}(p+1) \leq 2^{4\,\mathrm{S}'^{-1}_{\min}(p)}$.*

Proof. Let $h' = \mathrm{S}'^{-1}_{\min}(p)$, $h = \mathrm{S}'^{-1}_{\min}(p+1)$. From Lemma 5 follows $h - h' \leq 2^{h'+p+1}$, what is equivalent to $\mathrm{S}'^{-1}_{\min}(p+1) \leq 2^{\mathrm{S}'^{-1}_{\min}(p)+p+1} + \mathrm{S}'^{-1}_{\min}(p)$.

From the definition of $\mathrm{S}'^{-1}_{\min}$ and from Lemma 3 and Lemma 4 trivially follows, that $\mathrm{S}'^{-1}_{\min}(p) \geq 2p$. Therefore $2^{\mathrm{S}'^{-1}_{\min}(p)+p+1} + \mathrm{S}'^{-1}_{\min}(p) \leq 2^{4\,\mathrm{S}'^{-1}_{\min}(p)}$ for $p \geq 1$. Also, the second inequality holds.

From Lemma 6 follows $h - h' \geq 2^{h'+p-2}$, what is equivalent to $\mathrm{S}'^{-1}_{\min}(p+1) \geq 2^{\mathrm{S}'^{-1}_{\min}(p)+p-2} + \mathrm{S}'^{-1}_{\min}(p)$. Therefore for $p \geq 2$ it holds $\mathrm{S}'^{-1}_{\min}(p+1) \geq 2^{\mathrm{S}'^{-1}_{\min}(p)}$. □

Theorem 4. $\mathrm{S}_{\min}(h) = h + \Theta(\lg^*(h))$

Proof. From the previous lemma follows, that $\mathrm{S}'^{-1}_{\min}(p) = O(\overbrace{16^{16^{\cdots 16}}}^{p})$ and that $\mathrm{S}'^{-1}_{\min}(p) = \Omega(\overbrace{2^{2^{\cdots 2}}}^{p})$. Because $\mathrm{S}'_{\min}(h) = h + (\mathrm{S}'^{-1}_{\min})^{-1}(h)$, it holds that $\mathrm{S}'_{\min}(h) = h + \Omega(\lg^*(h))$ and $\mathrm{S}'_{\min}(h) = h + O(\lg^*(h))$. Therefore $\mathrm{S}'_{\min}(h) = h + \Theta(\lg^*(h))$. From Lemma 3 follows, that $\mathrm{S}_{\min}(h) = h + \Theta(\lg^*(h))$. □

4.2 Extension to Butterflies

Butterfly graphs create important class of graphs to study, as they share superconcentrator property and the butterflies form inherent structure of some important problems in numerical computations, as discrete FFT.

A butterfly graph of order d is a graph $G = (V, E)$, where $V = \{1 \dots d\} \times \{0 \dots 2^{d-1} - 1\}$ and $E = \{((i,j),(i+1, j \operatorname{xor} 2^{i-1}))|1 \leq i < d, 0 \leq j \leq 2^{d-1} - 1\}$. This graph can be decomposed into 2^{d-1} complete binary trees of height d. The root of i-th tree is vertex $(1,i)$ and this tree contains all vertices, that can be reached from the root.

The decomposition property implies, that the minimal space complexity of a complete computation on butterfly graph of order d cannot be lower than the minimal space complexity on a complete binary tree of height d (otherwise we can restrict a complete computation on butterfly to any binary tree to obtain a contradiction).

On the other side, by sequentially applying complete computations to all binary trees obtained by decomposition of the butterfly graph, we obtain a complete computation on it. Also, we can construct a complete computation on a butterfly graph of order d with space complexity equal to minimal space complexity of the binary tree of height d. Therefore the minimal space complexity of the butterfly topology equals to the minimal space complexity of the binary tree topology (e.g. the minimal space for a butterfly graph of order d is $d + \Theta(\lg^*(d))$).

5 Conclusion

In this paper we have analysed an abstract model for reversible computations – a reversible pebble game. We have described a technique for proving time and space complexity bounds for this game and presented a tight optimal space bound for a chain topology, upper and partial lower bounds on time of optimal space for a chain topology, an upper bound on time-space tradeoff for a chain topology and a tight optimal space bound for a binary tree topology. These results implies, that reversible computations require more resources than standard irreversible computations. (For a space complexity of a chain topology it is $\Theta(1)$ vs. $\Theta(\lg n)$ and for a space complexity of a binary tree topology it is $h + \Theta(\log^*(h))$ vs. $h + \Theta(1)$.)

For further research, it would be interesting to examine the time complexity of the reversible pebble game for tree and butterfly topology and to consider other important topologies, for example pyramids.

Acknowledgements

I would like to thank Peter Ružička for his valuable advice and consultations.

References

1. Bennett, C. H.: Time-space trade-offs for reversible computation. SIAM J. Comput., 18(1989), pp. 766–776
2. Buhrman, H., Tromp J. and Vitányi, P.: Time and space bounds for reversible simulation. Proc. ICALP 2001, LNCS 2076, Springer-Verlag, 2001
3. Levine, R. Y. and Sherman, A. T.: A note on Bennett's time-space tradeoff for reversible computation. SIAM J. Computing, 19(4):673–677, 1990
4. Li, M. and Vitányi P. M. B.: Reversibility and adiabatic computation: trading time and space for energy. Proceedings of the Royal Society of London Ser. A, 452:1–21, 1996.
5. Li, M. and Vitányi P. M. B.: Reversible simulation of irreversible computation. Proc. 11th IEEE Conference on Computational Complexity, Philadelphia, Pennsylvania, May 24–27, 1996
6. Paterson, M. S. and Hewitt, C. E.: Comparative Schematology, MAC Conf. on Concurrent Systems and Parallel Computation, 1970, pp. 119–127
7. Ružička, P.: Pebbling – The Technique for Analyzing Computation Efficiency. SOFSEM'89, 1989, pp. 205–224 (see also Information Systems 18:287–313, 1990)
8. Ružička, P. and Waczulík, J.: On Time-Space Trade-Offs in Dynamic Graph Pebbling. MFCS'93, LNCS 711, Springer-Verlag, 1993, pp. 671–681
9. Williams, R.: Space-Efficient Reversible Simulations, DIMACS REU report, July 2000
10. Zavarský, A.: On the Cost of Reversible Computations: Time-Space Bounds on Reversible Pebbling. Manuscript, 1998

The HiQoS Rendering System

Tomas Plachetka, Olaf Schmidt, and Frank Albracht

University of Paderborn, Department of Computer Science, Fürstenallee 11,
D-33095 Paderborn, Germany

Abstract. Simulation of global illumination in 3D scenes is a computationally expensive task. One of the goals of the project *HiQoS* (High Performance Multimedia Services with Quality of Service Guarantees) was to develop and test a prototype of an e-commerce system which simulates realistic lighting of large scenes on high performance parallel computers. The system, although tailored to the needs of this specific application, is very generic and exhibits metacomputing features: 1.the access to high performance computers is fully transparent to the user; 2.the modular architecture of the system allows to dynamically add or remove computing resources in geographically different computing centers. The prototype of the proposed system was evaluated in the industrial contexts of architectural visualization and film production. This paper summarizes scientific and technical problems which arose during the project as well as their solutions and engineering decisions.

1 Introduction

Photo-realistic visualization of 3D models is important in many industrial areas, for instance in architecture, entertainment industry and film production. The requirements to the level of realism as well as the complexity of the models grow very fast and so do the requirements to the performance of the systems used for the visualization. The computing power needed for the synthesis of photo-realistic images (*rendering* in this paper) in a reasonable time falls into the category of high performance computing.

Companies needing high-quality visualizations seldom have supercomputers on their own. Such machines are either too expensive or regarded as irrelevant by many IT managers. A rental and a temporary physical installation of additional computers in order to meet production deadlines are accompanied by technical problems and additional costs.

A prototype of an advanced e-commerce rendering system addressing the above problems has been developed during the project *HiQoS* (**Hi**gh Performance Multimedia Services with **Q**uality **o**f **S**ervice Guarantees) [1], [2]. This system, *HiQoS Rendering System*, allows a user to submit rendering jobs via a simple web interface. The further processing of jobs is fully automatic. The complexity and the distributed nature of the system are hidden from the user. The project HiQoS was financially supported by the German Ministry of Education and Research (BMBF). The HiQoS Rendering project partners were: Axcent Media AG, GPO mbH, IEZ AG, University of Paderborn and Upstart! GmbH.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 304–315, 2001.

We know about several projects related to a remote global lighting simulation. *Virtual Frog* [3] is one of the pioneering works which aims to support teaching of biological principles: "... our goal is to provide accessibility over the Web in order to reduce the complexity of installing, running, and managing the software." The interactivity played a major role in this project – the model (of a frog) resides on a server and the server computes a desired visualization of the model (a schematic view, a view of a scanned slice, a volume traced view, etc.) *Online Rendering of Mars* [4] is technically very similar: the user chooses a perspective and a lighting and within a few minutes gets back a rendered picture of Mars. Closer to the HiQoS project idea is a rendering server using the (sequential) *Radiance* program to compute ray traced pictures of a user's model [5]. There also are companies offering their computers for rendering purposes [6] – however, the data exchange, job specification and scheduling of the rendering jobs are handled by human operators. The HiQoS rendering project went further, offering an *automatic and parallel* rendering service on demand, whereby parallel computers in several computing centers can be combined into a single computing system.

The global illumination problem is defined in section 2. Parallelizations of two global illumination methods, ray tracing and radiosity, is discussed in the same section. Special attention is devoted to an efficient representation of the diffuse global illumination resulting from the radiosity method. Two approaches to the simplification of radiosity solutions are presented: mesh decimation and radiosity maps. The architecture of the HiQoS Rendering System is described in section 3. Section 4 describes an evaluation of the HiQoS Rendering System in two industrial scenarios, in architectural visualization and in film production. In section 5 we draw our conclusions and sketch our future research directions.

2 Simulation of Global Illumination

Photorealistic visualization of 3D models is one of the quests of computer graphics. Almost all light phenomena are well explained by the quantum electrodynamics. However, for practical purposes it is not desirable to simulate the propagation of light or to model large-scale 3D models on a subatomic level. Synthesis of photorealistic pictures is a chain of simplifications. Kajiya's rendering equation [7] provides a framework for the simulation of global lighting on the level of geometrical optics. The rendering equation is an integral equation describing an energy balance in all surface points of a 3D model. With exception of extremely simple cases this equation cannot be solved analytically. The ray tracing and radiosity methods [8] make additional assumptions about the light-object interactions in order to simplify the rendering equation.

The assumption of ray tracing is that all indirect light reflections are perfectly specular. Ray tracing does not solve the rendering equation explicitly – it traces photons from the camera into the 3D scene (in a backward direction), measuring the contributions of light sources to the camera pixels along the traced paths. Ray tracing is inherently view dependent. Its product is the picture seen by a given camera. The illumination is not stored in the 3D model.

Radiosity assumes that all light reflections are perfectly diffuse and that the 3D model consists of a finite number of small elements (*patches*). Under these assumptions the rendering equation can be formulated as a system of linear equations. The system is usually very large and the computation of all its coefficients is prohibitively expensive. Radiosity algorithms solve the system iteratively, storing the illumination in the 3D model. The radiosity method is view independent. A converged radiosity solution is an illuminated 3D model which can be viewed from different perspectives without having to rerun the lighting simulation.

Data-parallel radiosity and ray tracing algorithms have been developed and implemented within the HiQoS project. The following sections briefly describe the parallelization ideas. (An asynchronous distributed memory model with message passing is assumed.)

2.1 Parallel Ray Tracing

Our ray tracing parallelization is based on a screen subdivision strategy which exploits the independence of computations on any two pixels. Processor farming is a natural way of the parallelization. However, there are two potential problems with a straightforward implementation: bad load balancing (computation times for two pixels are different) and no data-scalability (replication of the 3D model in each processor imposes a limit to the maximum model size).

The problem of unequal processor load can be solved under an assumption that the ratio of the computation times on any two equally large areas of the screen can be bounded by a constant [9]. The idea is to assign large screen areas to processors first, then smaller, ending up with single pixels (or other sufficiently small pieces). Almost linear speedups can be measured up to 64 processors.

A distributed object database is used to avoid the necessity of storing the whole 3D model in every processor. Large models (requiring several Gigabytes of memory) can so be rendered on currently available parallel machines. The only limitation to the model size is the *total* memory of the processors used for the computation. Each processor holds a resident subset of all objects of the 3D model. The remaining objects (or a part of them) are stored in a cache memory. If a non-resident object is needed during the computation, the processor stops the computation, sends a request to the processor holding the object and makes space for the requested object in its cache by removing some other objects from the cache if needed. Upon having received the requested object, the processor resumes the computation. An LRU strategy is used for the cache management (always the *L*ast *R*ecently *U*sed object is removed from the cache). A similar implementation is described in [10].

2.2 Parallel Radiosity

A progressive refinement method [11] is used for the simulation of light propagation. Each patch of the model is a potential light source. The unshot energy

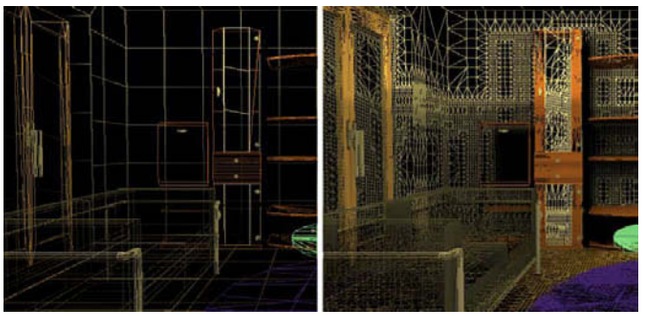
and the total reflected energy are stored by each patch. The original (sequential) progressive refinement method iteratively selects a patch with the most unshot energy (a *shooter*) and shoots the whole unshot energy in the half-sphere surrounding the shooting patch. The total reflected and unshot energies of all other patches (*receivers*) are updated during the shooting according to their visibility from the shooter (the receiving patches can be refined in this step to store the gathered energy accurately enough). This process is iterated until the total unshot energy drops down under a threshold (or some additional termination criterium is fulfilled). The computation of visibility between two patches (a so-called *form factor*) is a non-trivial problem. We are currently using a ray casting method for the form factor computation [12]. This method randomly generates samples on the shooter and the receiver and checks how many pairs of the shooter-receiver samples are mutually visible. The number of visible sample pairs is used in estimation of the total mutual visibility between the two patches.

The parallel radiosity algorithm begins with a preprocessing step (a meshing step) in which the model is discretized into patches (in our implementation the mesh consists only of triangle and quadrangle patches). The discretized model is partitioned into 3D subscenes which are distributed onto processors. The shooting iterations run asynchronously in all processors. Aside of the locally stored patches, each processor maintains an incoming message queue which contains information about shooters selected by other processors. At the beginning of each iteration a processor looks for the shooter with the most unshot energy among the patches in its message queue and the locally stored patches. Then it performs either an *external shooting* (if a shooter from the message queue has been selected) or an *internal shooting* (if a local shooter has been selected). A shooting influences only the locally stored patches. To spread the information about shooting iterations performed locally, the processor broadcasts the selected shooter in the case of an internal shooting [13], [14].

Additional improvements to the parallelization described above include dynamic load balancing (a processor's load can be measured by the number of yet unshot external shooters waiting in the message queue) and two representations of the 3D model used to speed up form factor computations [15].

2.3 Simplification of Radiosity Solutions

The diffuse illumination computed by the parallel radiosity program is stored in the vertices of the polygon mesh (vertex radiosities). Interpolation is used to compute the outgoing radiosities in other surface points. The original mesh gets progressively refined during the radiosity computation (new vertices are created) in order to store the illumination accurately enough. This refinement leads to huge data volumes resulting from radiosity simulations (Fig. 1). Memory requirements of radiosity solutions cause problems by transferring the data to the user over the Internet, by a subsequent postprocessing of the scene, by interactive walkthroughs, by rendering final pictures, etc. The following sections describe two methods of simplification of radiosity solutions. Both methods were imple-



(a) (b)

Fig. 1. Refinement of a polygonal mesh: (a) original mesh; (b) refined mesh

mented (both sequential and data-parallel versions) and the later was integrated in the HiQoS Rendering System.

Mesh Decimation. The idea of the mesh decimation method is an iterative deletion of edges from the model using the *vertex-unify* operation (joining of two adjacent vertices into one vertex). The problem of overwhelming memory complexity also arises by acquisition of 3D models using 3D scanners and the first existing mesh decimation methods have been developed in this application area [16], [17]. The presence of additional radiosity information stored in the vertices of an illuminated model influences only the choice of a metric used in the algorithm.

Mesh decimation algorithm
INPUT: a polygon mesh M, a desired compression ratio c
WHILE (compression ratio c is not reached)
(1) In M, select edge $e = [P_1, P_2]$ for removal
(2) Remove edge e by joining points P_1, $P2$ into point P
(3) Find optimal placement for point P
OUTPUT: a simplified polygon mesh M

It is not obvious in which order the edges should be scheduled for the removal (line 1) and how to find the optimal placement for the vertices resulting from the *vertex-unify* operation (line 3). An objective *metric* is used to answer both questions in a single optimization step. The metric evaluates the current quality of the simplified mesh. The selection of the edge to be removed and the placement of the resulting vertex are a solution to an optimization problem which minimizes the quality reduction over all possible edge selections and all possible placements of the resulting vertex. The choice of the metric determines the complexity of the optimization problem (costs of evaluation of the metric alone must also be considered) and influences the final visual quality of the simplified mesh. We

used a quadric metric [18], [19] which takes the vertex radiosities into account [20].

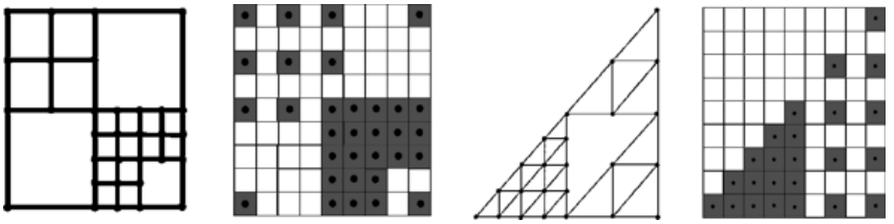
One problem of the mesh decimation method is that the visual quality of the simplified mesh is not guaranteed. Mesh decimation can be seen as a lossy compression of a model. Whereas compression ratios of up to 90% can be achieved retaining an acceptable visual quality for some scenes, the visual error by the same compression ratios is high for other scenes. The compression ratio can be chosen interactively (using the operator's visual perception as a quality measure) but interaction makes the method unsuitable for use in an automated remote rendering system.

Radiosity Maps. The method of radiosity maps provides a non-lossy compression of illuminated meshes using a more efficient representation of the illumination. The vertex radiosities are stored in radiosity maps (texture maps) instead of in mesh vertices. One radiosity map is assigned to each object (an object is a set of patches forming a logical element, e.g. chair, table, staircase, etc.) Additionally, *uv*-mapping coordinates are stored in the vertices of the resulting mesh. The entire substructuring information is removed from the mesh. Typical compression ratios are 70%–80%. Moreover, texture mapping (see Fig. 3 (a)) is supported by the hardware of recent graphics cards which significantly increases framerates by interactive walkthroughs.

Computation of radiosity maps
INPUT: a mesh M with vertex radiosities
FOR (each object obj_k in mesh M)
  FOR (each patch p_i in object obj_k)
    (1) Find optimal resolution of radiosity map for patch p_i
    (2) Create radiosity map $pmap_i$ for patch p_i
    (3) Fill radiosity map $pmap_i$ for patch p_i (Fig. 2)
  (4) Pack patch maps $pmap_i$ into object map $omap_k$ (Fig. 3 (b))
  (5) Remove substructuring from all patches p_i in object obj_k
  (6) Assign *uv*-mapping coordinates to all patches p_i in object obj_k
OUTPUT: a simplified mesh M with radiosity maps assigned to objects
  (substructuring information removed)

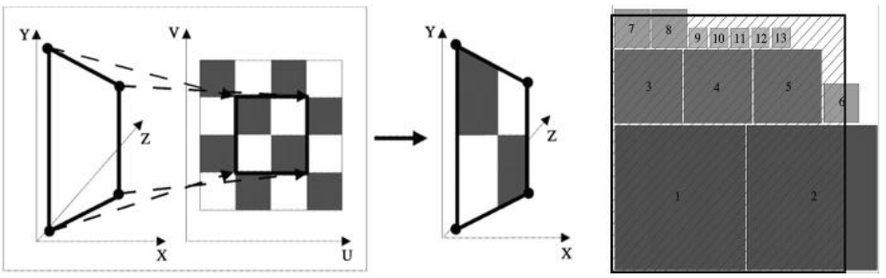
Optimal resolution of a patch map (line 1) is the minimal resolution allowing a non-lossy representation of the original illumination information. This resolution depends of the depth of the substructuring of a given patch (examples are shown in Fig. 2).

An optimal packing of patch maps into an object map (line 4) is an NP-hard problem. A heuristic is used in this step. The heuristic tries to keep the object map as small as possible and to use the rectangular area of the object map efficiently (Fig. 3 (b)).



(a) (b)

Fig. 2. Optimal resolutions of radiosity maps: (a) a quadrangle patch and the corresponding map; (b) a triangle patch and the corresponding map.
Original vertex radiosities are stored in the marked texels. The colours of empty texels are interpolated from the marked ones



(a) (b)

Fig. 3. (a) Mapping of a radiosity map onto a 3D object (*uv*-texture mapping). The *uv*-mapping coordinates are stored in the object's vertices; (b) Packing of patch radiosity maps into an object radiosity map. The dashed area corresponds to the total area of the patch maps (a lower bound for the optimal object map size)

3 Architecture of the HiQoS Rendering System

The HiQoS Rendering System is a distributed system consisting of four subsystems: *Client*, *Service Broker*, *Scheduling Subsystem* and *Rendering Server*. All of these subsystems may be replicated. An exception is the *Service Broker*, the system's central entry point. The subsystems and their components communicate via TCP/IP sockets. No shared file system is needed. A possible configuration of the system is shown in Fig. 4. The main objectives of this architecture are:

- minimal hardware and software requirements on the user's side (only an Internet connection and a web browser are needed)
- transparent access to high performance computing systems (the user does not know where the computation takes place or what computing systems are used for computing his job)
- good resource utilization (especially by rendering walkthroughs consisting of many frames).

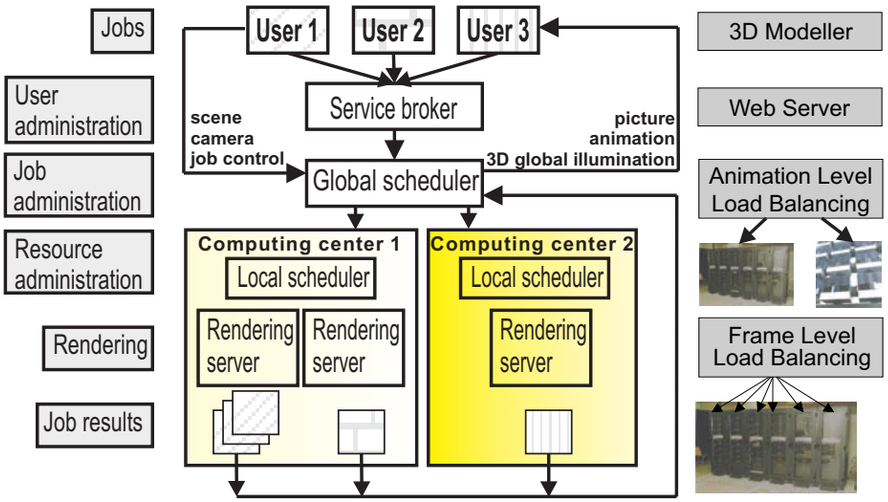


Fig. 4. An example configuration of the HiQoS Rendering System

3.1 Client

The *Client* subsystem is responsible for a user's authorization by the *Service Broker*, submission of a rendering job and providing input data needed by the job. The input data generated automatically by the user's modelling software describe a 3D scene: its surface geometry, surface materials, light sources and cameras. Extended *VRML 2* and *3DS* formats are currently supported. It is practical to separate the camera information from the rest of the scene description (a proprietary camera format is used which allows a definition of single cameras as well as camera paths). The rest of the data are controlling parameters specific to the selected illumination method. These are provided by a human operator who fills out an HTML form when submitting a rendering job.

The 3D scene data can be large. They bypass the service broker and flow from the *Client* directly to the *Global Scheduler*. Two transmission scenarios were considered:

- *A passive client scenario*, in which the user places the exported scene data in a public area in the Internet (e.g. the user's web area) and tells the rendering system where they are (URLs). This is the currently implemented scenario. Main disadvantages of this scenario are an additional load in the system related to the downloading the user's data and a violation of the privacy of the data. An advantage is a simple implementation.
- *An active client scenario*, in which the user (e.g. a software component integrated in the user's 3D modeller) uploads the data to the rendering system. Advantages are a possibility of integration of accessing the remote rendering system directly from the user's modeller, a possibility of maintaining an object cache on the rendering system's side for reducing the transmission

costs, ensuring the privacy of the transmitted scenes, etc. A disadvantage is that additional software components and communication protocols between them must be implemented.

3.2 Service Broker

The *Service Broker* is the system's entry point. This software component communicates with users (and administrators of the system), accepts new jobs, generates a unique id for each accepted job and delivers computed results to users. The *Service Broker* is implemented as an Apache web server (PHP scripts are used for implementing the communication with the *Global Scheduler* and for accessing databases kept on the server).

3.3 Scheduling Subsystem

The *Scheduling subsystem* consists of a *Global Scheduler* and several *Local Schedulers*). The former downloads the job data, converts the data, distributes jobs (or their parts) to the network of *Local Schedulers*, collects the results (or partial results) from *Local Schedulers* and passes the results and a status information to the *Service Broker*. There is usually only one *Global Scheduler* running in the entire system.

Different parallel computing systems in different computing centers can be used in the system. There is usually one instance of the *Local Scheduler* running in one computing center. The main task of the *Local Scheduler* is to hide the differences between the parallel systems, providing a unique interface for allocation and deallocation of processors, for starting a parallel application on allocated processors, etc.

Such two-level scheduling scheme is modular and also helps to efficiently utilize the computing resources by rendering of ray traced walkthrough animations. In this case the *Global Scheduler* acts as a farmer distributing single frames to several *Local Schedulers*. After having computed a frame, the *Local Scheduler* sends the frame (a picture) to the *Global Scheduler* and gets in turn a new frame to compute. Between the computations of two subsequent frames the 3D scene persists in memories of running parallel processors on the *Local Scheduler*'s side. Thus only a short description of the new camera must be retransmitted between the schedulers with each single frame.

3.4 Rendering Server

Data-parallel ray tracing and data-parallel radiosity algorithms are currently supported by the HiQoS Rendering System. These programs are precompiled for the target parallel systems in computing centers. An actual implementation of parallel global illumination algorithms is hidden in the *Rendering Server*. The *Rendering server* provides an interface to a *Local Scheduler* allowing starting, continuation and termination of a chosen parallel program on an allocated

partition of a parallel system. The interface is independent of the illumination method, even though the methods considerably differ from each other (also in input and output parameters). A radiosity job is handled by a *Local Scheduler* in the same way as a ray tracing job with one camera.

4 Evaluation of the HiQoS Rendering System

The remote rendering system has been evaluated in two industrial scenarios: architectural visualization and film production. Although the final goal of both scenarios is a synthesis of photorealistic pictures, the way of achieving this goal differs. The user of the first scenario was the company *GPO mbH* which creates complex CAD models using the software *Speedikon* (by *IEZ AG*) and *Arcon: Architektur Visualisierung* (by *mb-Software AG*). The requirement to the rendering system was a synthesis of high quality visualization pictures and camera animations of the models. The results have been used for supporting the building contractors during the planning phase and for a public presentation of the models. The parallel ray tracing offered by the HiQoS Rendering System allows to significantly reduce the rendering times in comparison to a sequential computation on a PC. The measured total overhead of the system (the effective rendering time against the time spent in downloading the models and in scheduling) is below 10% already by small rendering jobs consisting of about 10 frames (and smaller by more complex jobs).

The user of the film production scenario has been the company *Upstart! GmbH* which creates special visual effects for movies (e.g. "Operation Noah") and advertisements. *3D Studio Max* (by *Discreet*) is used for the 3D modeling. A frequent problem is a realistic illumination of building interiors which do not really exist. A sequence of pictures is not desirable as a product of the global illumination simulation because this would drastically reduce the flexibility by the final composition. Rather than pictures, an explicit 3D representation of the illuminated model is required as an intermediate result of the simulation. The (sequential) radiosity of *Lightscape* (by *Discreet*) is used in this step of the original production chain. The sequential radiosity computation of a complex model can take many hours, sometimes days. Another, even more serious problem are the run-time memory requirements of the radiosity method which sometimes exceed the possibilities of a PC. The data-parallel radiosity integrated in the HiQoS Rendering System leads to shorter computation times and overcomes the memory problems by using more processors with more total memory. The radiosity maps described in section 2.3 are used for the compression of the resulting data.

5 Conclusions

We described a parallel rendering system which allows computation of the global illumination of complex 3D models sent by users via the Internet. Currently data-parallel ray tracing and radiosity illumination methods are supported by

(a) (b)

Fig. 5. (a) Model of the Dom in Wetzlar. Global diffuse illumination was computed by radiosity, textures and camera effects were added in a subsequent rendering step; (b) Ray traced model of a furnished house

the system. The system was integrated as a prototype of an e-commerce service and successfully evaluated in two industrial scenarios. Possible extensions and research areas include implementations of further global illumination methods, a seamless integration of the remote rendering service with existing 3D modellers, a support for dynamical scenes and a better resource management.

References

1. P. Altenbernd, F. Cortes, M. Holch, J. Jensch, O. Michel, C. Moar, T. Prill, R. Lüling, K. Morisse, I. Neumann, T. Plachetka, M. Reith, O. Schmidt, A. Schmitt, A. Wabro: BMBF-Projekt HiQoS. Statustagung des BMBF, HPSC '99, Höchstleistungsrechnen in der Bundesrepublik Deutschland, Bonn, R. Krahl (ed.), (1999) 29–32
2. Projekt HiQoS: High Performance Multimediadienste mit Quality of Service Garantien. http://www.c-lab.de/hiqos
3. D. W. Robertson, W. E. Johnston, W. Nip: Virtual Frog Dissection: Interactive 3D Graphics via the Web. Proc. of The Second International WWW Conf. Mosaic and the Web, Chicago, Illinois (1994)
4. Mars Online Rendering. http://astronomy.swin.edu.au/mars
5. Artifice, Inc., RenderCity! – Free Online Ray Tracing with Radiance. http://www.artifice.com/rendering/render.html
6. MTP Grafx, Render Farm. http://www.mtpgrafx.com/rfarm.htm
7. J. Kajiya: The Rendering Equation. Computer Graphics, 20, **4** (1986) 143–150
8. A. Watt, M. Watt: Advanced Animation and Rendering Techniques. Addison-Wesley (1992)
9. T. Plachetka: POV‖Ray: Persistence of Vision Parallel Raytracer. Proc. of Spring Conf. on Computer Graphics, Budmerice, Slovakia (1998) 123–129
10. S. Green: Parallel Processing for Computer Graphics. Pitman MIT Press (1991)

11. M. F. Cohen, S. E. Chen, J. R. Wallace, D. P. Greenberg: A Progressive Refinement Approach to Fast Radiosity Image Generation. Computer Graphics, 22 (1988) 75–84
12. J. R. Wallace, K. A. Elmquist, E. A. Haines: A Ray Tracing Algorithm for Progressive Radiosity. Computer Graphics, 23, **3** (1989) 315–324
13. O. Schmidt, J. Rathert, L. Reeker, T. Plachetka: Parallel Simulation of Global Illumination. Proc. of the Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA '98), Las Vegas, Nevada, CSREA Press, H. R. Arabnia (ed.), **3** (1998) 1289-1296
14. L. Reeker, O. Schmidt: New Dynamic Load Balancing Strategy for Efficient Data-Parallel Radiosity Calculations. Proc. of the Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA '99), Las Vegas, Nevada, CSREA Press, H. R. Arabnia (ed.), **1** (1999) 532–538
15. O. Schmidt: Parallele Simulation der globalen Beleuchtung in komplexen Architekturmodellen. PhD Dissertation, Fachbereich Mathematik/Informatik, Universität Paderborn (2000)
16. J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, W. Wright: Simplification Envelopes. Computer Graphics (Proc. of SIGGRAPH '96) (1996) 119–128
17. H. Hoppe: Progressive Meshes. Computer Graphics (Proc. of SIGGRAPH '96) (1996) 99–108
18. M. Garland, P. Heckbert: Simplifying Surfaces with Color and Texture using Quadric Error Metrics. Proc. of IEEE Visualization '98 (1998) 263–269
19. H. Hoppe: New quadric metric for simplifying meshes with appearance attributes. Proc. of IEEE Visualization '99 (1999) 59–66
20. M. Rasch, O. Schmidt: Parallel Mesh Simplification. Proc. of the Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA '00), Las Vegas, Nevada, CSREA Press, H. R. Arabnia (ed.), **3** (2000) 1361–1367

Two-Way Restarting Automata and J-Monotonicity*

Martin Plátek

Charles University, Department of Computer Science,
Malostranské nám. 25, 118 00 PRAHA 1, Czech Republic,
platek@ksi.ms.mff.cuni.cz

Abstract. We introduce restarting automata as two-way automata in order to obtain a more general model which is closer to our linguistic motivations. We study the notion of j-monotonicity to show the advantages of this model. We show that the j-monotonicity can be considered as a degree of non-context-freeness and that it is a robust notion due to the considered models of restarting automata. Some other aspects concerning power and applications of two-way restarting automata are mentioned.

1 Introduction

Roughly speaking, a restarting automaton accepts a word by a sequence of changes (cycles) where each change results in a shorter word for which the automaton restarts.

One important (linguistic) motivation for studying restarting automata is to model *analysis by reduction* of natural language sentences in a similar way as in [9]. The analysis by reduction consists of a gradual simplification of a full sentence so that the (in)correctness of the sentence is not affected. Thus, after a certain number of steps, a simple sentence is achieved or an error is found. Two-way restarting automata are more adequate for this purpose than one-way restarting automata because of an easier (deterministic) implementation of desired 'global outlook' on the whole reduced sentence. We feel that two-way restarting automata are closer to the human procedure performing analysis by reduction. Two-way restarting automata can be considered not only as a generalization of one-way restarting automata but also as a generalization of contraction automata (see [10]).

This paragraph serves for further explanation of our linguistic motivations. In order to give a possibility to characterize a type of complexity of different (natural) languages and their analysis by reduction we introduce and study in a formal way the notion of degree of (non)monotonicity. In this way we obtain a new formal characterization of the still vague notions of word-order complexity of

* The research reported in this paper has been supported by the Grant Agency of the Czech Republic, Grant-No. 201/99/0236, and by the Grant Agency of Charles University, Grant-No. 157/1999/A INF/MFF.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 316–325, 2001.

(natural) languages. In [2] the word-order complexity in languages was measured using a measure of nonprojectivity of dependency (or syntactic) trees. We can use the degree of (non)monotonicity in a similar way as was used the measure of nonprojectivity in [3]. We have shown there that the degree of word-order complexity of English is significantly lower than the degree of word-order complexity of Czech or Latin. I.e., using our results, it should be also possible to show that the 'syntax' of Czech is 'more distant' from context-free syntax than the 'syntax' of English. Let us note that the degree of (non)monotonicity uses much more elementary (easier observable) syntactic features of natural languages than the measure of nonprojectivity of D-trees. The formal results presented in this paper should demonstrate the idea, that the j-monotonic two-way restarting automata create a formal tool useful for analyzers, which perform the analysis by reduction for natural languages, particularly for natural languages with a high degree of word-order freeness. Let us recall that the analysis by reduction (formally or informally, implicitly or explicitly) creates a starting point for development of syntactic parsers (or for formal descriptions of syntax and its (un)ambiguity) of natural languages.

Now we focus our attention on the technical content of the paper. The (non)monotonicity of degree 1 of a computation $\mathcal{C}$ means, roughly speaking, that the sequence of distances between the individual changes in the recognized word (sentence) and the right end of the word is decreasing by $\mathcal{C}$. The (non)monotonicity of degree j (j-monotonicity) of a computation means that the mentioned sequence of distances can be obtained by a 'shuffle' of j (or less) decreasing sequences.

We have started to study the j-monotonicity a short time ago, see [7]. This paper should stress the power of (deterministic) two-way automata, and the robustness of the basic observation that the j-monotonicity determines several infinite scales of languages and their analyzes by reductions.

2 Definitions

Throughout the paper, λ denotes the empty word. We start with a definition of restarting automata, in special forms in which we shall be interested.

A *two-way restarting automaton* M, formally presented as a tuple $M = (Q, \Sigma, \Gamma, k, I, q_0, Q_A, Q_R)$, has a control unit with a finite set Q of *(control) states*, a distinguished *initial state* $q_0 \in Q$, and two disjoint sets $Q_A, Q_R \subseteq Q$ of *halting states*, called *accepting* and *rejecting* respectively. M further has one head moving on a finite linear list of items (cells). The first item always contains a special symbol ¢, i.e. the *left sentinel*, the last item always contains another special symbol \$, i.e. the *right sentinel*, and each other item contains a symbol either from an *input alphabet* Σ or a *working alphabet* Γ; we stipulate that Σ and Γ are finite disjoint sets and $¢, \$ \notin \Sigma \cup \Gamma$. We assume the head having a *look-ahead* window of the size k, so that it always scans k consecutive items ($k \geq 1$). There is one exception, in the case when the right sentinel \$ appears in the window M can also scan less than k symbols (\$ is the last scanned symbol).

A *configuration* of M is a string $\alpha q \beta$ where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{¢\} \cdot (\Sigma \cup \Gamma)^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot (\Sigma \cup \Gamma)^*$ and $\beta \in (\Sigma \cup \Gamma)^* \cdot \{\$\}$; here q represents the current (control) state, $\alpha\beta$ the current contents of the list of items while it is understood that the head scans the first k symbols of β (or the whole β when $|\beta| < k$). A *restarting configuration*, for a word $w \in (\Sigma \cup \Gamma)^*$, is of the form $q_0 ¢ w \$$; if $w \in \Sigma^*$, $q_0 ¢ w \$$ is an *initial configuration*.

As usual, a *computation* of M (for an input word $w \in \Sigma^*$) is a sequence of configurations starting with an initial configuration where two consecutive configurations are in the relation $\vdash_M$ (denoted $\vdash$ when M is clear from the context) induced by a finite set I of instructions. Each *instruction* is of one of the following four types:

(1) $(q, \gamma) \to (q', MVR)$ (2) $(q, \gamma) \to (q', MVL)$
(3) $(q, \gamma) \to (q', REWRITE(\gamma'))$ (4) $(q, \gamma) \to RESTART$

Here q is a nonhalting state ($q \in Q - (Q_A \cup Q_R)$), $q' \in Q$, and γ, γ' are "look-ahead window contents"(i.e., $k \geq |\gamma|$), where $|\gamma| > |\gamma'|$ (i.e., rewriting must shorten the word).

Type (1) (moving right), where $\gamma = a\beta$, a being a symbol, induces $\alpha q a \beta \delta \vdash \alpha a q' \beta \delta$. Type (2) (moving left), where $\alpha = \alpha_1 b$, $\gamma = a\beta$, a, b being symbols, induces $\alpha q a \beta \delta \vdash \alpha_1 q' b a \beta \delta$. Type (3) (rewriting) induces $\alpha q \gamma \delta \vdash \alpha \gamma' q' \delta$; but if $\gamma = \beta\$$, in which case $\gamma' = \beta'\$$, we have $\alpha q \beta \$ \vdash \alpha \beta' q' \$$. Type (4) (restarting) induces $\alpha q \gamma \delta \vdash q_0 \alpha \gamma \delta$.

We assume that there is an instruction with the left-hand side (q, γ) for each nonhalting state and each (possible) γ (i.e., in any configuration a further computation step is possible iff the current state is nonhalting). In general, the automaton is *nondeterministic*, i.e., there can be two or more instructions with the same left-hand side (q, γ), and thus there can be more than one computation for an input word. If this is not the case, the automaton is *deterministic*.

An input *word* $w \in \Sigma^*$ *is accepted by* M if there is a computation which starts with the initial configuration $q_0 ¢ w \$$ and finishes with an *accepting configuration* – where the current control state is accepting. $L(M)$ denotes the language consisting of all words accepted by M; we say that M *recognizes (accepts) the language* $L(M)$.

We observe that any finite computation of a two-way restarting automaton M consists of certain phases. A phase called a *cycle* starts in a (re)starting configuration, the head moves along the input list until a restart operation is performed and thus a new restarting configuration is reached. If no further restart operation is performed, any finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*. For technical convenience, we assume that M performs at least one rewrite operation during any cycle (this is controlled by the finite state control unit) – thus the new phase starts on a shortened word.

We use the notation $u \Rightarrow_M v$ meaning that there is a cycle of M beginning with the restarting configuration $q_0 ¢ u \$$ and finishing with the restarting configuration $q_0 ¢ v \$$; the relation $\Rightarrow_M^*$ is the reflexive and transitive closure of $\Rightarrow_M$. We sometimes (implicitly) use the following obvious fact:

Fact 1. Error preserving property:
Let M be a restarting automaton, and u, v two words in its *input* alphabet. If $u \Rightarrow_M^* v$ and $u \notin L(M)$, then $v \notin L(M)$.

Correctness preserving property:
Let M be a deterministic restarting automaton, and u, v two words in its *input* alphabet. If $u \Rightarrow_M^* v$ and $u \in L(M)$, then $v \in L(M)$.

Now we define the subclasses of restarting automata relevant for our consideration.

- An *RLWW-automaton* is a two-way restarting automaton which performs exactly one *REWRITE*-instruction in each cycle (this is controlled by the finite state control unit).
- An *RLW-automaton* is an *RLWW*-automaton whose working alphabet is empty. Note that each restarting configuration is initial in this case.
- An *RL-automaton* is an *RLW*-automaton whose rewriting instructions can be viewed as deleting, i.e., in the instructions of type (3), γ' is obtained by deleting some (possibly not continuous subsequence of) symbols from γ.
- An *RRWW-automaton* is an *RLWW*-automaton which does not perform the instruction of the type (2) (move to the left) at all.
- An *RRW-automaton* is an *RRWW*-automaton whose working alphabet is empty.
- An *RR-automaton* is an *RRW*-automaton whose rewriting instructions can be viewed as deleting.

We recall the notion of monotonicity for a computation of an *RLWW*-automaton and we introduce the notion of j-monotonicity. Any cycle C contains a unique configuration $\alpha q \beta$ in which a rewriting instruction is applied. We call $|\beta|$ the *right distance*, *r-distance*, of C, and denote it $D_r(C)$. We say that a *sequence of cycles* $Sq = C_1, C_2, \cdots, C_n$ is *monotonic* iff $D_r(C_1) \geq D_r(C_2) \geq \cdots \geq D_r(C_n)$; A *computation* is *monotonic* iff the respective sequence of cycles is monotonic. (The tail of the computation does not play any role here.)

Let j be a natural number. We say that the sequence of cycles $Sq = (C_1, C_2, \cdots, C_n)$ is *j-monotonic* iff (informally speaking) there is a partition of Sq into j subsequences, where each of them is monotonic. I.e., the sequence of cycles Sq is *j-monotonic* iff there is a partition of Sq into j subsequences
$Sb_1 = (C_{i^1{}_1}, C_{i^1{}_2}, \cdots, C_{i^1{}_{p1}})$,
$Sb_2 = (C_{i^2{}_1}, C_{i^2{}_2}, \cdots, C_{i^1{}_{p2}})$,
...
$Sb_j = (C_{i^j{}_1}, C_{i^j{}_2}, \cdots, C_{i^j{}_{pj}})$,
where each Sb_i, $1 \leq i \leq j$ is monotonic. Obviously a sequence of cycles $(C_1, C_2, \cdots, C_n)$ is not j-monotonic iff there exist $1 \leq i_1 < i_2 < ... < i_{j+1} \leq n$ such that $D_r(C_{i_1}) < D_r(C_{i_2}) < \cdots < D_r(C_{i_{j+1}})$.

A *computation* is *j-monotonic* iff the respective sequence of cycles is j-monotonic. (The tail of the computation does not play any role here.) We can see that 1-monotonicity means monotonicity.

Let j be a natural number. An *RRWW-automaton* M is *j-monotonic* iff all its computations are j-monotonic.

Notation. For brevity, *Nat* denotes the set of natural numbers, prefix *det-* denotes the property of being deterministic, prefix *j-mon* denotes j-monotonicity. $\mathcal{L}(A)$, where A is some class of automata, denotes the class of languages recognizable by the automata from A. E.g., $\mathcal{L}$(*det-j-mon-RRW*) denotes the class of languages recognizable by deterministic j-monotonic RRW-automata. CFL denotes the class of context-free languages, $DCFL$ the class of deterministic context-free languages. The sign $\subset$ means the proper subset relation. We will sometimes write regular expressions instead of the respective regular languages. Similarly we will write only *mon* instead of *1-mon*.

3 Results

Proposition 2. *Let ML be an RLWW-automaton. Then there is a (nondeterministic) RRWW-automaton MR such that* $L(ML) = L(MR)$, $u \Rightarrow_{ML} v$ *iff* $u \Rightarrow_{MR} v$, *and the right distances in the corresponding cycles by ML and MR are equal.*

Proof. We outline the main idea only. We need to construct *MR* in such a way that it is able to simulate any cycle of *ML*. For this aim we can use the technique of simulation of two-way finite automata by one-way finite automata (the technique of crossing functions). We need to ensure the simulation of the first part of the cycle (the part before the single rewriting) of *ML* and at the same time we need to ensure the simulation of the second part of the cycle (the part after the rewriting). It means, that *MR* needs to use two sets of crossing functions (instead of one) in order to check the correctness of the simulated cycle. □

Theorem 3. *It holds for any* $j \in Nat$:

$\mathcal{L}$*(RRWW)* $= \mathcal{L}$*(RLWW),* $\mathcal{L}$*(j-mon-RRWW)* $= \mathcal{L}$*(j-mon-RLWW),*
$\mathcal{L}$*(j-mon-RRW)* $= \mathcal{L}$*(j-mon-RLW),* $\mathcal{L}$*(j-mon-RR)* $= \mathcal{L}$*(j-mon-RL).*

Proof. The theorem follows from the previous proposition. □

Theorem 4. $\mathcal{L}$*(mon-RRWW)* $= \mathcal{L}$*(mon-RLWW)* $= CFL$

Proof. The fact $\mathcal{L}$*(mon-RRWW)* $= CFL$ was shown e.g. in [1]. The theorem follows from the equality $\mathcal{L}$*(mon-RRWW)* $= \mathcal{L}$*(mon-RLWW)* shown in Theorem 3. □

The following fact is in [1] formulated for *RRWW*-automata. It holds for *RLWW*-automata because of the Proposition 2 and Theorem 3.

Fact 5. For any RLWW-automaton M there is a constant p such that the following holds. Let $uvw \Rightarrow_M uv'w$, $|u| = p$, and u_2 is a subword of the word u, i.e., we can write $u = u_1u_2u_3$. Then we can find a nonempty subword z_2, such that we can write $u_2 = z_1z_2z_3$, and $u_1z_1(z_2)^iz_3u_3vw \Rightarrow_M u_1z_1(z_2)^iz_3u_3v'w$ for all $i \geq 0$ (i.e., z_2 is a "pumping subword" in the respective cycle). Similarly, such a pumping subword can be found in any subword of length p of the word w.

Theorem 6. *$\mathcal{L}$(det-j-mon-RRWW) = DCFL holds for any $j \in Nat$.*

Proof. We only outline the idea of the proof here. Let us suppose that M is a *det-j-mon-RRWW*-automaton and k is the size of its look-ahead window. We can see that M computes in such a way that in any cycle it scans at least one item which was scanned (rewritten) by M in the previous cycle by its (single) rewriting instruction. That means that during a computation of M the value of their right distances can increase less than $k \cdot j$ only. It means that any computation (cycle) of M can be simulated by a computation of a *det-mon-RRWW*-automaton M_{jk} with a look-ahead window of the size $k \cdot j$. Since $\mathcal{L}$*(det-mon-RRWW) = DCFL* (see [6]) the theorem is proved. □

Theorem 7. *$DCFL = \mathcal{L}$(det-mon-RR) $\subset \mathcal{L}$(det-mon-RL)*

Proof. The fact $DCFL = \mathcal{L}$*(det-mon-RR)* was proved in [6]. We will prove that the language $L_a = L_c \cup L_d$, where $L_c = \{a^n b^n c \mid n \geq 0\}$, $L_d = \{a^n b^{2n} d \mid n \geq 0\}$ is a *det-mon-RL*-language.

L_a is recognized by a *det-mon-RL*-automaton M which works as follows:

- M accepts if it scans $\¢c\$$, $\¢d\$$ or $\¢abc\$$, otherwise
- M moves to the right end in order to check whether the input word has the form a^+b^+c . If the check is positive then M moves to the "middle" (a followed by a different symbol) and deletes ab. If the previous check is not successful M moves to the right end in order to check whether the input word has the form a^+b^+bd . If the check is positive then M moves to the "middle" and deletes abb. If the last check is not successful, M rejects, otherwise it restarts.

On the other hand it was shown e.g. in [6] that $L_a \notin DCFL$.This completes the proof. □

Next we will introduce a sequence of languages serving as witness languages in the following text.

Examples. Let $j \in Nat$, let us consider the following alphabet $A_j = \Sigma_{ab} \cup \Sigma_{cd} \cup \{c_0, c_1, \ldots, c_{j-1}\}$, where $\Sigma_{ab} = \{a, b\}$ and $\Sigma_{cd} = \{c, d\}$. Further, let $L_{ab} = (\Sigma_{ab} \cdot \Sigma_{cd})^*$ and $L_{cd} = \Sigma_{cd} \cdot (\Sigma_{ab} \cdot \Sigma_{cd})^*$. We define a sequence of languages in the following way: for each $j \in Nat$ let

$$L_j = \bigcup_{0 \leq i < j} \{c_0 x w c_1 x w \ldots c_{i-1} x w c_i w c_{i+1} w \ldots c_{j-2} w c_{j-1} w \mid (x \in \Sigma_{ab} \ , \ w \in L_{cd}) \text{ or } (x \in \Sigma_{cd} \ , \ w \in L_{ab})\}.$$

For a word $c_0 w_1 c_1 w_2 c_2 \ldots c_{j-1} w_j$ from L_j, the words $w_1, \ldots, w_j$ are either empty words or words in which the symbols from Σ_{ab} and Σ_{cd} alternate and their last symbol is from Σ_{cd}. If at least one of the words $w_1, \ldots, w_j$ is nonempty, then

$$w_1 = w_2 = \ldots = w_i = xw \qquad w_{i+1} = w_{i+2} = \ldots = w_j = w$$

for some i, $1 \leq i \leq j$, some symbol $x \in \Sigma_{ab} \cup \Sigma_{cd}$ and some word w such that $xw \in L_{ab} \cup L_{cd}$.

Proposition 8. *For any $j \in Nat$: $L_j \in \mathcal{L}$(det-j-mon-RL).*

Proof. We outline *j-mon-RL*-automaton M recognizing L_j. Let $u \in (A_j)^*$ be the current word in the list of M (between the sentinels). M looks at the first two symbols of u, if they equal to:

- c_0c_1, then M checks the equality $u = c_0c_1...c_{j-1}$. In the positive case M accepts, in the negative case M rejects.
- c_0x, for some $x \in \Sigma_{ab} \cup \Sigma_{cd}$, then M checks whether u is of the form

$$c_0xw_1c_1xw_2c_2 \dots c_{i-1}xw_ic_iw_{i+1}c_{i+1}w_{i+2} \dots c_{j-2}w_{j-1}c_{j-1}w_j,$$

  where either $x \in \Sigma_{ab}$ and $w_1, \dots, w_j \in L_{cd}$ or $x \in \Sigma_{cd}$ and $w_1, \dots, w_j \in L_{ab}$. If the result of the check is negative, M rejects. In the positive case, M deletes the first symbol after c_{i-1} (i.e. x) during the corresponding cycle and it restarts.

Any computation of M can be divided into parts in which the same symbol x from $\Sigma_{ab} \cup \Sigma_{cd}$ is deleted. Each part comprising cycles deleting some symbol from Σ_{ab} can be followed by a part comprising cycles deleting some symbol from Σ_{cd} only, and vice versa. In an accepting computation the first part consists of not more than j cycles, all the remaining parts have exactly j cycles. In a rejecting computation the first and the last part cannot contain more than j cycles, all the remaining parts contain exactly j cycles.

It is easy to see that any computation (without the tail) can be partitioned into (at most) j monotonic subsequences of cycles, simply so that for the first subsequence the first cycles are taken from the parts, for the second one the second cycles of the parts and so on. This observation proves the claim. □

Proposition 9. *For any $j \in Nat$: $L_{j+1} \notin \mathcal{L}$(j-mon-RLW).*

Proof. Let us suppose that $M = (Q, \Sigma, \emptyset, k, I, q_0, Q_A, Q_R)$ is a j-monotonic *RLW*-automaton recognizing L_{j+1}. Let p be the constant determined for M by the Fact 5 (pumping lemma). Let us take a word $z_0 = c_0wc_1wc_2...c_jw \in L_{j+1}$ where $w \in L_{ab}$ and $|w| = n > 3kp$ (k is the size of the look-ahead window of M).

Let us consider an accepting computation $\mathcal{C}$ of M on z_0. This computation contains at least one cycle C_0. Otherwise because of the length of z_0, using Fact 5 we can construct an accepting tail for a word outside L_{j+1}. Let the resulting word after C_0 be z_1. Because $z_1 \in L_{j+1}$ ($\mathcal{C}$ is an accepting computation with the error preserving property) the only possible change performed by C_0 is the deletion of the first symbol after the symbol c_j. Therefore, $z_1 = c_0wc_1wc_2...w_1c_{j-1}wc_jw'$, where $w = xw'$ for some $x \in \Sigma_{ab}$.

We can see that $\mathcal{C}$ continues (for similar reasons) by another j cycles C_1,...C_j, where C_i deletes the first symbol after c_{j-i} (for $1 \leq i \leq j$). We can see that $D_r(C_0) < D_r(C_1) < ... < D_r(C_j)$. That is a contradiction with the assumption that M is a j-monotonic *RLW*-automaton. □

Theorem 10. *For any $j \in Nat$:*

$$\mathcal{L}(j\text{-mon-RLW}) \subset \mathcal{L}((j+1)\text{-mon-RLW}),$$
$$\mathcal{L}(\text{det-}j\text{-mon-RLW}) \subset \mathcal{L}(\text{det-}(j+1)\text{-mon-RLW}),$$
$$\mathcal{L}(j\text{-mon-RL}) \subset \mathcal{L}((j+1)\text{-mon-RL}),$$
$$\mathcal{L}(\text{det-}j\text{-mon-RL}) \subset \mathcal{L}(\text{det-}(j+1)\text{-mon-RL}).$$

Proof. We can see from the definition of j-monotonicity that the improper inclusions hold. The proper inclusions follow from the previous two propositions. □

Theorem 11. *For any $j \in Nat$:*

$$\mathcal{L}(RL) \subset \mathcal{L}(RLW),$$
$$\mathcal{L}(j\text{-mon-RL}) \subset \mathcal{L}(j\text{-mon-RLW}).$$

Proof. We can see the improper inclusions. We use the idea of the proof of Lemma 4.2 in [1]. Let us take the language
$L_1 = \{ f, ee \} \cdot \{ c^n d^n \mid n \geq 0 \} \bigcup \{ g, ee \} \cdot \{ c^n d^m \mid m > 2n \geq 0 \}$.
L_1 can be recognized by a monotonic RLW-automaton M in the following way:

- M immediately accepts the word f, otherwise
- if the word starts by fc then M simply deletes cd "in the middle" of the word and restarts,
- if the word starts by gc then M deletes cdd "in the middle" of the word and restarts,
- if the word starts by gd then M scans the rest of the word. If it contains only d's then it accepts otherwise rejects,
- if the word starts by ee then M nondeterministically rewrites ee by f or g and restarts.

It is easy to see that M is monotonic and $L(M) = L_1$.

L_1 cannot be recognized by any RL-automaton. For a contradiction let us suppose $L_1 = L(M)$ for some RL-automaton M with the look-ahead of the length k. Let us choose (and fix) a sufficiently large n ($n > k$) s.t. n is divisible by $p!$ (and hence by all $p_1 \leq p$) where p is taken from Fact 5. Now consider the first cycle C of an accepting computation of M on $eec^n d^n$. M can only shorten both segments of c's and d's in the same way, i.e. $eec^n d^n \Rightarrow_M eec^l d^l$, for some $l < n$. (Any accepting computation of M on $eec^n d^n$ has at least two cycles – otherwise using we can construct a word outside L_1 which will be accepted by M in one cycle) Fact 5. Due to Fact 5, d^n can be written $d^n = v_1 auv_2 auv_3$, $a = d$, $u = d^{k-1}$, $|auv_2| \leq p$, where M in the cycle C enters both occurrences of a in the same state.

Recall that nothing is deleted out of auv_2 in C and $|auv_2|$ divides n due to our choice of n. Then there is some i s.t. $d^{2n} = v_1(auv_2)^i auv_3$; hence $eec^n d^{2n} \notin L(M)$ but (by the natural extending of the cycle C) we surely get $eec^n d^{2n} \Rightarrow_M eec^l d^{n+l}$, where $2l < n + l$ and therefore $eec^l d^{n+l} \in L(M)$ – a contradiction with the error preserving property (Fact 1). □

4 Additional Remarks

Let us note that our aim is to present basic formal considerations which should serve for a preparation of a software environment performing analysis by reduction for Czech or other natural languages like German or Latin. We are deeply influenced by our linguistic tradition. Our linguists feel that there is something like 'basic word-order' in Czech sentences on the one hand, and several types of relaxation of this basic word-order on the other hand. We feel that the monotonic computations can represent the analysis by reduction of sentences in the basic word-order, the j-monotonic computations can represent the more relaxed variants of the word-order. The j-monotonicity allows to formulate 'characteristic' restrictions of the relaxation for individual natural languages. Cooperating with linguists, we are looking for the adequate 'characteristic' restrictions for Czech.

We can see that already the second witness language L_2 is not a context-free language. These observations and the presented results give us the opportunity to consider the j-monotonicity for the degree of non-context-freeness of RL-languages, RLW-languages and for their deterministic variants.

Let us outline some conjectures and ideas for the future work. We believe that we will be able to show a polynomial algorithm for recognition of j-monotonic $RLW(W)$-languages. We will be interested in nondeterministic RLW-automata which keep (completely or by some instructions only) the correctness preserving property. This type of consideration is useful for the technology of localization of syntactic inconsistencies (grammar-checking). We believe that we are able to show, that any RLW-, RRW-automaton, which fulfills the correctness preserving property, can be transformed into an equivalent deterministic RLW-automaton. The results from [7] ensure the fact that there are nondeterministic RRW-automata with the correctness preserving property, which cannot be transformed into the equivalent deterministic RRW-automaton. This observation stresses the meaning of the notion of two-way restarting automata for applications and theoretical considerations as well. The more general study following this direction can lead us to the comparison of the results from [5] with the properties of $RLWW$-automata. We consider to look for classes of g-systems (see [8]) similar to the classes of restarting automata studied here.

Let us note at the end that there is an essential difference between one-way restarting automata and two-way restarting automata: two-way restarting automata allow infinite computations and (nondeterministic) cycles of an unlimited length. This fact will turn our attention to the study of normal forms of $RLWW$-automata in the near future.

References

1. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: On Monotonic Automata with a Restart Operation. Journal of Automata, Languages and Combinatorics 4 (1999) 287–311
2. Holan, T., Kuboň, V., Oliva, K., Plátek, M.: Two useful measures of word order complexity. In: Polguére, A., Kahane, S. (eds): Proceedings of the Workshop

'Processing of Dependency-Based Grammars', Université de Montréal, Montréal, Quebeck, Canada, August (1998) 21–28
3. Holan, T., Kuboň, V., Oliva, K., Plátek M.: On Complexity of Word Order. T.A.L, vol. 41. n^o 1 (2000) 243–267
4. Mráz, F.: Lookahead hierarchies of restarting automata. In: O.Boldt, H. Jürgensen (eds.): Pre-Procedings of DCAGRS 2000, Dept. of Comp. Science, The University of Western Ontario, London, Canada (2000)
5. Niemann, G., Otto, F.: Restarting automata, Church Rosser languages, and representations of r.e. languages. In: Rozenberg G., Thomas, W. (eds.): Developments in Language Theory. Foundations, Applications, and Perspectives. Proceedings DLT'99 Word Scientific, Singapore (2000)103–114
6. Mráz,F., Plátek, M., Jančar P., Vogel, J.: Monotonic Rewriting Automata with a Restart Operation. In: Plášil, F., Jeffery, K., J.: (eds.): SOFSEM'97. Theory and Practice of Informatics. F. Lecture Notes in Computer Science, Vol. 1338. Springer Verlag, Berlin Heidelberg New York (1997) 505–512
7. Plátek, M., Mráz, F.: Degrees of (non)monotonocity of RRW-automata. In: Dassow, J., Wotschke, D. (eds): Pre-Proceedings of DCAGRS 2001, Dept. of Comp. Science, Otto-von-Guericke-University of Magdeburg, Germany (2001) 159–166
8. Rovan, B.: Complexity Classes of g-systems are AFL. *Acta Mathematica Universitatis Comenianae*, XLVIII -XLIX (1986) 283–297
9. Straňáková, M.: Selected Types of Pg-Ambiguity. The Prague Bulletin of Mathematical Linguistics 72 (1999) 29–82
10. S. H. von Solms. The characterization by automata of certain classes of languages in the context sensitive area. *Information and Control (now Information and Computation)* 27(3) (1975) 262–271

P-Hardness of Equivalence Testing on Finite-State Processes*

Zdeněk Sawa and Petr Jančar

Department of Computer Science, Technical University of Ostrava (FEI VŠB)
17. listopadu 15, CZ-708 33 Ostrava, Czech Republic
zdenek.sawa@vsb.cz, petr.jancar@vsb.cz

Abstract. The paper shows a simple LOGSPACE-reduction from the boolean circuit value problem which demonstrates that, on finite labelled transition systems, deciding an arbitrary relation which subsumes bisimulation equivalence and is subsumed by trace preorder is a polynomial-time-hard problem (and thus can not be expected to be efficiently parallelizable). By this, the result of Balcázar, Gabarró and Sántha (1992) for bisimilarity is substantially extended.

1 Introduction

It is not necessary to emphasize the importance of theoretical foundations for design, analysis and verification of (complex) systems, especially concurrent systems, which are composed from communicating components running in parallel. One particular research area studies computational complexity of various verification problems for finite state systems.

A general model of such systems is given by so called labelled transition systems (LTSs for short), which capture the notion of (global) states and their changes by performing transitions – which are labelled by actions (or action names). Since here we deal only with *finite LTSs*, they can be viewed as classical nondeterministic finite automata.

We consider the verification problem of testing behavioural equivalences on finite LTSs. Let us recall that classical language equivalence turned out to be mostly too coarse, and it was *bisimilarity* which was established as the most appropriate notion of general behavioural equivalence (cf. [6]). Nevertheless, other notions of equivalences (or preorders) turned out to be useful for more specific aims. Van Glabbeek [9] classified these equivalences in a hierarchy called *linear time/branching time spectrum*. The diagram in Fig. 1 shows most prominent members of the hierarchy and their interrelations (the arrow from R to S means that equivalence R is finer than equivalence S). As depicted, bisimilarity (i.e., bisimulation equivalence) is the finest in the spectrum; the coarsest is *trace equivalence*, which is the classical language equivalence when we assume all states as accepting (i.e., we are interested in the set of all sequences of actions which are performable).

* Supported by the Grant Agency of the Czech Republic, Grant No. 201/00/0400

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 326–335, 2001.

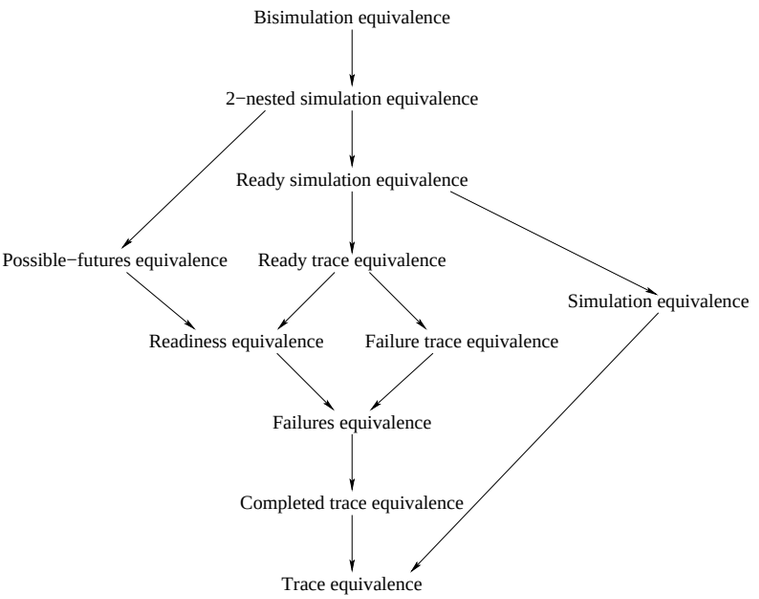


Fig. 1. The linear time/branching time spectrum

For the aims of automated verification, a natural research task is to establish the complexity of the problem

INSTANCE: A finite labelled transition system and two of its states, p and q.

QUESTION: Are p and q equivalent with respect to X ?

for each equivalence X in the spectrum. From language theory results we can easily derive PSPACE-completeness of trace equivalence. On the other hand, there is a polynomial time algorithm for bisimilarity [7,4]. The paper [3] is a (preliminary) survey of all results in this area. Loosely speaking, 'trace-like' equivalences (on the bottom part of the spectrum) turn out to be PSPACE-complete, the 'simulation-like' equivalences (on the top of spectrum) are in PTIME. Balcázar, Gabarró and Sántha [1] have considered the question of an efficient parallelization of the algorithm for bisimilarity, and they have shown that the problem is P-complete (i.e., all polynomial-time problems are reducible to this problem by a LOGSPACE-reduction). This shows that the bisimilarity problem seems to be 'inherently sequential'; we can not get a real gain by parallelization, unless NC = PTIME, which is considered to be very unlikely (cf. e.g. [2]).

Paper [1] shows a (LOGSPACE) reduction from (a special version of) the boolean circuit value problem which is well-known to be P-complete. The reduction is aiming just at bisimilarity; in particular, it does not show P-hardness of other 'simulation-like' equivalences (which are known to be in PTIME as well).

In this paper, we show another reduction from (a less constrained version of) circuit value problem which we find simple and elegant, and which immediately

shows that deciding an *arbitrary* relation which subsumes bisimulation equivalence and is subsumed by trace equivalence (more generally, by trace preorder) is P-hard. By this, the result of [1] is substantially extended; though it brings nothing new for the ('trace-like') equivalences for which PSPACE-hardness has been established, it surely is relevant for 'simulation-like' equivalences (those between bisimulation and simulation equivalences in the spectrum).

Section 2 gives necessary definitions and formulates the main result, and Section 3 contains the technical proof. We then add remarks on a possibility to 'lift' the result to settle a conjecture in [8].

2 Definitions

A *labelled transition system* (an *LT-system* for short) is a tuple $\langle S, Act, \longrightarrow \rangle$ where S is a set of *states*, Act is a set of *actions* (or *labels*), and $\longrightarrow \subseteq S \times Act \times S$ is a *transition relation*. We write $p \xrightarrow{a} q$ instead of $\langle p, a, q \rangle \in \longrightarrow$; we also use $p \xrightarrow{w} q$ for finite sequences of actions ($w \in Act^*$) with the natural meaning. In this paper, we only consider *finite* LT-systems, where both the sets S and Act are finite.

We need precise definitions of trace and bisimulation equivalences on states in LT-systems. Let us remark that it is sufficient for us only to relate states of *the same* LT-system; this could be naturally extended for states of *different* LT-systems (we can take disjoint union of these).

For a state p of an LT-system $\langle S, Act, \longrightarrow \rangle$, we define the set of its *traces* as $tr(p) = \{ w \in Act^* \mid p \xrightarrow{w} q \text{ for some } q \in S \}$. States p and q are *trace equivalent*, iff $tr(p) = tr(q)$; they are in *trace preorder* iff $tr(p) \subseteq tr(q)$.

A binary relation $\mathcal{R} \subseteq S \times S$ on the state set of an LT-system $\langle S, Act, \longrightarrow \rangle$ is a *simulation* iff for each $(p, q) \in \mathcal{R}$ and each $p \xrightarrow{a} p'$ there is some $q \xrightarrow{a} q'$ such that $(p', q') \in \mathcal{R}$. $\mathcal{R}$ is a *bisimulation* iff both $\mathcal{R}$ and its inverse $\mathcal{R}^{-1}$ are simulations. States p, q are *bisimulation equivalent* (or *bisimilar*), written $p \sim q$, iff $(p, q) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$.

We recall that a problem P is P-hard if any problem in PTIME can be reduced to P by a LOGSPACE reduction; recall that a Turing machine performing such a reduction uses work space of size at most $O(\log n)$, where n denotes the size of the input on a read-only input tape (the output is written on a write-only output tape and its size may be polynomial). A problem P is P-complete if P is P-hard and $P \in$ PTIME.

Remark. We recall that if a problem P is P-hard then it is unlikely that there exists an efficient parallel algorithm deciding P. 'Efficient' here means working in polylogarithmic time, i.e., with the time complexity in $O(\log^k n)$ for some constant k, while the number of the processors used is bounded by a polynomial in the size n of the input instance. (See e.g. [2] for further details.)

We say that a *relation* X (relating states in transition systems) is *between bisimilarity and trace preorder* iff $p \sim q$ implies pXq and pXq implies $tr(p) \subseteq tr(q)$. And we formulate the main result of our paper:

Theorem 1. *For any relation X between bisimilarity and trace preorder, the following problem is P-hard:*

INSTANCE*: A finite labelled transition system and two of its states, p and q.*
QUESTION*: Is pXq ?*

We shall prove this in the next section by a LOGSPACE reduction from the problem of monotone boolean circuit value, mCVP for short.

To define mCVP we need some definitions. *Monotone boolean circuit* is a directed, acyclic graph, in which the nodes (also called *gates*) are either of in-degree zero (*input* gates) or of indegree 2 (*non-input* gates). There is exactly one node of outdegree zero (the *output* gate). Non-input gates are labelled by one of $\{\wedge, \vee\}$ (notice that in monotone circuit no $\neg$-gates are used). *Input of the circuit* is an assignment of boolean values (i.e., values from the set $\{0, 1\}$) to input gates. A value on a non-input gate labelled with $\wedge$ (resp. $\vee$) is computed as the conjunction (resp. disjunction) of values on its ancestors. The output value of the circuit is the value on the output gate.

The mCVP problem is defined as follows:

INSTANCE: A monotone boolean circuit with its input.
QUESTION: Is the output value 1 ?

The problem is well-known to be P-complete (cf. e.g. [2]). We also recall that if P_1 is P-hard and P_1 is LOGSPACE reducible to P_2 then P_2 is P-hard as well.

In the next section we show how, given an instance of mCVP, to construct (in LOGSPACE) a transition system with two designated states p, q so that if the output value of the circuit is 1 then $p \sim q$, and if the output value is 0 then $tr(p) \not\subseteq tr(q)$. So for any relation X between bisimilarity and trace preorder, pXq iff the output value is 1. This immediately implies the theorem.

3 The Reduction

Let us have an instance of mCVP where the set of gates is $V = \{1, 2, \ldots, n\}$. For every non-input gate i, we define $l(i), r(i)$ (left and right ancestor of i) to be the gates, such that there are edges from $l(i)$ and $r(i)$ to i (and $l(i) \neq r(i)$). For technical reasons we assume the gates are topologically ordered, i.e., for every non-input gate i we have $i > l(i) > r(i)$, and n is the output gate (mCVP is still P-complete under this assumption). We define a function $t : V \longrightarrow \{0, 1, \wedge, \vee\}$, where $t(i)$ denotes the 'type' of gate i:

$$t(i) = \begin{cases} 0 & \text{if } i \text{ is an input gate with value 0} \\ 1 & \text{if } i \text{ is an input gate with value 1} \\ \wedge & \text{if } i \text{ is a non-input gate labelled with } \wedge \\ \vee & \text{if } i \text{ is a non-input gate labelled with } \vee \end{cases}$$

Let $v_i \in \{0, 1\}$ denote the actual value on gate i, i.e., if i is an input gate then $v_i = t(i)$, and if i is a non-input gate, then v_i is computed from $v_{l(i)}, v_{r(i)}$ using operation indicated by $t(i)$.

We assume, that an input instance of mCVP consists of n and of values $l(i), r(i)$ and $t(i)$ for every $1 \leq i \leq n$ (in fact, it suffices that these values can be computed from the input instance in LOGSPACE).

Given an instance of mCVP, we construct LT-system $\Delta = \langle S, Act, \longrightarrow \rangle$, where $Act = \{0, 1\}$ and S is a union of the following sets:

- $\{p_i \mid 0 \leq i \leq n\}$,
- $\{q_i^j \mid 1 \leq j \leq i \leq n\}$,
- $\{q_i^{j,k} \mid 1 \leq k < j \leq i \leq n\}$.

We organize states in S into *levels*. Level i (where $0 \leq i \leq n$) contains all states with the same lower index i, i.e., $\{p_i\} \cup \{q_i^j \mid 1 \leq j \leq i\} \cup \{q_i^{j,k} \mid 1 \leq k < j \leq i\}$ as depicted in Fig. 2 (already with *some* transitions).

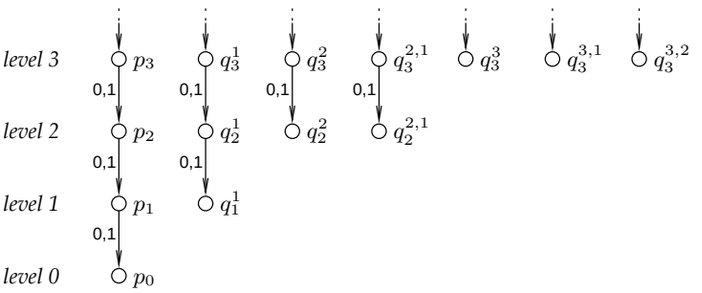


Fig. 2. The states of Δ organized into levels

Informally speaking, the intended purpose of states q_i^j is to 'test' whether $v_j = 1$. State q_i^j is viewed as 'successful' if indeed $v_j = 1$ and is 'unsuccessful' if $v_j = 0$. Similarly, state $q_i^{j,k}$ is 'successful' if $v_j = 1$ and $v_k = 1$, and 'unsuccessful' otherwise.

The way we construct the transition relation $\longrightarrow$ will guarantee, that each successful state q on level i (i.e., of the form q_i^j or $q_i^{j,k}$) is bisimilar with p_i, and if q (on level i) is unsuccessful, then $tr(p_i) \not\subseteq tr(q)$. So p_n and q_n^n are two distinguished states announced in the previous section, with the required property, that if $v_n = 1$, then $p_n \sim q_n^n$, and otherwise $tr(p_n) \not\subseteq tr(q_n^n)$.

Transition relation $\longrightarrow$ will contain only transitions going from states on level i to states on level $i-1$. We will construct transitions level by level, starting with transitions going from (states on) level 1 to level 0, then from level 2 to level 1 and so on. The actual transitions going from level i to level $i-1$ will depend on $t(i)$, $l(i)$ and $r(i)$, so in this sense level i corresponds to gate i. It is worth to emphasize here, that the added transitions does not depend on information, whether a state is successful or not.

Now we describe in detail the construction of transitions leading from states on level i to states on level $i-1$.

Firstly, the following transitions are always possible from states on level i (see Fig. 2):

$$\begin{aligned} &p_i \xrightarrow{0,1} p_{i-1}, \\ &q_i^j \xrightarrow{0,1} q_{i-1}^j \qquad \text{for each } j, \text{ such that } 1 \le j < i, \\ &q_i^{j,k} \xrightarrow{0,1} q_{i-1}^{j,k} \qquad \text{for each } j,k, \text{ such that } 1 \le k < j < i. \end{aligned}$$

(We write $q \xrightarrow{0,1} q'$ instead of $q \xrightarrow{0} q'$ and $q \xrightarrow{1} q'$.) Depending on $t(i)$, some other transitions leading from states on level i may be added.

To simplify the notation, we need some further definitions. Let Q_i be the set of all states of the form q_i^j or $q_i^{j,k}$ on level i, i.e., $Q_i = \{q_i^j \mid 1 \le j \le i\} \cup \{q_i^{j,k} \mid 1 \le k < j \le i\}$. Let *Succ* be the set of all successful states, i.e., $Succ = \{q_i^j \in S \mid v_j = 1\} \cup \{q_i^{j,k} \in S \mid v_j = 1 \text{ and } v_k = 1\}$.

We use w_i to denote the sequence of actions that correspond to actual values on gates $i, i-1, \ldots, 1$, i.e., $w_1 = v_1$ and $w_i = v_i w_{i-1}$ for $i > 1$.

We will construct Δ in such a way, that each level i will satisfy the following condition:

$$\text{For each } q \in Q_i\text{:} \quad \text{if } q \in Succ, \text{ then } p_i \sim q, \text{ otherwise } w_i \notin tr(q). \tag{1}$$

Notice, that $w_i \notin tr(q)$ implies $tr(p_i) \not\subseteq tr(q)$, because $tr(p_i)$ contains all possible traces of length i over *Act*, so in particular $w_i \in tr(p_i)$.

Now we describe the remaining transitions together with a proof that each level satisfies the condition (1). We proceed by induction on i.

Remark. To show for some $q \in Q_i$ (such that $q \notin Succ$) that $w_i \notin tr(q)$, it suffices to show for every q', such that there is a transition $q \xrightarrow{v_i} q'$, that $w_{i-1} \notin tr(q')$, i.e., to show that $q' \notin Succ$ (because $q' \notin Succ$ implies $w_{i-1} \notin tr(q')$ by induction hypothesis).

Base of induction ($i = 1$): Because the circuit is topologically ordered, gate 1 must be an input gate, so $t(1)$ is either 0 or 1. If $t(1) = 0$, we do not add any transitions (see Fig. 3). Because $t(1) = 0$ implies $v_1 = 0$, we have $q_1^1 \notin Succ$.

Fig. 3. The construction for $i = 1$

Obviously $w_1 \notin tr(q_1^1)$, so the condition (1) holds.

If $t(i) = 1$, we add transitions $q_1^1 \xrightarrow{0,1} p_0$ (see Fig. 3). From $t(1) = 1$ follows $q_1^1 \in Succ$, and $p_1 \sim q_1^1$ is obvious, so again the condition (1) holds.

Inductive step ($i > 1$):

- If $t(i) = 0$: We do not add any transitions. We first consider q_i^j where $i > j$ (see Fig. 4).

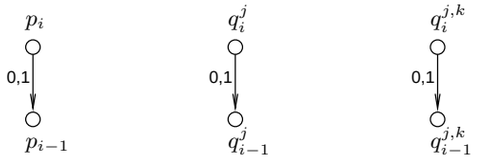


Fig. 4. The transitions added if $i > j$ and $t(i) \in \{0, 1, \wedge\}$

It is clear, that $q_i^j \in \mathit{Succ}$ iff $q_{i-1}^j \in \mathit{Succ}$, so q_i^j satisfies the condition (1), as can be easily checked (by induction hypothesis $q_{i-1}^j \in \mathit{Succ}$ implies $p_i \sim q_{i-1}^j$, and $q_{i-1}^j \notin \mathit{Succ}$ implies $w_{i-1} \notin tr(q_{i-1}^j)$). The proof for $q_i^{j,k}$, where $i > j$, is similar.
Now, let q be q_i^j or $q_i^{j,k}$, where $i = j$, i.e., one of $q_i^i, q_i^{i,k}$. From $t(i) = 0$ we have $v_i = 0$, and this implies $q \notin \mathit{Succ}$. Obviously $w_i \notin tr(q)$, because no transitions from q are possible.

- If $t(i) = 1$: If q is q_i^j or $q_i^{j,k}$, where $i > j$, the situation is exactly the same as in the previous case (see Fig. 4). So let q be q_i^i or $q_i^{i,k}$. We add transitions from q as depicted in Fig. 5.

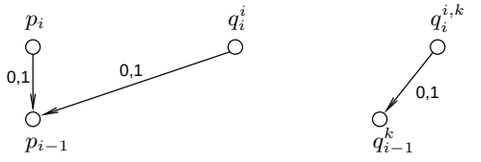


Fig. 5. The transitions added if $i = j$ and $t(i) = 1$

If q is q_i^i, then surely $q \in \mathit{Succ}$ (because $v_i = 1$), and obviously $p_i \sim q$.
If q is $q_i^{i,k}$, we can imagine this as the situation when it was tested that $v_i = 1$ and it is continued with testing that $v_k = 1$. Because $v_i = 1$, we have $q_i^{i,k} \in \mathit{Succ}$ iff $q_{i-1}^k \in \mathit{Succ}$, so the condition (1) is satisfied in $q_i^{i,k}$, as can be easily checked.

- If $t(i) = \wedge$: If q is q_i^j or $q_i^{j,k}$, where $i > j$, the situation is the same as in two previous cases (see Fig. 4). So let q be q_i^i or $q_i^{i,k}$. We add transitions from q as depicted in Fig. 6.
If $q \in \mathit{Succ}$, then $v_i = 1$. Because $t(i) = \wedge$, $v_i = 1$ implies $v_{l(i)} = 1$ and $v_{r(i)} = 1$, so $q_{i-1}^{l(i),r(i)} \in \mathit{Succ}$. From this $p_{i-1} \sim q_{i-1}^{l(i),r(i)}$ by induction hypothesis, so $p_i \xrightarrow{0}$

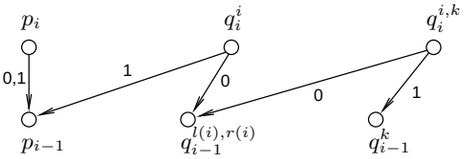


Fig. 6. The transitions added if $i = j$ and $t(i) = \wedge$

p_{i-1} can be matched by $q \xrightarrow{0} q_{i-1}^{l(i),r(i)}$ (and vice versa). Other transitions can be matched also (notice that $p_{i-1} \sim q_{i-1}^k$ if $q_i^{i,k} \in \mathit{Succ}$), so we have $p_i \sim q$.
Now consider the case $q \notin \mathit{Succ}$. If q is q_i^i, then $v_i = 0$, and this implies $v_{l(i)} = 0$ or $v_{r(i)} = 0$, so $q_{i-1}^{l(i),r(i)} \notin \mathit{Succ}$, and from this we have $w_i \notin tr(q_i^i)$, because $q_i^i \xrightarrow{0} q_{i-1}^{l(i),r(i)}$ is the only transition labelled with 0 leading from q_i^i.
If q is $q_i^{i,k}$, then v_i either 0 or 1. The case $v_i = 0$ is similar as before. So let us have $v_i = 1$. Then $v_k = 0$, and $q_{i-1}^k \notin \mathit{Succ}$, from which $w_i \notin tr(q_i^{i,k})$ follows.

- If $t(i) = \vee$: For every $q \in Q_i$ we add transitions $q \xrightarrow{0} q_{i-1}^{l(i)}$ and $q \xrightarrow{0} q_{i-1}^{r(i)}$. We also add transitions $p_i \xrightarrow{0} q_{i-1}^{l(i)}$ and $p_i \xrightarrow{0} q_{i-1}^{r(i)}$.
Let q be q_i^j where $i > j$ as in Fig. 7. (The case when q is $q_i^{j,k}$ where $i > j$ is almost identical.) If $q_i^j \in \mathit{Succ}$, then $q_{i-1}^j \in \mathit{Succ}$, so by induction hypothesis

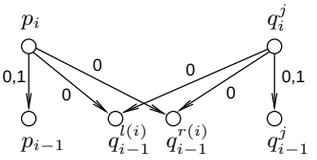


Fig. 7. The transitions added if $i > j$ and $t(i) = \vee$

$p_{i-1} \sim q_{i-1}^j$. From this $p_i \sim q_i^j$ easily follows, because every possible transition can be matched (for example $p_i \xrightarrow{0} q_{i-1}^{l(i)}$ by $q_i^j \xrightarrow{0} q_{i-1}^{l(i)}$, etc.).
If $q_i^j \notin \mathit{Succ}$, then $q_{i-1}^j \notin \mathit{Succ}$, so $w_{i-1} \notin tr(q_{i-1}^j)$. We need to consider two cases, v_i is either 1 or 0. If $v_i = 1$, we have $w_i \notin tr(q_i^j)$, because $q_i^j \xrightarrow{1} q_{i-1}^j$ is the only possible transition labelled with 1. If $v_i = 0$, then $v_{l(i)} = v_{r(i)} = 0$, so $q_{i-1}^{l(i)}, q_{i-1}^{r(i)} \notin \mathit{Succ}$, and from this $w_i \notin tr(q_i^j)$ easily follows.
Let us now consider the case, where q is q_i^i or $q_i^{i,k}$. We add transitions as depicted in Fig. 8.
If $q \in \mathit{Succ}$, then $p_i \xrightarrow{0} p_{i-1}$ can be matched by (at least) one of $q \xrightarrow{0} q_{i-1}^{l(i)}$, $q \xrightarrow{0} q_{i-1}^{r(i)}$, because from $v_i = 1$ we have $v_{l(i)} = 1$ or $v_{r(i)} = 1$. All other transitions are matched as in previous cases, so $p_i \sim q$.

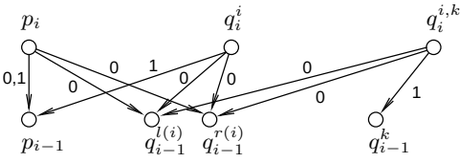


Fig. 8. The transitions added if $i = j$ and $t(i) = \vee$

Now suppose $q \notin \mathit{Succ}$. If q is q_i^i, then $v_i = 0$, so $v_{l(i)} = v_{r(i)} = 0$, and $q_{i-1}^{l(i)}, q_{i-1}^{l(i)} \notin \mathit{Succ}$. From this $w_i \notin tr(q_i^i)$ easily follows. The case, when q is $q_i^{i,k}$, is similar if $v_i = 0$. The only remaining possibility is that $v_i = 1$ and $v_k = 0$. Then $q_{i-1}^k \notin \mathit{Succ}$, so obviously $w_i \notin tr(q_i^{i,k})$.

To finish the proof of Theorem 1, it remains to show that the described reduction is in LOGSPACE.

The algorithm performing the reduction requires only fixed number of variables, such as i, j, k, and values of $t(i), l(i), r(i)$ for every i, that are part of the read-only input instance of mCVP. In particular, to construct transitions leading from a given state, only a fixed number of such values is needed. Obviously, $O(\log n)$ bits are sufficient to store them. No other information is needed during the construction, so the algorithm uses work space of size $O(\log n)$. This finishes the proof.

Remark. LT-system Δ contains $O(n^3)$ states and also $O(n^3)$ transitions (because the number of possible transitions in every state is in $O(1)$ and does not depend on n). Notice, that a state of the form $q_i^{j,k}$ is not reachable from p_n nor q_n^n, if there is no $i' \in V$, such that $t(i') = \wedge$, $l(i') = j$ and $r(i') = k$, as can be easily proved by induction. There is at most $O(n)$ pairs j, k, where such i' exists, and it is possible to test for given j, k the existence of i' in LOGSPACE, so we can add to Δ only those states $q_i^{j,k}$, where such i' exists. In this way we can reduce the number of states (and transitions) to $O(n^2)$.

Additional Remarks

We have considered complexity as a function of the size of given labelled transition systems (which describe the state space explicitly). Rabinovich [8] considered the problem for concurrent systems of finite agents, measuring complexity in the size of (descriptions of) such systems; the corresponding (implicitly represented) state space is exponential in that size. He conjectured that all relations between bisimilarity and trace equivalence are EXPTIME-hard in this setting. Laroussinie and Schnoebelen [5] have confirmed the conjecture partially. They showed EXPTIME-hardness for all relations between simulation preorder and bisimilarity, and EXPSPACE-hardness on the 'trace equivalence end' of van Glabbeek's spectrum. We plan to explore the probable possibility that our construction can be 'lifted' (i.e., programmed concisely by a concurrent system of finite agents), which would settle Rabinovich's conjecture completely.

Acknowledgement

We thank Philippe Schnoebelen for fruitful discussions.

References

1. J. Balcázar, J. Gabarró, and M. Sántha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4(6A):638–648, 1992.
2. A. Gibbons and W. Rytter. *Efficient Parallel Algorithms*. Cambridge University Press, 1988.
3. H. Hüttel and S. Shukla. On the complexity of deciding behavioural equivalences and preorders. Technical Report SUNYA-CS-96-03, State University of New York at Albany, Dec. 28, 1996. Also available at `http://www.cs.auc.dk/~hans/Publications/pubs.html`.
4. P. Kanellakis and S. Smolka. CCS expressions, finite state processes and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.
5. F. Laroussinie and P. Schnoebelen. The state explosion problem from trace to bisimulation equivalence. In *Proc. FoSSaCS 2000*, volume 1784 of *Lecture Notes in Computer Science*, pages 192–207. Springer Verlag, 2000.
6. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
7. R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16:937–989, 1987.
8. A. Rabinovich. Complexity of equivalence problems for concurrent systems of finite agents. *Information and Computation*, 139(2):111–129, 1997.
9. R. van Glabbeek. The linear time – branching time spectrum. In *Proceedings CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297, Amsterdam, 1990. Springer-Verlag.

Software Geography: Physical and Economic Aspects

Vaughan R. Pratt

[1] Stanford University pratt@cs.stanford.edu
[2] Tiqit Computers pratt@tiqit.com

Abstract. To the metaphors of software engineering and software physics can be added that of software geography. We examine the physical and economic aspects of the Software Glacier (once an innocent bubbling brook, now a vast frozen mass of applications imperceptibly shaping both the Hardware Shelf below and User City above), Quantum Planet (colonization of which could be fruitful if and when it becomes practical), and Concurrency Frontier (an inaccessible land with rich resources that we project will be exploited to profound economic effect during the next half-century).

1 The Software Glacier

It used to be that the computer was the hard thing to build, programming it was almost an afterthought. Building a computer was hard because the problem was so constrained by the laws of nature. Writing software on the other hand was very easy, being unconstrained other than by the size and speed of available function units, memory, and storage.

But as time went by it became harder to write software and easier to build computers.

One might suppose this to be the result of the hardware constraints somehow loosening up while the software constraints tightened. However these basic constraints on hardware and software have not changed essentially in the past half century other than in degree. When hardware design was a vertical operation, with complete computers being designed and built by single companies, hardware designers had only the laws of nature to contend with. But as the economics of the market place gradually made it a more horizontal operation, with different specialists providing different components, the need for specifiable and documentable interoperability sharply increased the constraints on the hardware designers.

At the same time clock rates, memory capacity, and storage capacity have increased geometrically, doubling every eighteen or so months, with the upshot that today one can buy a gigabyte of DRAM and a 60 gigabyte hard drive for $150 each. This has enormously reduced the speed and storage constraints on software developers.

So on the basis of how the contraints have evolved, hardware should be harder than ever and software easier than ever.

L. Pacholski and P. Ružička (Eds.): SOFSEM 2001, LNCS 2234, pp. 336–345, 2001.

Besides constraints, both tasks have become more complex. Hardware increases in complexity as a result of decreasing line geometries combining with increasing die size to permit more circuits to be packed into each chip. This has two effects on the complexity of the computer as a whole. On the one hand the additional per-chip functionality and capacity increases the overall computer complexity. On the other hand it also facilitates a greater degree of integration: more of the principal components of the computer can be packed into each chip, simplifying the overall system at the board level.

The complexity of software (but not its utility and effectiveness) increases linearly with each of time, number of programmers, and effectiveness of programming tools. There is some attrition as retiring technology obsoletes some packages and others receive major overhauls, but mostly software tends to accumulate.

So we have two effects at work here, constraints and complexity.

Hardware is complex at the chip level, but is getting less complex at the board level. At the same time hardware design is so constrained today as to make it relatively easy to design computers provided one has the necessary power tools. There are so few decisions to make nowadays: just pick the components you need from a manageably short list, and decide where to put them. The remaining design tasks are forced to a great extent by the design rules constraining device interconnection.

The combined effect of contraints and complexity on hardware then is for system design to get easier. Chip design remains more of a challenge, but even there the design task is greatly facilitated by tight design rules. Overall, hardware design is getting easier.

Software design on the other hand is by comparison hardly constrained at all. One might suppose interoperability to be a constraint, but to date interoperability has been paid only lip service. There is no rigid set of rules for interoperability comparable to the many design rules for hardware, and efforts to date to impose such rules on software, however well-intentioned, have done little to relieve the prevailing anarchy characterizing modern software.

The dominant constraint with software today is its sheer bulk. This is not what the asymptotics of the situation predict, with hardware growing exponentially and software accumulating linearly. However the quantity of modern software is the result of twenty years of software development, assuming we start the clock with the introduction of the personal computer dominating today's computing milieu. Multiply that by the thousands of programmers gainfully or otherwise employed to produce that software, then further multiply the result by the increase in effectiveness of programming tools, and the result is an extraordinary volume of software.

Software today has much in common with a glacier. It is so huge as to be beyond human control, even less so than with a glacier, which might at least be deflected with a suitable nuclear device—there is no nuclear device for software.

Software creeps steadily forward as programmers continue to add to its bulk. In that respect it differs from a glacier, which is propelled by forces of nature.

This is not to say however that the overall progress of software is under voluntary human control. In the small it may well be, but in the large there is no overall guiding force. Software evolves by freewill locally, but globally determinism prevails.

Unlike a glacier, software is designed for use. We can nevertheless continue the analogy by viewing software users as residents of a city, User City, built on the glacier. The users have no control over the speed, direction, or overall shape of the glacier but are simply carried along by it. They can however civilize the surface, laying in the user interface counterparts of roads, gardens, and foundations for buildings. The executive branch of this operation is shared between the software vendors, with the bulk currently concentrated in Microsoft. The research branch is today largely the bailiwick of SIGCHI, the Special Interest Group on Computer-Human Interfaces.

Hardware is to software as a valley is to the glacier on it. There is only one glacier per valley, consisting of all the software for the platform constituting that valley. In the case of Intel's Pentium and its x86 clones for example, much of the software comes from Microsoft, but there are other sources, most notably these days Linux.

A major breakdown in this analogy is that whereas mortals have even less control over the valley under a real glacier than over the glacier itself, the ease of designing modern hardware gives us considerable control of what the software glacier rides on. We are thus in the odd situation of being able to bring the valley to the glacier.

This is the principle on which David Ditzel founded Transmeta six years ago. When Ditzel and his advisor David Patterson worked out the original RISC (Reduced Instruction Set Computer) concept in the early 1980s, the thinking back then was that one would build a RISC machine which would supplant the extant CISC (Complex ditto) platforms.

The building was done, by Stanford spinoffs Sun Microsystems and MIPS among others, but not the supplanting. In a development that academic computer architects love to hate, Intel's 8008 architecture evolved through a series of CISC machines culminating in the Pentium, a wildly successful machine with many RISC features that nevertheless was unable to realize the most important benefit of RISC, namely design simplicity, due to the retention of its CISC origins.

Ditzel's idea, embarked on in 1995, was to build a Pentium from scratch, seemingly in the intellectual-property shelter of IBM's patent cross-licensing arrangements with Intel. The CISC soul of this new machine was to be realized entirely in software. Transmeta's Crusoe is a pure RISC processor that realizes the complexity benefits of RISC yet is still able to run the Pentium's heavily CISC instruction set.

Whether the benefits of a pure RISC design are sufficient to meet the challenges of competing head-on with Intel remains to be seen. (The stock market seems to be having second thoughts on Transmeta's prospects, with Transmeta's stock currently trading at $2.50, down from $50 last November.) The point to

be noted here is that Ditzel saw the software benefits of going to the glacier. Just as bank robber Willie Sutton knew where the money was, Ditzel could see where the software was. That insight is independent of whether pure RISC is a sound risk.

The question then naturally arises, must we henceforth go to the glacier, or is it not yet too late to start a new software glacier in a different valley?

Apropos of this question, a new class of processors is emerging to compete with the Pentium. The Pentium's high power requirements make it a good fit for desktops and large-screen notebooks where the backlighting power is commensurate with the CPU power and there is room for a two-hour battery. However the much smaller cell phones and personal digital assistants (PDAs) have room for only a tiny battery, ruling out the Pentium as a practical CPU choice. In its place several low-powered CPUs have emerged, notably Motorola's MC68328 (Dragonball) as used in the Palm Pilot, the Handspring Visor, and the Sony CLIÉ; the MIPS line of embedded processors, made by MIPS Technologies, Philips, NEC, and Toshiba, as used in several brands of HandheldPC and PalmPC including the Casio Cassiopeia; the Hitachi SH3 and SH4 as used in the Compaq Aero 8000 and the Hitachi HPW-600 (and the Sega Dreamcast); and Intel's Strongarm.

Software for these is being done essentially from scratch. Although Unix (SGI Irix) runs on MIPS, it is too much of a resource hog to be considered seriously for today's lightweight PDAs. The most successful PDA operating system has been PalmOS for the Palm Pilot. However Microsoft's Windows CE, lately dubbed Pocket PC, has lately been maturing much faster than PalmOS, and runs on most of today's PDAs.

Linux has been ported to the same set of platforms on which Windows CE runs. The difference between Linux and Windows here is that, whereas Windows CE is a new OS from Microsoft, a "Mini-ME" as it were, Linux is Linux. The large gap between Windows CE and Windows XP has no counterpart with Linux, whose only limitations on small platforms are those imposed by the limited resources of small handheld devices.

This uniformity among PDA platforms gives them much of the feel of a single valley. While there is no binary compatibility between them, this is largely transparent to the users, who perceive a single glacier running through a single valley.

It is however a small glacier. While Linux is a rapidly growing phenomenon it has not yet caught up with Microsoft in sheer volume of x86 software, whence any port of Linux to another valley such as the valley of the PDAs lacks the impressive volume of the software glacier riding the x86 valley.

The same is true of Windows CE. Even with Pocket PC 2002 to be released a month from this writing, the body of software available for the Windows CE platform is still miniscule compared to that for the x86.

Why not simply port the world's x86 software to PDA valley?

The problem is with the question-begging "simply." It is not simple to port software much of which evolved from a more primitive time. (Fortunately for Linux users, essentially the whole of Linux evolved under relatively enlightened

circumstances, greatly facilitating its reasonably successful port to PDA valley.) It would take years to identify and calm down all the new bugs such a port would introduce. Furthermore it wouldn't fit into the limited confines of a PDA.

Now these confines are those of a 1996 desktop, and certainly there was already a large glacier of software for the x86 which fitted comfortably into those parameters back then. Why not just port that software?

The problem with this scenario is that software has gone places in the interim, greatly encouraged and even stimulated by the rapidly expanding speed and storage capacity of modern desktops and notebooks. To port 1996 software to PDA valley would entail rolling back not just the excesses of modern software but also half a decade of bug fixes and new features. A glacier cannot be selectively torn into its good and bad halves. With the glitzy new accessories of today's software comes geologically recent porcine fat that has been allowed to develop unchecked, along with geologically older relics of the software of two or more decades ago. Liposuction is not an option with today's software: the fat is frozen into the glacier.

Nowhere has this descent into dissipation been more visible than with the evolution of Windows ME and 2000 from their respective roots in Windows 3.1 and NT. (The emphasis is on "visible" here: this scenario has played out elsewhere, just not as visibly.) Whereas 16 MB was adequate for RAM in 1994, now 128 MB is the recommended level. Furthermore the time to boot up and power down has increased markedly.

The one PDA resource that is not currently in short supply, at least for PDAs with a PCMCIA slot such as HP-Compaq's iPAQ, is the hard drive. Toshiba and Kingston have been selling a 2 GB Type II (5 mm thick) PCMCIA hard drive for several months, and Toshiba has just announced a 5 GB version. Here the exponential growth of storage capacity has drawn well clear of the linear production of the world's software, which stands no chance now of ever catching up (except perhaps for those very few individuals with a need and budget for ten or more major desktop applications on a single PDA). However space on a PDA for all one's vacation movies, or Kmart's complete assets and accounts receivable database, will remain tight for the next two to four years.

Windows XP, aka NT 6.0, is giving signs of greater sensitivity to these concerns. XP Embedded is not Windows CE but rather XP stripped for action in tight quarters. And Microsoft distributes an impressive library of advice on shortening XP boot time under its OnNow initiative. Unfortunately the applications a power XP user is likely to run are unlikely to show as much sensitivity and shed their recently acquired layers of fat in the near future.

With these considerations in mind, the prospects for Windows CE/Pocket PC look very good for 2002, and perhaps even 2003. But the exponential growth of hardware is not going to stop then. We leave it to the reader to speculate on the likely PDA hardware and software picture in 2004.

2 Quantum Planet

2.1 Quantum Computation

Quantum computation (QC) is a very important and currently hot theoretical computer science topic.

The practical significance of QC is less clear, due to a gap of approximately two orders of magnitude between the largest quantum module of any kind we can build today and the smallest error-correcting modules currently known from which arbitrarily large quantum computers can be easily manufactured. This gap is presently closing so slowly that is impossible to predict today whether technology improvements will accelerate the closing, or unanticipated obstacles will emerge to slow it down or even stop it altogether on new fundamental grounds that will earn some physicist, quite possibly even one not yet born, a Nobel prize. If this gap turns out to be unclosable for any reason, fundamental or technological, QC as currently envisaged will remain a theoretical study of little more practical importance than recursion theory.

The prospect of effective QC is as remote as manned flight to distant planets. The appropriate metaphorical location for QC then is not a city or even a distant country but another planet altogether, suggesting the name Quantum Planet.

On the upper-bound side, there are depressingly few results in theoretical QC that have any real significance at all. The most notable of these is P. Shor's extraordinary quantum polynomial-time factoring algorithm and its implications for the security of number-theoretic cryptography [15].

There has been some hope for quantum polynomial-time solutions to all problems in NP. One approach to testing membership in an NP-complete set is via an algorithm that converts any classical (deterministic or probabilistic) algorithm for testing membership in *any* set into a faster quantum algorithm for membership in that set. Grover has given a quadratic quantum speedup for any such classical algorithm [8].

On the lower-bound side, several people have shown that Grover's speedup is optimal. But all that shows about membership in an NP-complete set is that a good quantum algorithm based on such an approach has to capitalize somehow on the fact that the set is in NP. Any method that works for the more general class of sets treatable by Grover's algorithm cannot solve this problem on its own.

We would all like to see QC turn out to be a major planet. For now it is turning out to be just a minor asteroid. I see this as due to the great difficulty people have been experiencing in getting other good QC results to compare with Shor's.

2.2 Quantum Engineering

There is a saying, be careful what you wish for, you may get it. While computer scientists are eagerly pursuing quantum computation with its promise of exponentially faster factoring, computer engineers are nervously anticipating a

different kind of impact of quantum mechanics on computation. At the current rate of shrinkage, transistors can expect to reach atomic scale some time towards the end of this century's second decade, i.e. before 2020 AD. (So with that timetable, as Quantum planets go, Quantum Engineering is Mars to Quantum Computation's Neptune.)

As that scale is approached the assumptions of classical mechanics and classical electromagnetism start to break down. Those assumptions depend on the law of large numbers, which serves as a sort of information-theoretic shock-absorber smoothing out the bumps of the quantum world. When each bit is encoded in the state of a single electron, the behavior of that bit turns from classical to quantum. Unlike their stable larger cousins, small bits are both fickle and inscrutable.

Fickle. In the large, charge can leak off gradually but it does not change dramatically (unless zapped by an energetic cosmic ray). In the small however, electrons are fickle: an electron can change state dramatically with no external encouragement. The random ticks of a Geiger counter, and the mechanism by which a charged particle can tunnel through what classically would have been an impenetrable electrostatic shield, are among the better known instances of this random behavior.

Inscrutable. Large devices behave reasonably under observation. A measurement may perturb the state of the device, but the information gleaned from the measurement can then be used to restore the device to prior state. For small devices this situation paradoxically reverses itself. Immediately after observing the state of a sufficiently small device, one can say with confidence that it is currently in the state it was observed to be in. What one cannot guarantee however is that the device was in that state *before* the measurement. The annoying thing is that the measurement process causes the device to first change state at random (albeit with a known contingent probability) and then to report not the old state but the new! The old state is thereby lost and cannot be reconstructed with any reasonable reliability. So while we have reliable reporting of current state, we do not have reliable memory of prior state.

So while quantum computation focuses on the opportunities presented by the strangeness of the quantum world, the focus of quantum engineering is on its challenges. Whereas quantum computation promises an exponential speedup for a few problems, quantum engineering promises to undermine the reliability of computation.

Quantum engineering *per se* is by no means novel to electrical engineers. Various quantum effects have been taken advantage of over the years, such as Goto's tunnel diode [6], the Josephson junction [9], and the quantum version of the Hall effect [16]. However these devices achieve their reliability via the law of large numbers, as with classical devices, while making quantum mechanics work to the engineer's advantage. Increasing bit density to the point where there are as many bits as particles makes quantum mechanics the adversary, and as such very much a novelty for electrical engineers.

3 Concurrency Frontier

Parallel computation is alive and well in cyberspace. With hundreds of millions of computers already on the Internet and millions more joining them every day, any two of which can communicate with each other, it is clear that cyberspace is already a highly parallel universe.

It is however not a civilized universe. Concurrency has simply emerged as a force of cybernature to be reckoned with. In that sense concurrency remains very much a frontier territory, waiting to be brought under control so that it can be more efficiently exploited.

Cyberspace is still very much a MIMD world, Multiple Instructions operating on Multiple Data elements. It is clearly not a SIMD world, the science fiction scenario of a single central intelligent agent controlling an army of distributed agents.

Nevertheless some SIMD elements are starting to emerge, in which the Internet's computers act in concert in response to some stimulus. The Y2K threat was supposed to be one of these, with all the computers in a given time zone misbehaving at the instant the year rolled over to 2000 in that time zone.

The fact that most computers today in the size spectrum between notebooks and desktops run Microsoft Windows has facilitated the development of computer viruses and worms. A particularly virulent worm is the recent W32.Sircam, which most computer users will by now have received, probably many times, as a message beginning "I send you this file in order to have your advice." If it takes up residence on a host, this worm goes into virus mode under various circumstances, one of which is the host's date being October 16. In this mode it deletes the C: drive and/or fills available space by indefinitely growing a file in the Recycled directory. As of this writing (September) it remains to be seen whether this instance of synchronicity will trigger a larger cyberquake than the mere occurrence of the year 2000.

More constructive applications of the SIMD principle have yet to make any substantial impact on the computing milieu. However as the shrinking of device geometries continues on down to the above-mentioned quantum level and sequential computation becomes much harder to speed up, computer architects will find themselves increasing the priority of massively parallel computation, not for just a handful of CPUs but for thousands and even millions of devices acting in concert.

Thinking Machines Corporation's CM-2 computer was an ambitious SIMD machine with up to 64K processors. Going by processor count, this is impressive even today: then it yielded a local bus bandwidth of 40 GB/s, and with today's faster buses could be expected to move terabytes per second locally. However main memory was 0.5 GB while hard drive storage ran to 10 GB, capacity that today can be matched by any hobbyist for $150. The blazing speed was definitely the thing.

SIMD shows up on a smaller scale in Intel's 64-bit-parallel MMX architecture and 128-bit SSE architecture, as well as in AMD's 128-bit 3DNow architecture.

However this degree of parallelism yields only small incremental gains sufficient to edge ahead of the competition in the ongoing speed wars.

As some of the more fundamental bottlenecks start showing up, evasive maneuvers will become more necessary. A return to CM-2 scale parallelism and higher is certainly one approach.

A major difficulty encountered at Thinking Machines was how to describe the coordinated movement and transformation of data in such highly parallel machines.

Rose and Steele [14] and Blelloch and Greiner [1] came up with some novel programming language approaches to Thinking Machines' problem. What struck me about these approaches was how hard it is to separate ourselves from the sequential modes of thought that pervade our perception of the universe. It is as though we cannot *see* pure concurrency. If we were ever confronted with it we simply would not recognize it as computation, since it would lack those sequential characteristics that characterize certain behaviors for us as essentially computational, in particular events laid out along a time line.

Programmers who would like to leave a useful legacy to their heirs need to spend more time reflecting on the nature of concurrency, learning what it feels like and how to control it. My own view of concurrency is that event structures [10,17,18] offer a good balance of abstractness and comprehensiveness in modeling concurrency.

The extension of event structures to Chu spaces, or couples as I have started calling them [12], simultaneously enriches the comprehensiveness while cleaning up the model to the point where it matches up to to Girard's linear logic [4]. The match-up is uncannily accurate [2] given that linear logic was not intended at all as a process algebra but as a structuring of Gentzen-style proof theory.

There is another ostensibly altogether different approach to the essence of concurrency, that of higher dimensional automata [11,5,3,7]. It is however possible to reconcile this approach with the Chu space approach by working with couples over 3, that is, a three-letter alphabet for the basic event states of *before*, *during*, and *after* [13]. It is my belief that the three-way combination of duality-based couples, geometry-based higher-dimensional automata, and logic-based linear logic, provides a mathematically richer yet simpler view of concurrency than any other approach.

The economic promise of Concurrency Frontier is its potential for further extending the power of computers when the limits set by the speed of light and quantum uncertainty bring the development of sequential computation to a halt. I do not believe that the existing sequentiality-based views of concurrency permit this extension, and that instead we need to start understanding the interaction of complex systems in terms of the sorts of operations physicists use to combine Hilbert spaces, in particular tensor product and direct sum. The counterparts of these operations for concurrency are respectively orthocurrence $A \otimes B$ and concurrence $A \| B$ (also $A \oplus B$), or *tensor* and *plus* as they are called in linear logic. Closer study of this point of view will be well rewarded.

References

1. G. Blelloch and J. Greiner. Parallelism in sequential functional languages. In *Proc. Symposium on Functional Programming and Computer Architecture*, pages 226–237, June 1995.
2. H. Devarajan, D. Hughes, G. Plotkin, and V. Pratt. Full completeness of the multiplicative linear logic of Chu spaces. In *Proc. 14th Annual IEEE Symp. on Logic in Computer Science*, pages 234–243, Trento, Italy, July 1999.
3. L. Fajstrup, E. Goubault, and M. Raussen. Detecting deadlocks in concurrent systems. In *Proc. of CONCUR'98*, volume 1466 of *Lecture Notes in Computer Science*, pages 332–347. Springer-Verlag, 1998.
4. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
5. E. Goubault and T.P. Jensen. Homology of higher dimensional automata. In *Proc. of CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*, pages 254–268, Stonybrook, New York, August 1992. Springer-Verlag.
6. E. Goto *et al.* Esaki diode high speed logical circuits. *IRE Trans. Elec. Comp.*, EC-9:25–29, 1960.
7. E. (editor) Goubault. Geometry and concurrency. *Mathematical Structures in Computer Science, special issue*, 10(4):409–573 (7 papers), August 2000.
8. L. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79(2), 1997.
9. B. Josephson. Possible new effects in superconductive tunnelling. *Physics Letters*, 1:251–253, 1962.
10. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures, and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.
11. V.R. Pratt. Modeling concurrency with geometry. In *Proc. 18th Ann. ACM Symposium on Principles of Programming Languages*, pages 311–322, January 1991.
12. V.R. Pratt. Chu spaces and their interpretation as concurrent objects. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 392–405. Springer-Verlag, 1995.
13. V.R. Pratt. Higher dimensional automata revisited. *Math. Structures in Comp. Sci.*, 10:525–548, 2000.
14. J. Rose and G. Steele. C*: An extended c language for data parallel programming. In *Proceedings Second International Conference on Supercomputing*, volume 2, pages 2–16, May 1987.
15. P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Computing*, 26:1484–1509, 1997.
16. K. von Klitzing, G. Dorda, and M. Pepper. New method for high-accuracy determination of the fine-structure constant based on quantized hall resistance. *Physical Review Letters*, 45(6):494–497, 1980.
17. G. Winskel. *Events in Computation.* PhD thesis, Dept. of Computer Science, University of Edinburgh, 1980.
18. G. Winskel. An introduction to event structures. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, REX'88*, volume 354 of *Lecture Notes in Computer Science*, Noordwijkerhout, June 1988. Springer-Verlag.

Author Index